Package 'lame'

November 5, 2026

Title Longitudinal Additive and Multiplicative Effects Models for Networks

Version 1.4.1

Description Additive and multiplicative effects models for both cross-sectional and longitudinal network analysis. The package provides two main functions: ame() for cross-sectional networks and lame() for longitudinal networks. It supports square and rectangular network structures. Key features include: (1) Cross-sectional network analysis via ame() with support for binary, continuous, ordinal, and count data; (2) Longitudinal network analysis via lame() with additive sender/receiver and multiplicative latent-factor effects that can evolve over time through AR(1) processes (Sewell and Chen (2015) <doi:10.1080/01621459.2014.988214>; Durante and Dunson (2014) <doi:10.1093/biomet/asu040>); (3) Handling of changing actor compositions across time periods in longitudinal models; (4) Performance improvements through C++ implementations via 'Rcpp' and 'RcppArmadillo'.

Author Cassy Dorff [aut],
Shahryar Minhas [aut, cre],
Tosin Salau [aut]

Maintainer Shahryar Minhas <minhassh@msu.edu>

License MIT + file LICENSE

URL https://netify-dev.github.io/lame/,
https://github.com/netify-dev/lame

BugReports https://github.com/netify-dev/lame/issues

Depends R (>= 3.5.0)

Encoding UTF-8

LazyData true

Config/roxygen2/version 8.0.0

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, pbivnorm, ggplot2, ggrepel, ggforce, gridExtra, coda, patchwork, cli, MASS, Matrix, abind, netify (>= 1.5.3), graphics, grDevices, parallel, stats, utils

Suggests knitr, rmarkdown, testthat (>= 3.1.7), withr, igraph, network, posterior, loo, digest, tibble, broom, generics, pROC, precrec, statmod, dplyr, modelsummary, callr, amen

VignetteBuilder knitr

Config/testthat/edition 3

SystemRequirements C++17

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-11-05 15:00:02 UTC

Contents

ab_plot . . . 5
ab_plot.ame_als . . . 7
addhealthc3 . . . 8
addhealthc9 . . . 9
als_dynamic_beta . . . 9
als_start_vals . . . 11
ame . . . 12
ame_als . . . 30
ame_als_bootstrap . . . 55
ame_als_refit . . . 61
ame_memory_settings . . . 63
ame_memory_usage . . . 64
ame_options . . . 65
ame_parallel . . . 66
array_to_list . . . 67
as_draws . . . 68
as_draws.ame . . . 69
as_lame_y . . . 70
autoplot.ame_als . . . 71
autoplot.lame . . . 71
boot_ame-no-fitted . . . 73
check_format . . . 73
coef.als_dynamic_beta . . . 75
coef.ame . . . 75
coef.ame_als . . . 76
coef.boot_ame . . . 77
coldwar . . . 77
combine_ame_chains . . . 78
compact_ame . . . 79
compute_mcmc_diagnostics . . . 80
comtrade . . . 81
confint.ame . . . 81
confint.ame_als . . . 82
confint.boot_ame . . . 83

detect_change_point . . . 84
dutchcollege . . . 86
dynamic_beta_prior_summary . . . 87
el2sm . . . 88
evaluate_heldout . . . 89
fitted.ame . . . 90
fitted.ame_als . . . 91
fitted.lame . . . 92
forecast_pit . . . 93
formula.ame . . . 95
glance . . . 95
glance.ame . . . 96
glance.ame_als . . . 97
gof . . . 98
gof_plot . . . 100
gof_plot.ame_als . . . 103
gof_stats . . . 104
gof_stats_bipartite . . . 106
gof_stats_unipartite . . . 107
gof_temporal . . . 108
init_dynamic_ab_cpp . . . 110
IR90s . . . 111
lame . . . 111
lame_als . . . 154
lame_multi . . . 159
lame_parallel . . . 161
lame_resume . . . 162
lame_robust . . . 163
lame_snap_als . . . 166
latent_positions . . . 167
lazegalaw . . . 169
lfo . . . 170
list_to_array . . . 171
logLik.ame . . . 172
logLik.ame_als . . . 173
loo . . . 173
loo.ame . . . 174
mhalf . . . 175
nobs.ame . . . 176
nobs.ame_als . . . 177
nodematch . . . 177
per_actor_slopes . . . 178
plot.ame . . . 179
plot.ame_als . . . 180
plot.lame . . . 181
posterior_options . . . 182
posterior_quantiles . . . 183
predict.ame . . . 184

predict.ame_als . . . 186
predict.lame . . . 187
predict.lame_robust . . . 189
prediction_draws_long . . . 191
print.als_dynamic_beta . . . 192
print.ame . . . 193
print.ame.sim . . . 194
print.ame_als . . . 195
print.boot_ame . . . 195
print.gof_temporal . . . 196
print.lame . . . 196
print.lame_multi . . . 197
print.lfo_lame . . . 197
print.per_actor_slopes . . . 198
print.summary.ame . . . 198
print.summary.ame_als . . . 199
print.summary.lame . . . 199
prior_summary . . . 200
procrustes_align . . . 201
rbeta_ab_bip_gibbs_cpp . . . 203
read_log_lik . . . 204
reconstruct_EZ . . . 204
residuals.ame . . . 206
residuals.ame_als . . . 207
residuals.lame . . . 207
rhat_dynamic_beta . . . 208
rmvnorm . . . 209
rUV_dynamic_bip_fc_cpp . . . 210
rUV_dynamic_fc_cpp . . . 211
rUV_dynamic_snap_fc_cpp . . . 212
rUV_dynamic_t_fc_cpp . . . 213
rUV_sym_fc . . . 214
rZ_bin_bip_batch_cpp . . . 215
rZ_nrm_batch_cpp . . . 215
sampler_describe . . . 216
sample_beta_dynamic_cpp . . . 217
sample_beta_static_cpp . . . 218
sample_dynamic_ab_cpp . . . 219
sample_rho_beta_cpp . . . 221
sample_sigma_beta_cpp . . . 222
sampsonmonks . . . 222
sheep . . . 223
simulate.ame . . . 223
simulate.ame_als . . . 225
simulate.lame . . . 226
simulate_posterior . . . 229
snap_category_summary . . . 230
snap_index_draws . . . 231

snap_index_summary . . . 232
snap_rank_summary . . . 233
summary.ame . . . 234
summary.ame.sim . . . 235
summary.ame_als . . . 236
summary.boot_ame . . . 236
summary.lame . . . 237
summary.lame.sim . . . 239
summary.lame_robust . . . 240
tidy . . . 241
tidy.ame . . . 242
tidy.ame_als . . . 243
tidy.boot_ame . . . 244
trace_plot . . . 244
update.ame . . . 247
update.ame_als . . . 248
uv_plot . . . 249
vcov.ame . . . 251
vcov.ame_als . . . 252
vcov.boot_ame . . . 253
vignette_data . . . 254
waic.ame . . . 255
Xbeta . . . 255
Xcol . . . 256
Xdyad . . . 256
Xrow . . . 257
Y . . . 257
YX_bin . . . 257
YX_bin_list . . . 258
YX_bin_long . . . 258
YX_cbin . . . 259
YX_frn . . . 259
YX_nrm . . . 260
YX_ord . . . 260

Index **262**

`ab_plot` *Visualize sender and receiver random effects*

Description

Creates a visualization of the additive sender (row) and receiver (column) random effects from an AME or LAME model. Automatically detects whether effects are static or dynamic and provides appropriate visualization options.

Usage

```
ab_plot(
  fit,
  effect = c("sender", "receiver", "both"),
  sorted = TRUE,
  labels = NULL,
  title = NULL,
  time_point = NULL,
  plot_type = c("snapshot", "trajectory", "faceted", "ribbon"),
  show_actors = NULL
)
```

Arguments

fit	An object of class "ame" or "lame" from fitting an AME model
effect	Character string specifying which effect to plot: "sender" (default), "receiver", or "both". "both" draws the sender and receiver effects as two panels, each sorted on its own; for dynamic fits it is available in the snapshot views (plot_type = "snapshot", one period or time_point = "average"). Symmetric fits store a single set of actor effects, so only "sender" applies to them.
sorted	Logical; if TRUE (default), actors are sorted by effect magnitude. Applies to ame / lame fits; ame_als fits are always sorted by value.
labels	Logical; if TRUE, actor labels are shown on x-axis (default TRUE for n <= 50 actors). Applies to ame / lame fits.
title	Optional title for the plot (ame / lame fits).
time_point	For dynamic effects, which time point to plot (default: last). Can be a numeric index, "all" for a faceted plot, or "average" for time-averaged
plot_type	For dynamic effects: "snapshot" (single time), "trajectory" (evolution over time), "faceted" (grid of time points), or "ribbon" (effect path with a 95\ band per period). For static effects, this parameter is ignored.
show_actors	Character vector of specific actors to highlight (for dynamic trajectory / ribbon plots)

Details

The additive effects in AME models represent:

Sender effects (a) Actor-specific tendencies to form outgoing ties. Positive values indicate actors who send more ties than expected; negative values indicate actors who send fewer ties.

Receiver effects (b) Actor-specific tendencies to receive incoming ties. Positive values indicate actors who receive more ties than expected; negative values indicate actors who receive fewer ties.

For static effects, the plot displays these effects as a dot plot with vertical lines extending from zero to each effect estimate.

For dynamic effects (when fit contains a_dynamic/b_dynamic), additional options are available to visualize how effects evolve over time.

Value

A ggplot2 object that can be further customized

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Fit an AME model
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE, chains = 1)

# Visualize sender effects
ab_plot(fit, effect = "sender")

# Visualize receiver effects without sorting
ab_plot(fit, effect = "receiver", sorted = FALSE)
```

---

ab_plot.ame_als *Additive-effects plot for an ame_als fit*

Description

Sender (a) and receiver (b) additive-effect point estimates as a sorted lollipop chart. Mirrors ab_plot but without posterior intervals: this is a point estimator. If the fit was produced via ame_als(..., bootstrap = N), bootstrap 95\ error bars.

Usage

```
ab_plot.ame_als(
  fit,
  effect = c("sender", "receiver", "both"),
  top_n = Inf,
  ...
)
```

Arguments

fit	an ame_als fit.
effect	"sender" (default), "receiver", or "both".
top_n	integer; show only the top / bottom top_n actors by absolute effect (default Inf, show all).
...	reserved.

Value

A ggplot object.

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
fit <- ame_als(Y, R = 1, family = "normal", verbose = FALSE)
ab_plot(fit, effect = "sender")
```

---

addhealthc3 *AddHealth community 3 data*

Description

A valued sociomatrix (Y) and matrix of nodal attributes (X) for students in community 3 of the AddHealth study.

- Y: A sociomatrix in which the value of the edge corresponds to an ad-hoc measure of intensity of the relation. Note that students were only allowed to nominate up to 5 male friends and 5 female friends.
- X: Matrix of students attributes, including sex, race (1=white, 2=black, 3=hispanic, 4=asian, 5=mixed/other) and grade.

Originally available at http://moreno.ss.uci.edu/data.html#adhealth (site no longer accessible).

Usage

```
data(addhealthc3)
```

Format

list

addhealthc9 *AddHealth community 9 data*

Description

A valued 254 x 254 sociomatrix (Y) and a matrix of nodal attributes (X) for students in community 9 of the AddHealth study.

- Y: A sociomatrix in which the value of the edge corresponds to an ad-hoc measure of intensity of the relation. Note that students were only allowed to nominate up to 5 male friends and 5 female friends.
- X: A 254 x 3 matrix of student attributes, one row per actor of Y in the same order, including sex, race (1=white, 2=black, 3=hispanic, 4=asian, 5=mixed/other) and grade.

Originally available at `http://moreno.ss.uci.edu/data.html#adhealth` (site no longer accessible).

Usage

```
data(addhealthc9)
```

Format

list

als_dynamic_beta *Penalised ALS time-varying coefficient estimate*

Description

Computes a fast point estimate of a time-varying coefficient vector β_t for a longitudinal network model by solving the first-difference-penalised least-squares problem. No MCMC, no multiplicative effects, no random effects – purely a regression point estimate with a smoothing penalty on the coefficient path. Useful for rapid exploration and for generating starting values for a full `lame(dynamic_beta = ...)` fit.

Usage

```
als_dynamic_beta(Y, Xdyad, lambda = 0, intercept = TRUE)
```

Arguments

`Y`	A list of T response matrices (or a 3-D array with third dimension time).
`Xdyad`	A list of T dyadic covariate arrays (each $n \times n \times p$, or a single $n \times n$ matrix if $p = 1$).
`lambda`	Non-negative smoothing parameter. Default `0` (no penalty, per-period OLS).
`intercept`	Logical; include an intercept (default `TRUE`).

Details

$$\min_{\beta_{1:T}} \sum_t \|y_t - X_t\beta_t\|^2 + \lambda \sum_{t=2}^{T} \|\beta_t - \beta_{t-1}\|^2$$

At $\lambda = 0$ this is T independent per-period OLS fits. As $\lambda \to \infty$ the solution converges to a single pooled β (constant over time).

Value

A list with `beta` (a $p \times T$ matrix of estimates; rows `"intercept"` and then the names of the covariate slices of `Xdyad`, or `"x1"`, `"x2"`, ... when the slices are unnamed; columns `"t1"`, ..., `"tT"`), `lambda`, `residual_ss` (sum of squared residuals across periods), `intercept`, and `call`. Class `"als_dynamic_beta"`.

Relation to other entry points

`als_dynamic_beta` is a *regression-only smoother*; the returned object carries time-varying β_t alone, no additive (a, b) or multiplicative (U, V) AME components. It is therefore not a special case of `lame_als` (which always estimates the full AME decomposition) and not a special case of `lame(dynamic_beta = TRUE)` (which is Bayesian with an AR(1) / RW1 / RW2 / Matern 3/2 state-space prior). Use it when you want a fast, point-only, deterministic smoother for the coefficient path; use `lame(dynamic_beta = ...)` when you need full posterior uncertainty and additive / multiplicative effects.

See Also

`lame` for the full Bayesian dynamic-β fit (AR(1) / RW1 / RW2 / Matern 3/2); `lame_als` for the longitudinal AME point estimator (static β, full a, b, U, V).

Examples

```
set.seed(1)
n <- 10; Tn <- 4
X <- replicate(Tn, array(rnorm(n*n*2), c(n, n, 2)), simplify = FALSE)
beta_true <- rbind(seq(-1, 1, length.out = Tn), seq(0.5, -0.5, length.out = Tn))
Y <- vector("list", Tn)
for (t in seq_len(Tn)) {
  Yt <- X[[t]][, , 1] * beta_true[1, t] + X[[t]][, , 2] * beta_true[2, t] +
        matrix(rnorm(n*n, 0, 0.2), n, n)
  diag(Yt) <- NA
  Y[[t]] <- Yt
}
fit_als <- als_dynamic_beta(Y, X, lambda = 0)     # per-period LS
fit_smooth <- als_dynamic_beta(Y, X, lambda = 10) # smoother path
```

als_start_vals *Convert an ALS fit to MCMC starting values*

Description

Builds a start_vals list for ame or lame from an ALS point estimate. This is useful when the ALS fit has already found a good latent-space solution and the MCMC chain should start near that solution rather than from diffuse random values.

Usage

```
als_start_vals(fit, jitter = 0, seed = NULL)
```

Arguments

fit	a fitted ame_als, lame_als, or dynamic ALS object returned by lame with method = "als".
jitter	non-negative standard deviation for independent Gaussian perturbations added to numeric starting values. Use a small positive value for multiple MCMC chains.
seed	optional integer seed used for the jitter.

Value

A list that can be passed to start_vals. For a dynamic ALS fit it also carries rho_uv, sigma_uv, rho_ab, and sigma_ab, so the MCMC run starts its AR(1) hyperparameters from the ALS solution. The dyadic rho start is the single pooled correlation in fit$VC["rho"], matching ALS simulation. Period-specific rho_path / residual_rho_path values are residual diagnostics, including in legacy objects labelled dynamic_rho = TRUE, and are not used as covariance-path starting values. A MCMC model requesting dynamic_rho = TRUE expands this scalar initial value; explicit MCMC path starts remain available through its start_vals argument. For a penalised dynamic fit (bias_correct = "none" or "jackknife") read these as starting values rather than estimates: the persistences are the values the ALS smoothness penalty used (the prior means unless prior set them), and the innovation scales are implied by the fitted paths, which the penalty shrinks, so they run below the posterior scale. For a dynamic empirical-Bayes fit (the default for normal, binary and Poisson panels) they are estimates under the lame() hyperpriors. For a binary fit the intercept is shifted so that the implied tie rate, mean(pnorm(EZ)), equals the observed one, with everything else held at the ALS values; the shift is stored in attr(, "level_shift"). The ALS intercept is a working-response level: under the default IRLS it sits within a few hundredths of the level the sampler has to reach, and under non_normal_method = "transform" it can sit a probit unit or more above it, the sampler's slowest direction.

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
```

```
fit_als <- ame_als(Y, R = 1, family = "normal", verbose = FALSE)
sv <- als_start_vals(fit_als)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit_mcmc <- ame(Y, R = 1, start_vals = sv,
                nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
```

---

ame *AME model fitting routine*

Description

An MCMC routine providing a fit to an additive and multiplicative effects (AME) regression model to cross-sectional relational data of various types. This function supports both unipartite (square) and bipartite (rectangular) networks. For longitudinal networks, use the `lame` function. The additive and multiplicative effects framework is due to Hoff (2005, 2021).

Usage

```
ame(
  Y,
  Xdyad = NULL,
  Xrow = NULL,
  Xcol = NULL,
  rvar = TRUE,
  cvar = TRUE,
  dcor = !symmetric,
  nvar = TRUE,
  R = 0,
  R_row = NULL,
  R_col = NULL,
  mode = c("unipartite", "bipartite"),
  family = "normal",
  intercept = !(family == "ordinal"),
  symmetric = FALSE,
  odmax = rep(max(apply(Y > 0, 1, sum, na.rm = TRUE)), nrow(Y)),
  prior = list(),
  g = NA,
  seed = 6886,
  nscan = 10000,
  burn = 500,
  odens = 25,
  verbose = TRUE,
  gof = TRUE,
  custom_gof = NULL,
  plot = FALSE,
  start_vals = NULL,
  periodic_save = FALSE,
```

```
  out_file = NULL,
  save_interval = 0.25,
  posterior_opts = NULL,
  n_chains = 1,
  cores = NULL,
  chains = .lame_chains_default(),
  use_sparse_matrices = FALSE,
  method = c("mcmc", "als"),
  bootstrap = 0L,
  bootstrap_type = c("parametric", "block"),
  bootstrap_block_length = 1L,
  bootstrap_seed = NULL,
  multistart = c("none", "cheap", "full"),
  save_log_lik = FALSE,
  ordinal_cutpoints = c("data_induced", "explicit"),
  print,
  ...,
  log_lik_method = c("observed_exact", "observed_approx"),
  model.name = NULL,
  bias_correct = NULL
)
```

Arguments

`Y`	For unipartite: an n x n square relational matrix. For bipartite: an nA x nB rectangular relational matrix where nA is the number of row nodes and nB is the number of column nodes. A cross-sectional **netify** object is also accepted and converted with `netify::to_lame()`; when `family` or `mode` is omitted, the value inferred by **netify** is used. See family below for data types.
`Xdyad`	For unipartite: an n x n x pd array of dyadic covariates (e.g. distance, shared group). For bipartite: an nA x nB x pd array. A 2-D matrix (single dyadic covariate) must be wrapped as `array(x, dim = c(n, n, 1))`; `Inf`/`NaN` entries are rejected.
`Xrow`	For unipartite: an n x pr matrix of **sender** (row) covariates (e.g. sender's age, group). For bipartite: an nA x pr matrix of row-node covariates. A `data.frame` is accepted and coerced to numeric matrix internally.
`Xcol`	For unipartite: an n x pc matrix of **receiver** (column) covariates. For bipartite: an nB x pc matrix of column-node covariates. A `data.frame` is accepted and coerced internally.
`rvar`	logical: fit row random effects (asymmetric case)?
`cvar`	logical: fit column random effects (asymmetric case)?
`dcor`	logical: fit a dyadic correlation (asymmetric case)? Note: not used for bipartite networks. With `method = "als"` the argument goes to `ame_als`, whose directed binary probit, Poisson or normal fit then has the correlation in its likelihood.
`nvar`	logical: fit nodal random effects (symmetric case)?

R — integer: dimension of the multiplicative effects (can be zero). For bipartite networks, this is used as the default for both R_row and R_col if they are not specified. A binary fit whose multiplicative term reproduces the observed ties exactly is separated: the size of the term is then set by its prior, the fit says so, and the signs and the fitted probabilities are what the data determine; a smaller R (down to zero for a panel whose ties barely change over time) reproduces them without saturating.

R_row — integer: for bipartite networks, dimension of row node multiplicative effects (defaults to R)

R_col — integer: for bipartite networks, dimension of column node multiplicative effects (defaults to R)

mode — character: either "unipartite" (default) for square networks or "bipartite" for rectangular networks. Not all combinations of family and mode are supported – see the table under **Supported family x mode combinations** below.

family — character: one of "normal","binary","ordinal","cbin","frn","poisson". See the **Supported family x mode combinations** table below for which combinations are valid; see **Details** for the model assumptions behind each family.

intercept — logical: fit model with an intercept?

symmetric — logical: Is the sociomatrix symmetric by design?

odmax — a scalar integer or vector of length n giving the maximum number of nominations that each node may make - used for "frn" and "cbin" families. A cap below a row's observed out-degree contradicts Y and is an error. A cap that most rows' observed outdegrees reach erases the cross-row spread in outdegree that the sender effects rest on: a row at its cap shows only that its level is high enough to fill the cap. The sampler pools the prior scale of the row side with the column side once 80 percent of the observed rows (60 percent in a single period) nominate the full odmax, counting a row with hidden cells by what a full cap thins to at its observed share; fit$odmax_saturated_share records the share and fit$odmax_sat_threshold the threshold. va, vb and cab are reported at every share: on simulated panels of 24 to 30 actors their 95 percent intervals covered the generating values in 0.88 to 1.00 of the panels at every share, with and without a fifth of the cells hidden and at generating sender sds of 0.25, 0.5 and 1 (at 0.25 the posterior mean of va runs up to 2.3 times the generating variance while the interval still covers). va and the level of the rows follow the design. A design of two periods or more with a cap share of 0.5 or more and a posterior-mean va from 0.5 to 1.5 reports the intercept and the row-covariate coefficients as not estimated: the rows below the cap carry too little of the spread of the senders and the posterior of va sits near its prior. In simulated panels (24 actors, 4 and 6 periods, one chain at the package defaults, 1440 fits at generating sender sds of 0.5, 1 and 1.5) the intercept was covered in 0.89 of the 312 panels in that region (0.76 at sd 0.5 and 1.5, 0.97 at sd 1) and in 0.96 of the 1128 outside it. va is reported in the region: its interval covered 0.91 of the 312 panels (0.79 at sd 0.5, 1.00 at sd 1, 0.72 at sd 1.5) and 0.99 outside it. The posterior mean of va ran 0.18 and 0.23 above the generating variance at sd 0.5 (shares 0.3 and 0.7), 0.08 and 0.00 above it at sd 1 and 0.02 and 0.18 below it at sd 1.5. In a single period (ame() and one-period lame() fits) a row at its cap leaves the level of the rows unbounded above: with every row at the cap the intercept interval covered

0.64 of the panels, and with cells hidden and a share from 0.5 it covered 0.84, against 0.98 or more elsewhere. The intercept and any row-covariate coefficient are reported as not estimated there, with the draws kept in `fit$BETA`. On 200 simulated 170 by 100 bipartite panels (cap 10, shares 0.3 and 0.7) the intercept was covered in 1.00 of the panels, `vb` in 0.98 to 1.00 and `va` in 0.90 to 1.00, its posterior mean 0.09 to 0.15 below the generating variance; nothing is withheld there. When the effects prior is supplied (`prior$Sab0`, `eta0` or `etaab`), the fit names the prior in the summary (`fit$prior_supplied` records it); a prior on the effects does not fix the level of a row, so the intercept stays withheld in those designs, and `va` is reported under that prior. A cap the design itself imposes cannot be relaxed: read `APM` / `BPM` as relative sender and receiver levels there.

`prior` a list containing hyperparameters for the prior distributions. Available options and their defaults:

Sab0 Prior scale matrix for the additive-effects covariance. A 2x2 matrix where Sab0\[1,1\] is the prior variance for row effects, Sab0\[2,2\] is the prior variance for column effects, and off-diagonals control correlation between row and column effects. For a unipartite fit this defaults to `diag(2)` scaled to the observed sender/receiver heterogeneity in `Y` (unconditionally for the continuous families, conditionally for the discrete ones); a bipartite fit is scaled the same way, with one scale for both sides (a single period is too little to read the sender side of a capped `"cbin"` or `"frn"` panel on its own, so the scale is read over every row, not only the rows below the cap as for panels; see `Sab0` in `lame`). See the note on the additive-effects variance scale in Details. Pass `diag(2)` explicitly for a fixed unit-scale prior.

eta0 Prior degrees of freedom for the additive-effects covariance Σ_{ab} (default: round(4 + 3 * n/100) for a continuous-family fit, unipartite or bipartite (n is then the number of row actors, as in `lame`), and round(4 * vdfmlt) for a discrete-family fit, where n is the number of actors and vdfmlt is a probit-moment variance multiplier estimated from `Y`). Higher values impose stronger shrinkage of the row/column effects toward the prior scale. The multiplicative-effects degrees of freedom are controlled by `kappa0`, not `eta0`.

etaab Prior degrees of freedom for covariance of additive effects (default: 4 + 3 * n/100). Controls shrinkage of row/column random effects. Larger values shrink effects toward zero. *Used by the bipartite and longitudinal paths; ignored for a unipartite cross-sectional* `ame()` *fit, which controls the additive prior through* `Sab0` *and* `eta0`.

s20 Fixed positive prior scale for the dyadic residual variance v_e. The prior has one degree of freedom. By default, normal fits use the variance of the observed responses, computed once before sampling; constant responses use their mean square, or 1 if all zero. Poisson fits use 1 on the latent log-rate scale. The resolved scale is reported as `fit$s2_prior_scale`. It is never recomputed from the current residuals. A supplied value overrides this default. In a bipartite cross-sectional `ame()` fit, `s20` also multiplies the coefficient prior variance (`g * s20`); its default there is the variance of the response for the normal family (so that the prior is the same in any unit of the response) and 1 on the latent scale of the other families; elsewhere that prior is controlled separately by `g`.

s2u0 Prior variance for multiplicative effects (default: 1). *Used by the bipartite and longitudinal paths.*

Suv0 Inverse-Wishart (inverse) scale matrix for the multiplicative-effects covariance. The prior scale is `kappa0 * Suv0` and the prior mean is `kappa0 * Suv0 / (kappa0 - 2 * R - 1)`. When not supplied it is `s * diag(2 * R)`, s the scale of the additive effects estimated from the data, the one the default `Sab0` reads: for `"binary"`, the variance of the actor means of the standardised residuals $(y-\bar{y})/\phi(\Phi^{-1}(\bar{y}))$ of the less frequent value; for `"ordinal"`, `"cbin"` and `"frn"`, the same probit moments of the ranks; for `"poisson"`, the variance of the centred log row and column rates; for `"normal"`, the square root of the mean variance of the row and column means of the centred response (in the unit of the response, as the product $u_i'v_j$ is). Every period of a `lame` panel enters, and a single period gives the value of `ame()`. `lame` has the same default. A bipartite network has no covariance of this kind (the rows of its factors have a unit-free N(0, I) prior and the interaction matrix carries the scale), and supplying `Suv0` or `kappa0` for one is an error of class `"lame_prior_unused"`.

kappa0 Prior degrees of freedom for the multiplicative-effects covariance Σ_{uv}. The default when `R > 0` is `round((2 * R + 2) * m)`: $2R + 2$, the smallest value for which the prior has a finite mean, times $m = 0.25/\overline{p(1-p)}$, the inverse Bernoulli information of the cells relative to a cell at $p = 1/2$, with p the probit fit of the additive moments, for the `"binary"`, `"cbin"` and `"frn"` families ($m \geq 1$), and $m = 1$ for the others. A sparse network has little information per cell about its factors, and the prior then holds Σ_{uv} near the data scale sI with more pseudo-observations; m depends on the density, not on the number of actors. Higher values impose stronger shrinkage of the latent factors toward the prior scale.

actor_move Logical, default `TRUE`. For `"poisson"` and `"ordinal"` fits (unipartite, directed or symmetric, and bipartite): translate each actor's level together with the latent values of its row and column (an exact draw: the Gaussian density of the latent array is unchanged, so the conditional is the level prior times the Poisson factor, or the interval the ranks or cutpoints allow), draw the location of the latent array (ordinal), and, for the Poisson family, translate the coefficients and move the dyadic variance and correlation together with the latent residuals and redraw the positions with the latent values of their rows and columns. The augmented sweep moves an actor with no events, and the intercept and the spread of the levels with it, by steps of order `sqrt(s2 / number of cells)`, which takes thousands of sweeps on a sparse panel; the updates leave the posterior unchanged. For binary fits (unipartite, directed or symmetric, with any additive margin in the model, a fit with none drawing the positions alone, and bipartite; complete or with missing cells and actors): redraw each actor's level and static position from its exact conditional with the latent array integrated out (an elliptical slice step). The augmented sweep moves an actor whose cells hold no events by a step of order `1 / sqrt(number of cells)` however wide its posterior is, so such an actor relaxes slowly; the update leaves the posterior unchanged. A bipartite fit of any family with an intercept also draws, exactly, along the ridge that the intercept shares with the means of its (un-

centred) additive effects (a shift of the intercept and the opposite shift of the levels changes no predictor, so the draw has no likelihood term). For directed `"cbin"` and `"frn"` fits (unipartite or bipartite), each receiver's level is translated together with its latent column, an exact draw from the prior of the level cut to the interval the order constraints of the column leave (the augmented sweep moves a receiver nobody nominates by the same slow step), the rows free above translate with the intercept against the rows with room, and a directed fit with `dcor = TRUE` updates the dyadic correlation with the non-nominations integrated out (see the Details of `lame`). `FALSE` omits them; the work is recorded in `fit$actor_move`.

chain_extensions Number of times a chain that is still moving at the end of the run is continued before the call is refused (default 3; 0 refuses at the first flag). See the convergence policy.

actor_move_every Number of sweeps between visits of that update (default 10, and 2 for the Poisson translations; 1 for the translations of `"cbin"` and `"frn"` fits, whose reciprocity update runs on every tenth visit, and for the draw along the intercept ridge of bipartite fits of the other families).

uv_g_scale_move Logical, default `TRUE`. For bipartite fits with positions: each sweep also draws, exactly, the factors by which the columns of the positions and the rows and columns of the interaction matrix are rescaled without changing their product, because the unit priors anchor the split of scale between them weakly and the conditional updates cross it slowly. `FALSE` omits it; its work is recorded in `fit$uv_g_scale_move`.

uv_split_move Logical, default `TRUE`. For directed fits with latent factors (`R > 0`): draw the split of the scale between U and V exactly ($U \to cU$, $V \to V/c$ leaves the product and the likelihood unchanged; with the covariance of the rows integrated out $\log c$ has the matrix-t density along the orbit and is drawn by a slice step). The conditional draws moved the split by a few percent per sweep against a posterior sd of 0.2 to 0.5. Recorded in `fit$uv_split_move`.

sparse_position_move Logical, default `TRUE`. For directed `"cbin"`, `"frn"` and `"ordinal"` fits with latent factors (`R > 0`): each sweep also draws the position of every sender and receiver together with the latent values of its row or column, exactly. Moving u_i changes the predictor of row i by a different amount in each column; carrying the latent values along changes no residual, so the conditional of the position is its prior times the indicator that the shifted row is still a latent row the data allow, drawn by an elliptical slice step. Actors the data bound on one side only (no nominations, no nominators, only the lowest category) move across their wide posterior in a few sweeps instead of by random walk. Not applied to `"cbin"` / `"frn"` panels with hidden cells. Recorded in `fit$sparse_position_move`. An ordinal fit that runs it visits the level translations of `actor_move` every sweep unless `actor_move_every` says otherwise.

sparse_position_every Sweeps between visits of `sparse_position_move` (default 1: every sweep).

cov_ridge_move Logical, default `TRUE`. For bipartite fits with a row or column covariate (a design slice that is constant along the columns or the rows): each sweep also draws, exactly, the amount by which the coefficient and the

additive effects of that margin trade against one another without changing any cell's predictor. The two conditional draws cross this ridge slowly (a row coefficient had 18 to 44 effective draws of 1,600 in a nomination panel). Directed unipartite fits of `ame()` and `lame()` (static or dynamic additive effects, static coefficients) take the same draw. `FALSE` omits it.

ordinal_scale_move Logical, default `TRUE`. For `"ordinal"` fits: each sweep also draws, exactly, the scale of the whole state (the coefficients, the additive effects and the latent array together, with the free cutpoints of `ordinal_cutpoints = "explicit"`) and, with the default data-induced cutpoints, the shift and the scale of the latent array alone, along transformations that keep every rank. The windows the ranks leave the latent values move the implied cutpoints by steps of `1 / number of cells`, while the scale of the configuration has a posterior of width `1 / sqrt(number of cells)` and the coefficients and the dyadic correlation follow it: on 150 actors a dyadic coefficient had 8 effective draws of 600 and the correlation 95, now 600 and 400. `FALSE` omits it; its work is recorded in `fit$ordinal_scale_move`. See `lame`.

binary_scale_move Logical, default `TRUE`. For `"binary"` fits: every `binary_scale_every` sweeps the coefficients and the additive effects are multiplied by a factor drawn exactly from its conditional density, with the latent array integrated out. The augmented sweep crosses that direction by steps of `1 / sqrt(number of cells)`, and when a covariate separates the ties the posterior along it is tens of times wider: on 30 actors, after 8,000 sweeps, the separating coefficient of a directed panel had 12 effective draws of 2,000, now 460, and that of a bipartite panel, which had drifted over those sweeps (14.0 and 15.6 for two seeds), now has a posterior mean of 59 at 2,000, 8,000 and 32,000 sweeps. `FALSE` omits it; its work is recorded in `fit$binary_scale_move`. See `lame`.

binary_scale_every Positive whole number, default 5: the sweeps between the draws of that factor.

Common usage: prior = list(Sab0 = diag(c(2, 2)), eta0 = 10) for moderate shrinkage, or prior = list(Sab0 = diag(c(0.5, 0.5))) for tighter control. For a unipartite cross-sectional `ame()` fit the prior is controlled by `Sab0`, `eta0`, `Suv0` and g.

g optional **scalar** regression prior scale, independent of the residual variance. The unipartite Gaussian block uses precision `(XtX + 1e-6 * diag(diag(XtX))) / g`; an all-ones first column additionally receives intercept variance `100 * g / XtX[1,1]`. If not specified, `g = n_observed_cells * var(Y)` for normal data and `g = n_observed_cells` otherwise, where observed cells exclude the diagonal (before covariate-missingness propagation). Per-coefficient (vector) g is not supported by the unipartite path. A finite positive g is required when regression coefficients are fitted; invalid scales are not replaced or clipped. *Note:* g is a top-level argument to `ame()`, **not** an element of `prior = list(...)`; passing `prior = list(g = 0.1)` is a no-op (warned about).

seed random seed for the MCMC sampler (default 6886). The sampler is seeded internally with this value, so results are reproducible by default and an external `set.seed()` call has *no* effect on the chain – pass a different seed here to vary the draws (e.g. when running multiple chains). The caller's `.Random.seed` is restored on exit, so fitting never perturbs your RNG stream.

nscan	number of iterations of the Markov chain (beyond burn-in)
burn	burn in for the Markov chain. Typical use is burn far smaller than nscan; burn > nscan is allowed but warns.
odens	output density (thinning interval) for the Markov chain. nscan / odens samples are stored.
verbose	logical: print progress while running? Default TRUE.
gof	logical: calculate goodness of fit statistics? Setting to TRUE adds approximately 2-5% to runtime. For faster sampling without GOF overhead, set gof=FALSE and use gof() after model fitting.
custom_gof	optional function or list of named functions for computing custom goodness-of-fit statistics. Each function must accept a single matrix Y as input and return a numeric vector. If a single function is provided, it should return a named vector. If a list of functions is provided, each function should return a single value and will be named according to the list names. Custom statistics will be computed in addition to default statistics. Example: custom_gof = function(Y) c(density = mean(Y > 0, na.rm = TRUE))
plot	accepted for signature parity with lame; ignored. ame() runs a single-period MCMC and does not draw live trace panels. Default FALSE.
start_vals	List from previous model run containing parameter starting values for new MCMC
periodic_save	logical: indicating whether to periodically save MCMC results
out_file	character vector indicating name and path in which file should be stored if periodic_save is selected. For example, on an Apple OS out_file="~/Desktop/ameFit.rda".
save_interval	quantile interval indicating when to save during the post-burn-in period.
posterior_opts	optional list of posterior draw-storage options, usually built with posterior_options. Recognised names (unknown names trigger a warning): save_UV, save_UV_draws, save_ab, thin_UV, thin_ab. save_UV = TRUE stores per-iteration latent-position draws on fit$U_samples / fit$V_samples ([actor, dim, draw] arrays; bipartite fits also store the interaction-matrix draws on fit$G_samples); save_UV_draws (the lame spelling) is accepted as an alias. save_ab = TRUE stores additive-effect draws on fit$a_samples / fit$b_samples (a symmetric fit has one effect per actor: the two hold the same draws). thin_UV / thin_ab thin the stored draws. Default NULL.
n_chains	integer: the older name of chains.
cores	integer or NULL: workers for the chains. The chains run forked on Unix, at most one worker per chain; NULL uses getOption("lame.cores") if set and otherwise all cores but one. Windows runs the chains one after the other.
chains	integer: the number of independently seeded chains (default 2, or getOption("lame.chains")). Chain k runs with seed seed + 1000 * (k - 1). With chains > 1 the chains are checked, compared and pooled (see the convergence policy); chains = 1 runs one chain and the drift check only.
use_sparse_matrices	logical: use sparse matrix storage for large networks? (default: FALSE). Recommended only for truly sparse networks (< 10% non-zero entries).

`method` character: `"mcmc"` (default, the Bayesian MCMC fit) or `"als"` (the fast, MCMC-free iterative block coordinate descent point estimator). When `method = "als"`, MCMC-specific arguments (`nscan`, `burn`, `odens`, ...) are warned about and ignored; the call forwards to `ame_als`, which uses `prior$s20`.

`bootstrap` integer (only used when `method = "als"`): number of bootstrap replicates. `0` (default) skips the bootstrap; `N >= 30` runs `N` replicates and attaches the result so that `confint` returns bootstrap intervals (a smaller positive `N` is an error).

`bootstrap_type` character (only used when `method = "als"`): `"parametric"` (default) or `"block"`.

`bootstrap_block_length` integer: block length for the block bootstrap.

`bootstrap_seed` optional integer seed for the bootstrap.

`multistart` character (only used when `method = "als"` and `R > 0`): `"none"` (default), `"cheap"` (4 starts), or `"full"` (8 starts). Forwarded to `ame_als`, which reruns the block-coordinate fit from several starts and keeps the one with the lowest objective.

`save_log_lik` logical: attach a per-iteration matrix of matched observed marginal scores. LOO and WAIC additionally require a verified observed likelihood factorization; directed residual dyad correlation prevents cellwise factorization. Default `FALSE`.

`ordinal_cutpoints` character: cutpoint convention for `family = "ordinal"`. `"data_induced"` (default) uses the data-induced cutpoints; `"explicit"` samples ordered cutpoints with the first fixed at zero and unit latent residual variance. Independent directed and symmetric models use the Cowles (1996) marginal Metropolis-Hastings update. Directed models with `dcor = TRUE` use cutpoint Gibbs updates conditional on the correlated latent observations. Explicit cutpoints are supported for unipartite models; predictions retain the original numeric category labels and use each saved cutpoint draw. Ignored for other families.

`print` Deprecated. Use `verbose` instead.

`...` reserved for future use. Passing `lame()`-only arguments (e.g. `dynamic_beta`, `period_exposure`) here triggers a clean abort directing you to `lame`; passing any other unrecognised name warns so typos are visible rather than silently dropped.

`log_lik_method` `"observed_exact"` (default) supports normal, binary, Poisson-lognormal and explicit-cutpoint ordinal marginal scores. Poisson integration uses an adaptive log-integral with concavity tail bounds. Capped nominations, fixed ranks and data-induced ordinal models are refused. `"observed_approx"` retains the existing approximate diagnostic scores for these families without claiming an observed likelihood. AME does not save latent Z draws for augmented scoring. Inspect `fit$log_lik_contract`; approximate diagnostics cannot be used for observed-data LOO or WAIC.

`model.name` optional string for model selection output

`bias_correct` character or `NULL` (only used when `method = "als"`): how the ALS point estimate handles the finite-sample bias of a fixed-effects fit, forwarded to `ame_als`. `NULL` (default) fits a normal, binary or count network by the empirical-Bayes estimator (`"eb"`; Laplace standard errors from `vcov()`), directed, bipartite or undirected, with or without a latent rank (see `bias_correct` in `ame_als`). `"none"` returns the fixed-effects estimate at any size; `"jackknife"` and `"bootstrap"` require that correction of it.

Details

This command provides posterior inference for parameters in AME models of cross-sectional relational data, assuming one of six possible data types/models. The function supports both unipartite networks (square adjacency matrices) and bipartite networks (rectangular adjacency matrices with distinct row and column node sets) for single time point analysis.

Model. For a dyad (i, j) the AME linear predictor is

$$\eta_{ij} = \beta_0 + x'_{ij}\beta + a_i + b_j + u'_i D v_j,$$

where β are regression coefficients on dyadic / nodal covariates, a_i is a row (sender) random effect, b_j is a column (receiver) random effect, and $u'_i D v_j$ is the multiplicative latent-factor term (rank `R`). The observation model is $Y_{ij} \sim F(\eta_{ij}, \theta)$ with F specified by `family` (Gaussian for `"normal"`, probit for `"binary"` / `"cbin"`, etc.). For unipartite (`symmetric = FALSE`) the residual error has dyad-level correlation ρ between (i, j) and (j, i); for bipartite, dyad correlation is fixed at 0.

Priors (in brief). β has a Zellner-style g-prior (`g`); (a_i, b_i) are jointly Normal with covariance Σ_{ab} (Inverse-Wishart prior `Sab0 / eta0`); u_i, v_j are independent Normal with covariance Σ_{uv} (Inverse-Wishart prior with scale `kappa0 * Suv0` and `kappa0` degrees of freedom); the dyad-correlation ρ has the prior $2\,\mathrm{Beta}(2, 2) - 1$ on $(-1, 1)$ (density proportional to $1 - \rho^2$, the prior `lame` uses), updated with Metropolis steps. See `prior_summary(fit)` for the priors actually used.

Choosing R. The multiplicative rank `R` controls the dimensionality of latent homophily / heterogeneity not explained by covariates and additive effects. `R = 0` fits an additive-only social-relations model; `R = 1` or `2` is typical for small / medium networks. A rank above `floor(n/3)` is fitted like any other (on 18-actor panels with a rank-2 truth, ranks 6 to 17 ran without numerical failure; the chain mixed more slowly and the fitted values lost accuracy gradually as the rank grew past the truth), so compare ranks by their fit rather than by a cap. Latent factors capture unobserved structure (clusters, hub patterns, transitive triangles the covariates miss) and are accessed at `fit$U`, `fit$V`.

Identifiability. The latent factor term $u'_i D v_j$ is invariant to rotation and reflection of U, V; the package canonicalises with an SVD so successive draws are interpretable. For unipartite fits with both resolved actor margins enabled, reporting centers static latent positions and transfers the removed row, column and grand-mean components to the actor effects and intercept. A symmetric shared actor absorbs half the grand constant in each margin when no intercept is fitted. Restricted unipartite models retain sampled coordinates in their stored factor draws. This reporting map preserves each draw's predictor and leaves the sampler state unchanged. For visual stability across posterior summaries see `procrustes_align` and `latent_positions`.

Theoretical Foundation:

The AME model decomposes network structure into several components:

$$y_{ij} = \beta' x_{ij} + a_i + b_j + u'_i v_j + \epsilon_{ij}$$

where:

- $\beta' x_{ij}$: Fixed effects of dyadic/nodal covariates
- a_i: Additive sender (row) effect for node i
- b_j: Additive receiver (column) effect for node j
- $u'_i v_j$: Multiplicative interaction between latent factors
- ϵ_{ij}: Dyadic error term (may be correlated)

This specification generalizes the social relations model (Warner et al. 1979) and latent space models (Hoff et al. 2002) within a unified framework.

Prior Distributions:

The model uses conjugate and semi-conjugate priors where possible:

- Regression coefficients: $\beta \sim N(0, g\sigma^2(X'X)^{-1})$ (g-prior)
- Additive effects: $(a_i, b_i)' \sim N(0, \Sigma_{ab})$ jointly
- Covariance: $\Sigma_{ab} \sim IW(\eta_0, \eta_0 S_{ab0})$ (inverse-Wishart)
- Multiplicative effects: Hierarchical shrinkage via η_0
- Dyadic correlation: $2\,\mathrm{Beta}(2,2) - 1$ prior on $(-1, 1)$, density $p(\rho) \propto 1 - \rho^2$ (the prior of `lame`), with Metropolis updates

The inverse-Wishart prior on Σ_{ab} allows learning correlation between sender and receiver effects, capturing reciprocity patterns.

Note on the additive-effects variance scale: for a unipartite fit `Sab0` defaults to a data-scaled matrix rather than a fixed `diag(2)`. For the `"normal"` and `"poisson"` families the default is `Sab0 = diag(2) * vscale`, where `vscale` is an empirical-Bayes moment estimate of the sender/receiver variance: the mean of the variances of the centred row means and column means of `Y` (of `log1p(Y)` for `"poisson"`). The `"binary"`, `"ordinal"`, `"cbin"` and `"frn"` families apply the same idea to a probit-moment residual, but only when `start_vals` is not supplied and the design has at least one column; otherwise they fall back to `Sab0 = diag(2)` with `eta0 = 4`. A bipartite fit scales `Sab0` the same way (one scale for both sides). Call `prior_summary()` on a fitted object to see the prior actually used.

The data scaling matters because the inverse-Wishart prior contributes pseudo-data on the scale of `eta0 * Sab0` no matter what units `Y` is in, so a fixed `Sab0 = diag(2)` pulls the additive-effects variances `va`/`vb` upward whenever the true sender/receiver variance is well below 1, most visibly at small n. The data-scaled default sets the prior on the scale of `Y`; `prior_summary()` prints it.

To use a fixed identity scale instead, pass `prior = list(Sab0 = diag(2), eta0 = 4)`. Changing this prior can affect other jointly estimated quantities as well as `va`, `vb` and `cab`. Matching these prior settings alone does not establish equivalence to another package's posterior.

Multiplicative Effects (Latent Factors):

When R > 0, the model includes R-dimensional latent factors:

- Asymmetric case: $u_i, v_j \in \mathbb{R}^R$ with $u_i' v_j$ interaction
- Symmetric case: $u_i = v_i$ with eigendecomposition ULU'
- Captures homophily, transitivity, and community structure
- R chosen via model selection or set to 2-3 for visualization

Estimation Algorithm:

The model uses a Gibbs sampler with the following updates:

1. Sample latent Z given parameters (data augmentation for non-normal families)
2. Update regression coefficients β via g-prior conjugate update
3. Update additive effects (a,b) jointly with β
4. Update covariance Σ_{ab} from inverse-Wishart

5. Update multiplicative effects U,V via Gibbs or Metropolis-Hastings
6. Update dyadic correlation ρ via Metropolis-Hastings
7. Update variance σ^2 (for continuous families)

Standard Model Types:

The following data types/models are available:

"normal": A normal AME model (identity link: $E[Y] = \eta$).

"binary": A binary probit AME model (probit link: $P(Y = 1) = \Phi(\eta)$).

"ordinal": An ordinal probit AME model (cumulative probit link). An intercept is not identifiable with data-induced cutpoints. Explicit cutpoints anchored at zero support an intercept.

"cbin": An AME model for censored binary data (probit link with censoring). The value of 'odmax' specifies the maximum number of links each row may have.

"frn": An AME model for fixed rank nomination networks. A higher value of the rank indicates a stronger relationship. The value of 'odmax' specifies the maximum number of links each row may have.

"poisson": An overdispersed Poisson-lognormal model for counts (negative or non-integer observed values are an error): $Y \sim \text{Poisson}(e \exp(z))$ with $z \sim N(\eta, \sigma^2)$. Given η and σ^2, a cell's mean count is $e \exp(\eta + \sigma^2/2)$. `YPM`, `fitted()` and `predict(type = "response")` report it at the posterior mean η of the linear predictor and the posterior mean σ^2 of the residual variance, the form the fast estimators report. Predictive quantiles and event probabilities come from posterior predictive draws (`simulate()`, `predict(type = "distribution")`). Zero exposure gives a structural zero count (cross-sectional fits use unit exposure).

Value

Posterior Samples (full MCMC chains):

`BETA`	Regression coefficients (`nscan/odens` x p matrix; one row per stored draw)
`VC`	Variance components (`nscan/odens` x k matrix)
`GOF`	Goodness-of-fit statistics ((`nscan/odens` + 1) x 5 matrix; three columns, `sd.rowmean`, `sd.colmean` and `four.cycles`, for `mode = "bipartite"`). First row contains observed values, remaining rows contain posterior predictive samples. See `gof` for post-hoc computation and `gof_plot` for visualization.

Posterior Means (averaged over chain):

`APM`	Additive row/sender effects (n-vector)
`APM_sd`, `BPM_sd`	posterior standard deviation of each additive effect (static effects), used by `simulate()` to draw the effects
`BPM`	Additive column/receiver effects (m-vector); NULL for symmetric networks
`U`	Multiplicative row/sender factors (n xR matrix)
`V`	Multiplicative column/receiver factors (m xR matrix); NULL for symmetric networks
`L`	Eigenvalue matrix (R xR diagonal); symmetric networks only

`YPM` Stored response-scale fitted value: the average of the outcomes replicated at each stored draw, except for Poisson fits, where it is the mean count $\exp(\eta+\sigma^2/2)$ at the posterior means of the linear predictor and of the residual variance; see the Poisson model details.

Metadata:

`family` Model family (normal, binary, etc.)

`mode` Network mode (unipartite or bipartite)

`sampler_health` For MCMC fits, counts of numerical failures, separated from valid prior-bound rejections. A numerical failure (a caught collapsed reciprocity or utility-refresh error in unipartite fits, a failed U, V or G draw in bipartite ones) aborts the fit with an error of class `lame_sampler_failure` that carries the counts in its `health` field, so a returned fit has none. A proposal outside the prior's box (`prior$uv_max_abs`, `prior$G_max`) is a Metropolis rejection of the model: that conditional keeps its state, and the rejections are counted here. U and V proposal counts in bipartite fits are per proposed actor row and G counts per Gaussian interaction-matrix draw.

`sampler_health_iterations`
Number of burn-in plus production sweeps included in the MCMC health counts. U and V proposal counts in bipartite fits can exceed this number because each sweep visits multiple actor rows.

`uv_g_scale_move`
For bipartite fits with positions, the number of sweeps and of rescaled coordinates of the draw along the scale ridge of the positions and the interaction matrix (see `uv_g_scale_move` in `prior`); absent when the update did not run

`ordinal_scale_move`
For `"ordinal"` fits, the sweeps the scale moves visited (`visits`) and the sweeps at which the scale of the whole state moved (`orbit_moved`); absent when the moves did not run

`poisson_period_move`
For `"poisson"` fits, the sweeps the scale move visited (`visits`) and its proposals and accepts (`scale_attempts`, `scale_accepts`, and for the correlation alone `scale_rho_attempts`, `scale_rho_accepts`); absent when the move did not run

`binary_scale_move`
For `"binary"` fits, the draws of the scale of the linear predictor (`visits`), the evaluations of its density they used (`evals`) and the draws that moved the state (`moved`); absent when the draw did not run

`actor_move` For Poisson and ordinal fits, the work of the translations of the levels with the latent array: `visits`, the level translations drawn (`level_calls`), the likelihood evaluations of the Poisson ones (`level_evals`), `stuck` and `skipped` steps, the draws of the location of the latent array (`location_calls`, ordinal), the coefficient translations (`coef_calls`, `coef_evals`, `coef_stuck`; Poisson), the position steps (`uv_calls`, `uv_evals`, `uv_stuck`; Poisson with positions) and the attempts and accepts of the scale move of the dyadic variance and correlation (`scale_attempts`, `scale_accepts`, `rho_attempts`, `rho_accepts`; Poisson); for binary fits, the work of the collapsed per-actor update: `visits`, the

slice steps of the level translations (level_calls) and of the static positions (uv_calls), the likelihood evaluations they used (level_evals, uv_evals), the steps that failed to terminate (stuck) and, for bipartite fits, the draws along the ridge of the intercept and the additive-effect means (ridge_calls); for "cbin" and "frn" fits, the work of the receiver-level translation: visits, the receivers moved (col_calls) and the receivers left to the sampler because the current state lay outside the interval the translation needs (col_skips), the sweeps in which the level update moved a set of rows (level_calls) and the rows it moved (level_rows), the proposals and accepts of the reciprocity update (rho_attempts, rho_accepts) and, for a bipartite fit, ridge_calls; for the other families of a bipartite fit with an intercept, visits and the draws along the ridge of the intercept and the level means (ridge_calls); absent when the updates did not run (see actor_move in prior)

symmetric — Logical indicating if network is symmetric

R — Dimension of multiplicative effects

prior_supplied — TRUE when prior carried Sab0, eta0 or etaab: where the data alone leave the effect variances to the prior (a saturated nomination cap, see odmax), the reporting methods then show them under the supplied prior instead of as not estimated

Optional Posterior Samples (if requested via posterior_options):

U_samples — Samples of U (n x R x iterations array)

V_samples — Samples of V (m x R x iterations array)

a_samples — Samples of row effects (n x iterations matrix)

b_samples — Samples of column effects (m x iterations matrix); in a symmetric fit the same draws as a_samples

Note on the latent-scale matrices: The posterior-mean multiplicative product is stored on the fit (UVPM, or ULUPM for symmetric fits); EZ (the expected latent network) is not stored, to save memory. Accessors:

- reconstruct_EZ(fit) - Returns linear predictor (link scale, not response scale)
- reconstruct_UVPM(fit) - Returns the stored posterior-mean multiplicative product (UVPM / ULUPM) when present, otherwise U\

Generating posterior distributions: Use simulate_posterior(fit, component="UV") to generate posterior samples for components where only means are stored, or use posterior_options() during model fitting to save full posterior samples.

model.name — Name of the model (if provided)

Supported family x mode combinations

Every family is supported under both modes. The bipartite Z-samplers live in R/rZ_bipartite.R and dispatch per family; see also the inline comment in R/lame.R (the rectangular samplers live in R/rZ_bipartite.R).

family	unipartite	bipartite
normal	yes	yes
binary	yes	yes
ordinal	yes	yes
cbin	yes	yes
frn	yes	yes
poisson	yes	yes

Symmetric (`symmetric = TRUE`) fits require a symmetric `Y` and symmetric `Xdyad` slices: the regression term of a symmetric fit is symmetric, so an asymmetric slice is an error (class `lame_symmetric_xdyad`) that names the slice. Replace it by `(X + t(X)) / 2`. A value missing in one triangle of a slice is not an asymmetry: it is filled from the other triangle (see *Missing covariates*). `lame` applies the same checks to every period, and the ALS routes (`method = "als"`, `ame_als`) refuse an asymmetric slice the same way. `family = "ordinal"` with `symmetric = TRUE` is supported via the unit-variance unordered-dyad samplers for either cutpoint convention, which mirror upper-triangle draws to the lower triangle so $Z = t(Z)$ holds at every sweep. The rank-nomination families (`"cbin"`, `"frn"`) model each actor's own ranked nominations and are directed by construction, so `symmetric = TRUE` is rejected for them.

Symmetric input with one triangle missing. When `symmetric = TRUE` and one triangle of `Y` is fully `NA` (the user stored only the lower or upper triangle), the symmetry check `any(is.finite(Y - t(Y)))` is `FALSE`, and the call stops with an error instead of mirroring the populated triangle. Symmetrize `Y` explicitly (for example `Y[lower.tri(Y)] <- t(Y)[lower.tri(Y)]`) before calling; `lame()` applies the same check to every period.

Data preparation

`ame()` accepts a matrix directly. For long-format edgelists, bipartite data, and covariates, use **netify** to build the network object and pass that object as `Y`. `ame()` will call `netify::to_lame()` internally. If you already have an **igraph** or **network** object and only need a plain adjacency matrix, `as_lame_y` is still available as a small convenience helper.

Directed normal MCMC models with estimated reciprocity use the prior $2\,\mathrm{Beta}(2,2)-1$ on the full open interval $(-1,1)$, with density proportional to $1-\rho^2$. If every observed reciprocal response pair is exactly equal, the posterior can be improper at $\rho = 1$. A conservative structural check counts reciprocal pairs and bounds the antisymmetric actor/regression design dimension. With multiplicative rank `R`, it also includes a conservative `n * R` factor bound. The check refuses cases where the pair count exceeds that combined bound by four or more (the posterior is improper from that excess under this prior; under the arc-sine prior of versions before 1.4.0 it was improper from an excess of one); it does not certify propriety of other designs. For an undirected network, use `symmetric = TRUE`; to fix reciprocity at zero in a directed model, use `dcor = FALSE`. An exactly symmetric `Y` fitted with `symmetric = FALSE` is allowed: the two cells of a dyad are equal, so the residuals of reciprocal cells are equal and the posterior of the dyadic correlation concentrates near one (with `dcor = FALSE` the two cells count as two observations). A message says so and points at `symmetric = TRUE`, which fits the undirected model.

For an undirected/symmetric network, pass `symmetric = TRUE`; for a rectangular two-mode network (students x courses, donors x candidates), pass `mode = "bipartite"` or build the **netify** object with `mode = "bipartite"`. `Xrow` and `Xcol` accept either a numeric matrix or a data.frame (coerced internally); `Xdyad` must be a 3-D array `n x n x p` of numeric covariates with no `Inf`/`NaN`.

Missing covariates

A cell whose dyadic covariate, sender (row) covariate or receiver (column) covariate is missing is an unobserved tie, for unipartite and bipartite networks alike: it is set to `NA` in `Y` before the design is built, the sampler augments it like any other missing cell, and the covariate coefficients are estimated from the complete cells. A message gives the number of such cells. A missing value is never modelled as 0, which would bias the coefficient of a covariate that is not centred. In an undirected network (`symmetric = TRUE`) a dyad has one covariate value: a hole in one triangle of a dyadic covariate is filled from the other triangle (a message gives the number of values), so the slice stays symmetric and the dyad stays observed, and a dyad missing in both triangles is unobserved. A missing node covariate removes the actor's row and column. A covariate that is missing everywhere stops the fit. `lame` (per period) and the ALS routes (`ame_als`) apply the same rule and give the same messages.

Collinear covariates

A regression design whose columns are exactly aliased over the observed cells (a covariate that duplicates another, or one that is a linear combination of others, counting the intercept and the sender and receiver covariates) is an error of class `lame_design_rank`. It names the aliased columns and the combination each equals: the data cannot separate their coefficients, so the posterior along that direction would be the prior's alone. A column is aliased when it is a combination of the columns before it to a relative 1e-8 (as `lm()` reports them); a design that is only nearly collinear is fitted.

When to use AME vs ERGM

AME and ERGM are complementary tools for binary network analysis, not direct substitutes. **ERGM** is a class of exponential-family models built around explicit network statistics (counts of edges, mutual ties, triangles, geometrically-weighted shared partners, ...). You write the statistics you think matter, ERGM gives you their coefficients. ERGM excels when you have a substantive theory about which configurations drive tie formation.

AME models latent homophily / heterogeneity directly via sender, receiver, and multiplicative latent-factor effects. You don't enumerate triadic terms; the multiplicative-effects rank `R` captures higher-order structure (clustering, transitivity, hub patterns) implicitly. AME excels when (a) you have dyadic / nodal covariates whose effects you want to interpret cleanly without ERGM degeneracy, (b) higher-order structure is "nuisance" that you want to absorb but not parameterise, or (c) you need a posterior distribution over predictions for forecasting or imputation.

Practical guidance: if your research question is "do nodes that share attribute X tend to form triangles together?", reach for ERGM's `gwesp`. If your research question is "controlling for unobserved sender / receiver heterogeneity and latent clustering, what is the effect of dyadic covariate X?", reach for AME. The `R = 0` additive-only case is the social relations model (Warner, Kenny, Stoto 1979); `R >= 1` adds latent space.

ERGM to AME translation

For users coming from `statnet::ergm`, the rough analogues are:

ERGM term	AME analogue
`edges`	`intercept` (probit link, not logit)
`nodecov("x")`	`Xrow = x` or `Xcol = x`

nodematch("g")	dyadic covariate via nodematch(g) into Xdyad
nodefactor("g")	dyadic covariate via nodefactor(g) into Xdyad (drop one level)
absdiff("z")	dyadic covariate via absdiff(z) into Xdyad
mutual	dcor = TRUE -> the rho parameter (probit-scale, not log-odds; not numerically comparable to E
gwesp / transitivity	R >= 1 multiplicative latent factors (not the same statistic)
sender activity heterogeneity	rvar = TRUE, gives a_i, va
receiver popularity heterog.	cvar = TRUE, gives b_j, vb

For family = "binary" the link is probit, so the intercept is on the probit scale; do not compare it to an ERGM edges estimate by simple arithmetic.

Migration from amen

Both amen and lame export ame(); loading both packages fires a startup warning telling you to call lame::ame(...) or amen::ame(...) explicitly.

For default cross-sectional calls, lame::ame() follows the amen::ame() interface: the fit$BETA slot is a 2-D [n_stored, p] matrix in both packages, so scripts that call colMeans(fit$BETA) or apply(fit$BETA, 2, mean) continue to work unchanged. lame additionally accepts family = "binary" (which amen 1.4.5 no longer accepts; amen requires "bin"). The print argument is deprecated in favour of verbose; calls that pass print = ... still work but warn.

The cross-sectional path has no dynamic_beta option (an AR(1) prior on a single-period coefficient is unidentified), so the BETA 3-D shape that lame() can produce never arises from ame(). See lame for the longitudinal path and the silent-aggregation hazard with 2-D apply(fit$BETA, 2, mean) scripts under dynamic_beta = TRUE.

Notes on priors

- The regression-coefficient prior is a Zellner g-prior (beta ~ N(0, g * sigma^2 * (XtX)^-1) where XtX is the design cross-product). g is a *top-level* argument of ame(), not an entry in prior = list(...) (a common slip).
- There is no per-coefficient prior knob. If you need student_t, horseshoe, or to centre a slope away from zero, AME does not currently expose it; the g-prior structure is the only knob.
- Unknown names in prior = list(...) are warned about (a typo like Sab = ... instead of Sab0 would otherwise be silently dropped).

Chains and convergence policy

ame() runs chains chains (default 2; options(lame.chains = k) sets another default). Chain k is seeded with seed + 1000 * (k - 1), so chains = 1 is the first chain of a two-chain fit. On Unix the chains run forked, on at most cores workers (all cores but one by default); on Windows they run one after the other. The draws do not depend on the number of workers. The result is the pooled fit of combine_ame_chains, with fit$chains, fit$chain_seeds, fit$rhat (the R-hat of the monitored set), the state each chain stopped in, and the chain_indicator that as_draws() and trace_plot() use. Two chains cost twice the CPU time and about the same wall time on a multicore machine.

At the end of the post-burn run every chain is checked, and the chains are compared with each other. Within a chain, the first and last thirds of the stored draws of the intercept (or first coefficient), va,

vb, ve and rho are compared with batch-means standard errors (ten batches per third); a quantity is flagged when the two means differ by more than 3 standard errors and by more than half a posterior standard deviation of the whole series. Between chains, the split rank-normalised R-hat (Vehtari et al., 2021) of the monitored set is compared with 1.05: every regression coefficient, va, vb, cab, ve, rho, and summaries of the additive and latent structure recorded at every stored draw (the rms of a and b, the largest actor effect, the rms of the latent product UV', the norm of G of a bipartite fit and the split of scale between U and V of a directed fit). The dyadic products are not monitored. Quantities a fit reports as not estimated are left out, and a run with fewer than 60 stored draws per chain is not checked. The gate is computed on the draws the returned fit holds (the last block of every chain), and `summary()` and `compute_mcmc_diagnostics` report the same statistic of the same draws: the Rhat column of `summary()` is `fit$rhat` for every monitored coefficient and variance component.

When a chain drifts or an R-hat exceeds 1.05, every chain is extended by a block of nscan sweeps from its own final state (the seeds advance by the sweeps already run); the flagged block counts as extra burn-in, its draws are discarded, and both checks run again. Up to `prior$chain_extensions` extensions are made (default 3, so at most four times nscan post-burn sweeps per chain). The returned fit has the same number of stored draws as a fit that needed no extension, and `fit$auto_extend` records the extensions, the sweeps discarded and the quantities that triggered each. If the last block is still flagged, `ame()` stops with an error of class `lame_chain_unsettled`: its `fits` field holds the chains, `fit` the pooled fit of the last block, `rhat` the monitored table and `moving` the offending quantities, and `lame_resume(e$fit, nscan = nscan)` continues every chain (`lame_resume(e$fits[[k]], nscan = nscan)` one). `prior = list(chain_extensions = 0)` refuses at the first flag; `chains = 1` runs one chain with the drift check only. There is no mode that returns a flagged fit. The measured false-extension rate and power are in `lame`.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

`lame` for longitudinal models, `gof` for post-hoc goodness-of-fit computation, `gof_plot` for visualizing GOF results, `latent_positions` for extracting latent positions as a tidy data frame, `procrustes_align` for Procrustes alignment of latent positions, `summary.ame` for model summaries, `coef.ame` for coefficient extraction

Examples

```
data(YX_nrm)
# the default runs two chains and compares them on the regression
# coefficients and variance components
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, burn = 100, nscan = 600, odens = 10,
           verbose = FALSE)
summary(fit)
fit$rhat  # between-chain split R-hat; every value is below 1.05
# the defaults (burn = 500, nscan = 10000, odens = 25) are the length of a final run
```

`ame_als` *Fast (MCMC-free) AME estimation for a cross-sectional network*

Description

Fits an additive and multiplicative effects (AME) model to a single cross-sectional network by **iterative block coordinate descent**, producing a fast point estimate with no MCMC and no credible intervals. For normal, binary and Poisson data the default is an empirical-Bayes / variational estimator of the model that `ame` samples (see `bias_correct`); its `vcov()` and `confint()` are Laplace approximations.

The estimation algorithm adapts the *iterative block coordinate descent* estimator of the Social Influence Regression (SIR) model of Hoff & Minhas (the algorithm implemented in `sir::sir_alsfit()` and nicknamed "ALS" in that package) to the AME model. It is a port and adaptation, not original **lame** methodology.

Usage

```
ame_als(
  Y,
  Xdyad = NULL,
  Xrow = NULL,
  Xcol = NULL,
  R = 0,
  family = "normal",
  mode = c("unipartite", "bipartite"),
  symmetric = FALSE,
  max_iter = 200,
  tol = 1e-06,
  lowrank_method = c("mm", "als", "hybrid"),
  non_normal_method = c("irls", "transform"),
  link = c("probit", "logit"),
  linear_solver = c("eigen", "qr", "auto"),
  multistart = c("none", "cheap", "full"),
  bias_correct = NULL,
  bootstrap = 0L,
  bootstrap_type = c("parametric", "block"),
  bootstrap_block_length = 1L,
  bootstrap_seed = NULL,
  verbose = TRUE,
  seed = 6886,
  dcor = !symmetric,
  prior = NULL
)
```

Arguments

Y	a square (unipartite) or rectangular (bipartite) relational matrix. A cross-sectional **netify** object is also accepted and converted with `netify::to_lame()`; when `family` or `mode` is omitted, the value inferred by **netify** is used.
Xdyad	an `n_row` x `n_col` matrix or `n_row` x `n_col` x `pd` array of dyadic covariates, or `NULL`. A 2-D matrix is wrapped as a single slice named after the argument expression, so `Xdyad = distance` yields the coefficient `distance_dyad`, the same name `ame` gives it.
Xrow	an `n_row` x `pr` matrix of row/sender covariates, or `NULL`.
Xcol	an `n_col` x `pc` matrix of column/receiver covariates, or `NULL`.
R	integer dimension of the multiplicative effects (default `0`). **The covariate coefficients are conditional on this choice.** The multiplicative term $u_i' v_j$ is a flexible high-variance regressor that can correlate with the dyadic covariates, so the estimated `beta` can shift – and occasionally change sign – as `R` increases. Comparing against an `R = 0` fit is a useful check on whether the covariate story is being driven by the latent rank.
family	one of `"normal"`, `"binary"`, or `"poisson"`. The rank and censoring families are MCMC-only.
mode	`"unipartite"` (square) or `"bipartite"` (rectangular).
symmetric	logical; fit a symmetric (undirected) model. Unipartite only. Every dyadic covariate slice must be symmetric: an asymmetric one is an error of class `lame_symmetric_xdyad` that names the slice (and, for a panel, its first period), as in `ame` and `lame`.
max_iter	maximum number of block coordinate descent iterations (default 200), and maximum number of outer reweighting steps for IRLS. A fixed-effects fit that has not converged within it is refitted once with four times the limit, and is an error (class `"lame_als_convergence"`) if that run has not converged either, as is an empirical-Bayes fit that does not settle (without the second run); the limit also bounds each round of the joint refinement of a Gaussian fit. An empirical-Bayes reweighting step runs at most five sweeps (it refreshes the weights, the Laplace correction and the variance updates after them). The variational stage of an empirical-Bayes fit runs at most `2 * max_iter` iterations. An empirical-Bayes fit whose iteration does not settle retires its weakest latent column and fits the next rank down (see **A rank the data do not support**); a `max_iter` below 200 disables that for an iteration that merely ran out of it.
tol	convergence tolerance on the relative change in residual sum of squares (default `1e-6`). IRLS additionally requires the largest penalized likelihood score, scaled by the square root of its Fisher information plus penalty curvature, and the largest absolute log variance update to be at most `sqrt(tol)`.
lowrank_method	inner solver for the multiplicative (low-rank) block: `"mm"` (default) weighted majorise-minimise; `"als"` alternating least squares; `"hybrid"` runs both and keeps the lower-objective result. `"als"`/`"hybrid"` converge faster than `"mm"` on strongly unbalanced longitudinal panels and are available for directed and bipartite models (symmetric fits always use `"mm"`). All three minimise the same objective, so the point estimate is unchanged for balanced data.

`non_normal_method`
: for the non-normal ALS families (it does not apply to a normal fit), `"irls"` (default for `binary`/`poisson`) runs iteratively reweighted least squares, giving a fast GLM AME fit with coefficients on the requested link scale (Poisson log, binary logit/probit). `"transform"` fits one fixed Gaussian working response (`log(y+1)` for Poisson, rank-normal scores for binary); its coefficients are on an uncalibrated working scale and are mainly useful for direction/ranking checks. A directed `R > 0` IRLS fit uses the hybrid low-rank solver internally because the IRLS weights are unbalanced. Uncertainty for either path comes from the bootstrap or sandwich covariance.

`link`
: link for `non_normal_method = "irls"` with a `binary` family: `"probit"` (default; matches `ame`/`lame`) or `"logit"`. `poisson` always uses the log link; ignored otherwise.

`linear_solver`
: solver for the regression block: `"eigen"` (default) eigendecomposes the normal equations; `"qr"` uses a QR factorisation of the observed design, which is more stable for ill-conditioned covariates; `"auto"` picks `"qr"` when the design is ill-conditioned but full rank. All give the same answer for well-conditioned designs.

`multistart`
: for `R > 0` (a non-convex objective), `"none"` (default) fits from a single deterministic start; `"cheap"` (4 starts) and `"full"` (8 starts) also try random low-rank starts and keep the converged start with the lowest value of the criterion, evaluated at that start's own scales: for an IRLS fit the penalised deviance, which carries the factor penalty with its scale parameters (`"penalised deviance"`), and for an empirical-Bayes fit with a latent rank the Laplace marginal likelihood of the start, minus twice its logarithm (`"Laplace marginal likelihood"`; see **Choosing the rank**), which integrates the effects and the factors and does not move with the scale of a factor column the data do not use (with the default variational stage every start is refined, and starts of different modes end at the same variational fit, so they neither differ in `multistart_distance` nor warn). An empirical-Bayes start without it is rated by the penalised deviance plus the penalty of the actor effects, the log determinant of their information and the hyperprior of their covariance (`"empirical-Bayes objective"`). The raw deviance is not compared: it always favours the least-shrunk optimum, the one with the largest scales. Other fits keep the lowest working SSE. The criterion (`multistart_criterion`), its value for each start (`multistart_sse`; for an empirical-Bayes fit it includes the hyperprior, which `objective` leaves out, or is the marginal likelihood), each start's observed-data deviance (`multistart_deviance`, IRLS fits), its distance from the kept start (`multistart_distance`, IRLS fits) and its convergence are recorded. The distance is the squared difference of the two fitted linear predictors, weighted by the mean of the two IRLS weights and summed over the observed cells: an approximation of the change in the deviance between the two fits. An IRLS fit warns of materially different local optima when a converged start is more than one deviance unit away from the kept one. Starts that end at the same fitted values can still differ in the penalised objective, which depends on the scales of the factors, and those settle more slowly than the fitted values (the Laplace marginal likelihood differs far less); that difference does not warn. Every multistart fit also keeps the coefficients each start reports (`multistart_coefficients`) and the distance of each start's multiplicative sur-

face from the kept start's (multistart_product_distance), so that the spread of the estimates across the starts can be read from the fit. Reproducible given seed; the global RNG stream is left unchanged.

bias_correct how an ALS fit deals with the finite-sample (incidental-parameter) bias of a fixed-effects fit. NULL (default) fits normal models, and binary and poisson models on the IRLS path, by the empirical-Bayes estimator ("eb"), for directed, bipartite and undirected networks, with or without a latent rank: a Gaussian prior on the additive effects with its covariance estimated from the data, coefficients that solve the Laplace-marginal equations, and Laplace standard errors from vcov() for every coefficient (see **Empirical-Bayes estimator**, **Normal data** and **Counts** in Details). A normal response is fitted as the model that ame and lame sample, with an exact Gaussian cell, and a count network as the Poisson-lognormal model that ame and lame sample. Other families are not corrected. "eb" asks for the empirical-Bayes estimator explicitly; it is an error (class "lame_als_eb") for another family and for a binary or count fit with non_normal_method = "transform". (The dynamic routes of lame(method = "als") have their own empirical-Bayes estimator for normal, binary and Poisson panels; see bias_correct in lame.) "none" returns the fixed-effects fit at any size and records the request (for a normal response, the least-squares fit of the earlier releases with the conditional sandwich covariance of the dyadic coefficients; for counts, the fit of the earlier releases, in which the effect of an actor without a count on a side runs to minus infinity); "jackknife" requires the split-sample correction of that fit, for any family, and is an error where it cannot be computed; "bootstrap" requires the parametric-bootstrap correction of a binary IRLS fit at any size. See **Split-sample bias correction**, **Parametric-bootstrap bias correction** and **Size boundary** in Details. A corrected fit keeps the uncorrected values in fit$raw and records the correction in fit$bias_correct; an empirical-Bayes fit records method = "eb" (default = TRUE when it was the default) and refits nothing.

bootstrap integer: if > 0, additionally run bootstrap replicates of the parametric or block bootstrap (via ame_als_bootstrap) after the point fit and attach the result as fit$bootstrap. The downstream accessors (confint.ame_als, summary, print) then surface bootstrap intervals instead of the anti-conservative sandwich Wald intervals. At least 30 replicates are needed: a smaller positive number is an error of class "lame_als_bootstrap_replicates" raised before anything is fitted. A binary network fitted with bias_correct = "none" has no bootstrap (an error of class "lame_als_bootstrap_bias", also raised before anything is fitted): see ame_als_bootstrap. Default 0 (no bootstrap) – bootstrap is expensive, so it is not imposed on a user who just wants a quick fit.

bootstrap_type character: "parametric" (default) or "block" – the bootstrap scheme to use when bootstrap > 0. The block scheme returns no standard error or interval for the parameters that resampling time slices cannot vary (the node-covariate coefficients and the variance components of the actor effects, among others; fit$bootstrap$withheld lists them with the reason), and the parametric scheme gives them; see ame_als_bootstrap.

bootstrap_block_length
integer: block length for the block bootstrap; only used when bootstrap_type = "block".

`bootstrap_seed` optional integer seed for the bootstrap (the point fit uses `seed`).

`verbose` logical; print progress (default `TRUE`).

`seed` random seed (default `6886`) for the random starts of `multistart = "cheap" / "full"`; the caller's random number state is restored afterwards. The block coordinate descent itself is deterministic, so with `multistart = "none"` the point estimate does not depend on it.

`dcor` logical; whether a directed binary probit, Poisson or normal fit models the dyadic correlation of the latent errors of the two cells of a dyad, as `ame` and `lame` do with their `dcor = TRUE`. The default is `!symmetric`, the default of the samplers. With the empirical-Bayes estimator (the default of a binary fit) the correlation is part of the model that is fitted: the two cells of a dyad are one bivariate probit observation, `fit$VC["rho"]` is the correlation of that model, and the factors, the standard errors and the predictions are those of the model (see **Dyadic correlation** in Details). `dcor = FALSE` fits independent cells, the model of the earlier release, and reports the correlation of the pairs of cells at the fitted predictor as before. A directed count fit takes the pairs at its fitted state (see **Dyadic correlation of counts** in Details), and a directed normal fit takes the two cells of a dyad as one bivariate normal observation, in closed form (see **Dyadic correlation of normal data** in Details; it costs 1.3 to 1.6 times the independent-cell fit). A fit that has no pair likelihood (undirected and bipartite networks, the other estimators of a binary fit, and the fixed-effects normal fit) keeps its own estimate of `rho`; an explicit `dcor = TRUE` with the logit link is an error, because the latent correlation is that of normal errors. The pair likelihood of a binary or count fit costs 1 to 2.5 times the independent-cell fit for one period and 2 to 4.5 times for four, in proportion to the number of dyads and periods; `dcor = FALSE` is the fast path for very large networks. A directed normal network whose reciprocal pairs are all exactly equal (an undirected network given as a directed one) has an improper posterior for the correlation under the prior of the samplers, and is refused as `ame` refuses it (class `"lame_improper_normal_reciprocity"`): fit it with `symmetric = TRUE`, or with `dcor = FALSE`.

`prior` `NULL` or a named list of priors. The estimators read one key, `s20`, the scale of the inverse-gamma prior of the variance of the noise of a Poisson or a normal fit (one degree of freedom on `s20`; the default is 1 for counts and the variance of the observed responses for a normal response, the priors `ame` and `lame` draw it under). The other keys of the samplers' prior list have no counterpart in the estimators; they are reported and ignored. The prior of the multiplicative effects of a directed network is the samplers' default (`ame`, `Suv0` and `kappa0`).

Details

The AME decomposition

$$z_{ij} = \mu + \beta' x_{ij} + a_i + b_j + u_i' v_j + \epsilon_{ij}$$

is conditionally linear: it is a linear regression in (μ, β, a, b) for fixed multiplicative factors (U, V), and the optimal rank-`R` (U, V) for fixed (μ, β, a, b) is the truncated SVD of the residual matrix. The estimator therefore cycles, until the residual sum of squares stabilises, through three blocks — each a monotone (objective-non-increasing) update: a joint least-squares solve for the intercept

μ and regression coefficients β (an exactly rank-deficient design is refused with an error of class `"lame_design_rank"` that names the aliased columns); an exact solve of the additive effects (a, b) (Gauss-Seidel sweeps started from the direct solution of their normal equations, and an exact eigen solve wherever the sweeps do not reach the normal equations, as on a weakly connected observed-dyad graph; a residual above the tolerance after the exact solve stops the fit with an error of class `"lame_als_numerical"`); and a weighted low-rank update for the multiplicative factors (U, V) via SVD (eigen-decomposition when `symmetric = TRUE`). The weighting makes the multiplicative step correct for unbalanced longitudinal panels (dyads observed at unequal numbers of time points); severely unbalanced panels converge in more iterations. A fixed-effects fit that has not reached its criterion at `max_iter` is refitted once with four times that limit, and refused with an error of class `"lame_als_convergence"` if it still has not: the message gives the stop (iteration limit, or no step that lowers the objective), the score against its threshold, and the remedy. The last iterate is not returned. An empirical-Bayes fit that does not settle is refused with the same class at once, with the state it stopped in and the remedy.

For fixed-effects Gaussian fits (`bias_correct = "none"`) and fixed-transform fits, convergence additionally requires a centered first-order SSE score no larger than `sqrt(tol)`. A premature plateau or exhausted outer budget receives up to five joint refinements, each bounded by `max_iter` optimization iterations. These preserve the same objective and observation mask; a requested single sweep remains unfinished. `gaussian_diagnostics` records the initial and final scores, objective values and refinement counts. Multi-start fits retain these checks for every start in `multistart_gaussian_diagnostics`; warm refits and the Gaussian bootstrap use the same certificate. A first-order certificate attests a stationary point, a local optimum of the non-convex factor model when `R > 0`.

With `bias_correct = "none"` a `family = "normal"` fit is a least-squares (Gaussian maximum-likelihood) fit — the exact global solution when `R = 0`, and a local optimum of the non-convex low-rank objective when `R > 0`; the default for a normal response is the empirical-Bayes estimator of **Normal data (the Gaussian empirical-Bayes estimator)** below. For `"binary"` and `"poisson"`, the estimator uses an IRLS working-response path by default, so coefficients are on the requested link scale (binary probit/logit, Poisson log). The fixed-transform path is still available as a faster exploratory score, but its coefficients are not calibrated effect sizes. For `R > 0` the multiplicative factor block of the IRLS families solves a penalized (MAP) sub-problem: each factor row carries the ridge implied by the AME prior $[u_i, v_i] \sim N(0, \Sigma_{uv})$, with the per-column prior scale estimated by an EM step (the analog of the MCMC Σ_{uv} draw). This is what makes the estimate well-defined: the *unpenalized* rank-`R` binary MLE does not exist under quasi-separation, and fitting it directly inflates every coefficient (a runaway that grows with `R`, with the latent variance diverging). With the penalty the latent variance does not run away.

Empirical-Bayes estimator (the default of binary, Poisson and normal fits). A binary or Poisson IRLS fit, or a normal fit, with `bias_correct = NULL` (or `"eb"`) treats the sender and receiver effects as the MCMC `ame` / `lame` do, as draws from a Gaussian prior $(a_i, b_i) \sim N(0, \Sigma_{ab})$ (an undirected fit has one effect per actor and a scalar variance, a bipartite fit independent sender and receiver variances), instead of as free fixed effects, and estimates the model by empirical Bayes in two stages. The first is the posterior mode with a Laplace correction, in three parts. (1) The IRLS working problem carries the penalty $\sum_i \alpha_i' \Sigma_{ab}^{-1} \alpha_i$, where α_i is the actor's effects less the part its node covariates explain (the node-covariate coefficients are fixed effects of the prior mean, as in the MCMC model); the factor block is the penalised ridge above, whose scales are updated by EM from the posterior second moments of the factors, an undirected fit's too. At every reweighting step the intercept, the coefficients and all the effects are solved jointly and exactly from dyad-level sums, in one dense system with no centring step. (2) Σ_{ab} is updated by an EM step from the effects and their Laplace covariance, under the inverse-Wishart prior that `lame()` puts on it for binary data, so

that the penalty is the MCMC prior; without that prior the update drives the variance to zero on sparse networks and with a latent term. (3) The coefficients solve the Laplace-marginal estimating equations instead of the penalised-fit ones: the score of every cell loses half the derivative of the log determinant of the effects' information (the integral over the effects). The penalised fit alone had a slope too small by up to 4.3 percent with 24 percent ties and 6.8 percent with 7 percent ties (30 networks a cell). The latent factors get the same cell correction, so one objective, the penalised deviance plus that log determinant, is descended. This stage is what `fit$irls_diagnostics` certifies and what the marginal likelihood `fit$eb$log_marginal` (see **Choosing the rank**) is evaluated at.

Variational posterior (the values an empirical-Bayes fit reports). The mode is the starting point of the second stage, not its result. The mode of a latent factor that the data barely identify lies at the edge of a broad posterior and is too large (it fits noise), which inflates the coefficient of a covariate that follows the product and, one rank above the generating one, the slope. The reported values (`a`, `b`, `U`, `V`, `UVPM`, the coefficients and the variance components) are those of a Gaussian variational posterior of the sender and receiver effects and of the factors, independent across actors and between the sender and the receiver side of the factors; the coefficients and the variance components remain empirical-Bayes point estimates. Under this posterior the predictor of a cell has a mean, the fitted predictor, and a variance, that of the effects plus that of the product, $\hat{u}_i'\Omega_j\hat{u}_i + \hat{v}_j'\Sigma_i\hat{v}_j + \mathrm{tr}(\Sigma_i\Omega_j)$ (Σ_i and Ω_j are the posterior covariances of a sender's and a receiver's factor row), and the score and the information of the likelihood are expected over that Gaussian (nine-point Gauss-Hermite quadrature) instead of taken at its mean. Every equation of the fit uses the expected score and information, and the system of a factor row adds the second moment of the other side (mean product plus covariance). The means and these covariances are the stationary point of one objective, the expected log likelihood less the Kullback-Leibler divergence to the prior, at the scale of the prior of the factors; the step of the iteration is halved when the objective would fall, which leaves the fixed point unchanged. A fit without a latent rank is variational too (the effects only). `fit$eb$estimator` says which estimator produced the values (`"variational"`, or `"laplace"` for the mode alone), `fit$eb$variational` records the iteration (convergence, stop reason, number of iterations, the largest step a full update would still take, the lower bound and its parts, and the scale of the prior of the factors) and `fit$eb$factor_posterior` the covariances. A fit whose mode or whose iteration stops short of its fixed point is not returned: the call stops with an error of class `"lame_als_convergence"` that reports the state and the remedy (a smaller rank, a larger `max_iter`, or `ame` / `lame`); the values of an iteration that has not settled are never reported as the estimate, nor replaced by the mode.

The scale of the prior of the factors. The rows of the factors have a Gaussian prior whose covariance the data inform weakly where there are few events per loading. The fit estimates it by the variational update of the inverse-Wishart prior `ame()` and `lame()` put on it (`kappa0` degrees of freedom and scale matrix `kappa0 * Suv0`, the data-scaled default of `ame`), the inverse of the posterior mean precision, $(\Psi + SE)/(\mathrm{df} + n)$, with SE the second moments of the stacked sender and receiver rows. What second moments to use decides the result. The mean-field covariance of a row counts the uncertainty of the other side twice (the second moment of the other side in its system is the mean product plus the covariance) and is below the covariance of the exact posterior; the update evaluated there drives the scale to the floor of its hyperprior wherever the data are weak, and the product and the coefficients that follow it with it. Against an exact Gibbs sampler of the same model run at the same scale (fixed hyperparameters, 8 networks of 32 to 100 actors), the second moments of the rows were 4 to 19 percent below the sampler's at the mean-field covariances (on average 9.5). The default evaluates the update at the cavity covariances, $\Sigma_i = (\Lambda + \sum_j \nu_{ij}\hat{v}_j\hat{v}_j')^{-1}$, where ν_{ij} is the cavity weight of the cell, the curvature of the log likelihood integrated over its Gaussian predictor (the information the cell adds about the row once the rest of the predictor is integrated out), and

these were from 9 percent below to 5 percent above the sampler's (on average 3 below); they are the covariances `fit$eb$factor_posterior` reports (the covariances of the fixed point of the updates, from which the lower bound and a refit start, are kept beside them as `Sig_mf` and `Om_mf`). The covariance of the stacked rows of a directed network is isotropic, one scale and one correlation between the sender and receiver position of an actor (the sampler's prior is a full $2R \times 2R$ matrix: the scale carries one dimension of its uncertainty, see below); a bipartite fit has the priors of its sampler, the receiver factor at unit scale and, for the sender columns, the scales of $P = UGV'$ with standard normal G: the sender rows seen by the likelihood have covariance $G'G$, a Wishart matrix with R degrees of freedom, and the scales of the columns are its ordered eigenvalues, which repel each other (the density of the eigenvalues carries the product of their differences), so a column the data say nothing about falls to the small eigenvalues of that matrix rather than staying at the scale of the prior; and an undirected fit the prior of its sampler on each column (ULU' with $L_r \sim N(0, 1)$ and column precision ψ_r (gamma, shape 1, rate 1/2), so that a column of the factor seen by the likelihood has variance $|L_r|/\psi_r$). The scale is recorded in `fit$eb$variational$scale` (the rule, the scale, the correlation of a directed fit, and the standard deviation of its logarithm). `options(lame.eb_scale = "plugin")` selects the update at the mean-field covariances with a full $2R \times 2R$ covariance and the closed-form rescaling of the factors that it needs, the fit of the earlier release. The option, and `options(lame.eb_scale_uncertainty)` below, also govern the static factors of a normal, binary or Poisson dynamic fit (`lame(method = "als")`), not the persistence of its moving positions.

Dyadic correlation. A directed binary probit fit with the empirical-Bayes estimator and `dcor = TRUE` (the default for directed networks) takes the two cells of a dyad as one observation: the latent errors of y_{ij} and y_{ji} are bivariate normal with correlation ρ, and the likelihood of the pair is the bivariate normal probability $\Phi_2(s_1\eta_{ij}, s_2\eta_{ji}; s_1 s_2 \rho)$, $s = 2y - 1$. The variational posterior is taken in that model. The expected log likelihood of a pair, its score and its information (a 2×2 matrix: the cells of a dyad are not independent given the predictors) are computed under the Gaussian posterior of the two predictors, which are correlated through the effects of the two actors, by a Gauss-Hermite rule of five nodes a side. The equations of the coefficients and of the actor effects carry the cross information of the pairs, the rows of the factors see the exact score of the pair, the cavity weights that set the scale of the factors are those of the pair probability integrated over the predictors (a bivariate normal probability in closed form), and ρ is estimated with the other hyperparameters: at every iteration it moves to the posterior mean of the quadratic approximation of the expected log likelihood in ρ under the prior of the samplers, $2\,\mathrm{Beta}(2, 2) - 1$ (density proportional to $1 - \rho^2$), the one `ame` and `lame` share and the fast fit takes from the same source. The value is `fit$VC["rho"]`, its posterior standard deviation and the method are in `fit$rho_estimation`, and the record of the pair model (the correlation, its score and curvature, the number of pairs, the quadrature) is `fit$eb$pair`. Against the sampler (80 simulated networks of 60 and 100 actors, one and four periods, ranks 0 and 2, with correlations of 0, 0.3, 0.6 and 0.8) the correlation is within 0.02 of the sampler's posterior mean on average and within one posterior standard deviation (0.04 to 0.1) in 79 of the 80 networks, where the post hoc correlation of the independent-cell fit is 0.02 to 0.08 too high; the slope of a covariate is within 0.2 sampler standard deviations of the sampler's on average, and its standard error 1.01 to 1.02 times the sampler's standard deviation, with or without the pair model. The factors of an independent-cell fit absorb reciprocity, which the pair model attributes to ρ. The cost is the pair quadrature, in proportion to the number of dyads and periods: 1 to 2.5 times the independent-cell fit for one period (60 to 150 actors, ranks 0 and 2) and 2 to 4.5 times for four.

Counts (Poisson). A Poisson fit with the default `bias_correct` (or `"eb"`) is the same estimator for the model that `ame` and `lame` sample: a count is $y_{ij} \sim \mathrm{Poisson}(e^{z_{ij}})$ with $z_{ij} = \eta_{ij} + e_{ij}$ and $e_{ij} \sim N(0, v_e)$, where η is the predictor of the sections above (coefficients, sender and receiver effects, latent product). The fixed-effects Poisson fit (`bias_correct = "none"`, the default of the

earlier releases) has no finite estimate for the effect of an actor without a count on a side, which runs to minus infinity (-17.5 after 71 reweighting steps on a network of 30 actors with one empty row and one empty column), and it fits the marginal mean with a post hoc overdispersion. The empirical-Bayes fit gives such an actor a finite effect, shrunk towards the prior mean as in the sampler. The effects have the Gaussian prior of the binary fit, with the scale and the degrees of freedom that `ame()` and `lame()` give counts: the variance across actors of the centred log row and column rates (a half-count pseudo-observation keeps an empty actor finite) and $\mathrm{round}(4 + 3n/100)$ degrees of freedom, n the number of actors; the factors have the priors of the binary fit. The variance of the noise v_e is a hyperparameter, estimated as the covariance of the effects is, by type-II maximum likelihood: it is the value that maximises the Laplace marginal likelihood of the model (`fit$eb$log_marginal`, the mode of the effects and the factors solved at that value) plus the log density of the prior that `ame()` and `lame()` give it (the inverse-gamma of one degree of freedom on the scale one). It is `fit$VC["ve"]` and `fit$eb$ve`, and the objective it was found on is `fit$eb$ve_search$curve`. The stationarity condition of the maximum is the EM step that `ame()` and `lame()` draw v_e from, $(1 + \sum E[e^2])/(N - 1)$ (the sum running over the observed cells, the dyads once for an undirected network, of the second moment of the noise given the count), with the second moment taken, to the order of the Laplace approximation, under the posterior of the predictor of the cell given all the parameters; the step at the mode alone, or under the mean-field posterior of the predictor, which is shrunk where the data inform the latent product weakly, is too large (a count network with fewer than three nonzero counts per loading: 1.3 to 4.6 posterior standard deviations above the sampler's, where the maximum of the marginal likelihood is within 1). A count network without noise beyond the Poisson has a positive estimate, as the sampler's posterior does: the inverse-gamma prior holds v_e off zero (on four plain Poisson panels of 22 actors and five periods the estimate was 0.034 to 0.046, within 0.0015 of the sampler's posterior mean; 0.040 on average over 60 panels). The maximum is found by continuation on $\log v_e$: the mode is solved at 0.05, where every column of the latent product is alive, and then at larger values until the objective falls, then at the vertex of the parabola through the three points about its maximum, each warm from the nearest solved mode (typically 8 to 10 modes, 5 to 30 reweighting steps each); a mode solved cold at a large v_e loses a weakly supported column and keeps it lost at every smaller value, on a lower branch of the marginal likelihood. The variational stage holds v_e at the maximum, and a column the mode lost is restarted from the score of the cells before it iterates. The cell has everything the binary cell has. Its noise is integrated out, so that the log likelihood, the score and the information of a count at the predictor η are those of the Poisson-lognormal distribution (the information is positive: the log likelihood is concave in η); they are the cumulants of the posterior of the noise given the count, computed by an adaptive Gauss-Hermite rule centred at its mode (14 to 64 nodes, by the size of v_e), with the derivative of the information that the Laplace correction needs. The variational stage takes the expectation of the cell over the Gaussian posterior of the predictor with the rule of the binary fit (nine nodes where the predictor is uncertain, fewer where its variance is below 0.3), and the cavity weight of a cell is the information of the Poisson-lognormal cell whose noise variance is v_e plus the variance of the predictor. The linear predictor `EZ` is on the latent log scale of the model (there is no shift by half the noise variance), the mean of a count is $e^{\eta + v_e/2}$, which is what `fit$fitted`, `fit$YPM`, `fitted()` and `predict()` hold (the plug-in mean, as the samplers report it; the posterior predictive mean of a count is not finite in this model, and `plug_in = FALSE` returns the value the integration over the predictor gives, $e^{m + (v + v_e)/2}$). A network without a count above zero has no finite log rate, and a dyadic covariate of one sign that is non-zero only where every count is zero has no finite coefficient: both are refused (class `"lame_als_separation"`). Rank evidence (`fit$eb$log_marginal`) is the Laplace value of the Poisson-lognormal model.

Dyadic correlation of counts. A directed count network with `dcor = TRUE` (the default for directed

networks, as in ame and lame) takes the two counts of a dyad as one observation at the fitted state: the noises (e_{ij}, e_{ji}) are bivariate normal with variance v_e and correlation ρ, and the pair has the bivariate Poisson-lognormal probability. The effects, the factors and v_e are fitted with independent cells (they agree with the sampler's, see below); the pair stage then gives (1) ρ, the posterior mean, under the prior the samplers share, of the profile of the pair probability at the fitted predictors, reported as `fit$VC["rho"]` with its standard deviation in `fit$rho_estimation`, and (2) the information of the pairs: the pair probability integrated over the Gaussian posterior of the two predictors is the pair probability with the noise covariance plus their covariance, and its curvature in the two means (two diagonal weights and a cross weight, negative for a positive correlation) enters the covariance of the coefficients, so that the standard error of a covariate that is the same in the two cells of a dyad is not understated (`fit$eb$pair` records the correlation, its precision, the number of pairs and the cross weights). The pair stage costs 13 to 22 evaluations of the pair probability of every dyad; `dcor = FALSE` leaves it out and keeps the post hoc moment estimate of the correlation of the latent errors (`fit$rho_working` holds it in either case). Undirected and bipartite networks have no pair of cells.

Evidence for counts. Against the sampler (29 simulated count networks of 30 to 50 actors, one and four periods; directed, bipartite and undirected, ranks 0 to 3, mean counts 0.3 to 31, noise variances 0.05 to 2, empty actors, node covariates and reciprocity 0 to 0.8; `ame()` / `lame()`, 30,000 scans after the burn-in) the slopes are within 0.35 posterior standard deviations of the posterior means (median 0.05; the fixed-effects fit 2.8 and 0.6) and the intercepts within 0.38, the standard errors are 0.99 to 1.09 times the posterior standard deviations of the slopes (mean 1.04) and 0.97 to 1.02 times those of the intercept of a cross-section, `va` and `vb` are within 0.3 posterior standard deviations and `ve` within 0.5 (median 0.07; the fixed-effects fit 6.9 and 1.7), the actor effects differ from the posterior means by a median 0.029 of their standard deviation (the fixed-effects fit 0.17, and 2.3 with empty actors), and the correlation of a directed network is within 0.027 of the sampler's (median 0.010 and at most 1.2 of its posterior standard deviations). Without the pair stage the standard error of a slope was 0.77 to 0.87 of the sampler's for dense counts with a covariate that is the same in both cells of a dyad and a reciprocity of 0.6 to 0.8. On 1,500 simulated networks (25 designs of 60: 30 to 60 actors, mean counts 0.1 to 20, noise variances 0.1 to 1.5, ranks 0 to 3, directed, bipartite and undirected, one and four periods, reciprocal designs with a symmetric covariate, and ranks above the generating one) the fit converged in every network (the fixed-effects fit in 94.9 percent, 67 percent at worst), the slope had a bias of 0.6 percent on average (4.7 at most, with 30 actors, mean count 0.1 and noise variance 1.5; the fixed-effects fit -1.8, and -12.1 at worst, at rank 3) and its 95 percent interval covered the truth in 95.5 percent of the networks (88 to 100 by design); its sampling standard deviation was 0.76 times the fixed-effects fit's (median) and the root mean squared error of the actor effects 0.79 times (0.18 at the sparsest). The central 50 and 90 percent predictive intervals of 17,150 held-out cells (10 percent of the cells of 70 networks of seven designs) covered 51 to 54 and 90 to 91 percent of the observed counts, with no departure of the randomised probability integral transform from uniformity except at rank 3 with 3.0 nonzero counts per loading (Kolmogorov-Smirnov p = 0.007). The lowest rank within one unit of the largest Laplace value was the generating rank in 142 of 144 networks of 50 actors, and the rank of the largest value in 143 (each rank is compared at the noise variance that maximises its own marginal likelihood).

Counts with little information about the latent product. With fewer than about three nonzero counts per factor loading (`fit$eb$variational$information["per_loading"]`) the variational posterior shrinks the product. Below 2.1 units of the information-weighted signal per loading (`fit$eb$variational$information["signal_per_loading"]`) the fit withholds the product: `fit$U`, `fit$V` and `fit$UVPM` are `NA`, and `print()` states the statistic and the threshold (see **Choosing the rank of a normal fit**). The noise variance does not take up what the shrunk product leaves: on 22

networks of 40 and 50 actors at ranks 2 and 3 (1.1 to 6.6 nonzero counts per loading) ve is within 0.41 posterior standard deviations of the sampler's (mean 0.18; the step of the variational iteration itself was above it by 1.3 to 4.6, 2.6 on average, below three nonzero counts per loading), on 60 networks of 50 actors at rank 3 and 2.8 nonzero counts per loading it averaged 0.412 for a truth of 0.40 (0.406 at 4.2), and at a rank above the generating rank it is within 0.6 posterior standard deviations of the sampler's (32 networks; 4 percent below it on average). The variational posterior shrinks a product the data determine a little more than the sampler does: its norm is 0.95 to 0.98 of the norm of the sampler's posterior mean product (`lame()`, whose prior of the factors is the fit's) from 2.6 nonzero counts per loading and 0.80 to 0.90 up to 2.1 (against `ame()`, whose prior of the factors is tighter: 0.89 to 0.98 and 0.65 to 0.85); it is as close to the generating product as the sampler's posterior mean is (root mean squared difference 0.99 times the sampler's on average, 0.92 to 1.06 on the 22 networks), the mean of the predictor of a cell differs from the sampler's by 0.15 of the sampler's posterior standard deviation (root mean square over the cells, averaged over the networks; at most 0.23), and its variance is 0.92 of the sampler's (0.74 to 1.03). With about one nonzero count per loading or fewer the iteration still settles: on 54 networks of 40 actors at ranks 1 to 3 with 0.3 to 3.6 nonzero counts per loading every fit converges, none collapses a column of its product, and one, at rank 3 with 0.6 nonzero counts per loading, retires a column (see **A rank the data do not support**); ve averages 0.39, 0.35 and 0.39 at ranks 1, 2 and 3 for a truth of 0.40. The product correlates with the truth at 0.83 on average at rank 3 and 2.8 nonzero counts per loading (the fixed-effects fit 0.71).

Normal data (the Gaussian empirical-Bayes estimator). A normal fit with the default `bias_correct` (or `"eb"`) is the same estimator for the model that `ame` and `lame` sample, $y_{ij} = \mu + \beta' x_{ij} + a_i + b_j + u_i' v_j + e_{ij}$ with $e_{ij} \sim N(0, s^2)$, and it is the estimator of the binary and count fits with the one cell in which everything is exact. The cell of a normal response has the working response y itself, the weight $1/s^2$ and the score $(y - \eta)/s^2$; its information does not depend on the predictor, so the Laplace correction of the coefficients vanishes and the Laplace approximation of the coefficients and the effects is not an approximation: at `R = 0` the fit is the posterior mean (the best linear unbiased prediction) of the linear Gaussian model at the fitted hyperparameters, and `vcov()` is its generalised least squares covariance (checked against an independent computation to 1e-6 in the tests). The effects have the Gaussian prior $(a_i, b_i) \sim N(0, \Sigma_{ab})$ and the inverse-Wishart prior on Σ_{ab} that `ame()` and `lame()` give a normal response (the scale is the mean variance of the row and the column means of the centred response, $\mathrm{round}(4+3n/100)$ degrees of freedom), the coefficients are flat, and the node covariates are fixed effects of the prior mean of the effects. The variance of the noise s^2 is a hyperparameter with the sampler's inverse-gamma prior of one degree of freedom on the scale `prior$s20`, by default the variance of the observed responses. s^2 and Σ_{ab} are the fixed points of the EM images that the samplers draw from (the posterior mean of the inverse-gamma and inverse-Wishart full conditionals), formed from the posterior second moments, so that they are net of the degrees of freedom the effects and the factors use: $s^2 = (s_{20} + \sum E[e^2])/(N - 1)$ with $E[e^2] = (y - m)^2 + v$ (m and v the mean and variance of the predictor of the cell under the posterior). The factors of a latent rank are the variational posterior of the binary fit, with its cavity rule for the scale of their prior and the priors of the samplers (a directed network the shared prior of `ame()` and `lame()`, whose scale is that of the effects). The fitted values are the posterior means of the predictor; the response is Gaussian, so they are also the predictive means.

Dyadic correlation of normal data. A directed normal network with `dcor = TRUE` (the default) takes the two cells of a dyad as one bivariate normal observation with covariance $s^2[[1, \rho], [\rho, 1]]$. Everything is in closed form: the information of a pair is the constant matrix $[[1, -\rho], [-\rho, 1]]/(s^2(1-\rho^2))$, whose cross weight enters the equations of the coefficients and of the effects (the estimates are generalised least squares), the rows of the factors see the exact score of the pair, the cavity weights

of the factor scale are those of the pair density integrated over the predictors, and ρ takes its posterior mean under the prior of the samplers (density proportional to $1 - \rho^2$), computed on a fine grid from the sufficient statistics of the pairs and not by a quadratic approximation. The variance of the noise is estimated from the whitened pairs. The value is `fit$rho` (`fit$VC["rho"]`), its posterior standard deviation and method are in `fit$rho_estimation`, and the record of the pair model in `fit$eb$pair`. `dcor = FALSE` fits independent cells and reports the correlation of the residuals of the pairs as before; undirected and bipartite networks have no pairs.

Standard errors of normal fits. `vcov()` is the Laplace covariance of the coefficients jointly with the effects and the factors, built from the information of the Gaussian cell: the weight $1/s^2$ and the cross weight of the pairs, not the cavity weight that sets the scale of the factors. The information of a Gaussian cell is the same whatever the predictor, so integrating the variance of the predictor into the weight counts it twice: with the cavity weights the standard errors of the dyadic coefficients were 1.05 to 1.13 times the sampler's posterior standard deviations at ranks 1 to 3, with the information of the cell 0.97 to 1.02. Node-covariate and intercept coefficients are covered too, and the part of the uncertainty that comes from the scale of the prior of the factors is added as for the other families. Beyond the designs above the intervals keep their level: on 1,000 simulated normal panels (200 a cell: 40 actors with 8 and 12 periods at ranks 0 and 2, and 120 actors with 4 periods at rank 2; a dyadic covariate, a row and a column covariate, dyadic correlation 0.3) the 95 percent Laplace intervals covered the dyadic coefficient in 0.943 of the panels (cells 0.915 to 0.965), the row and column covariates in 0.959 and 0.956 (0.935 to 0.975) and the intercept in 0.950 (0.935 to 0.955); the Monte Carlo standard error of a pooled figure is 0.007 and of a cell 0.015 to 0.020.

Choosing the rank of a normal fit. `fit$eb$log_marginal` is the Laplace approximation to the log marginal likelihood of the independent-cell model at the posterior mode, solved from the variational state with the variance of the noise held at its fitted value and to a scaled score of 0.01 (the log marginal likelihood moves by at most $ms^2/2$ from the mode, 0.05 for the 1,000 effects and factor entries of 170 actors at rank 2); it is exact at `R = 0`, and it is the value of **Choosing the rank**. The mode is the one `fit$irls_diagnostics` certifies. `fit$eb$variational$information` measures the signal of the fitted product in noise variances per loading (for a binary or count fit, `signal_per_loading`: the product weighted by the information of each cell, which is the same quantity with the fisher weight of the cell in place of `1 / ve`). Below 2.1, in every family, the data do not determine a product of that rank: the sampler's posterior mean product is shrunk towards zero and the variational mean is not, so its size is not the posterior's. Such a fit withholds the product: `fit$U`, `fit$V` (and `fit$L`), `fit$UVPM` and the positions of `latent_positions` are `NA`, `uv_plot` stops with an error of class `"lame_product_withheld"`, and `fit$eb$product_withheld` records the reason, the statistic, the threshold and the log marginal likelihood of every rank from 0 to the fitted one with the rank it chooses, which `print()` and `summary()` state. `fitted()`, `predict()` and `residuals()` are those of the fit (the predictor of the variational posterior), and so are the coefficients, their standard errors and the variance components. The parametric bootstrap draws its replicates from the fit and reports no interval for the positions. The dynamic estimator of a normal panel (`lame(method = "als")` with `dynamic_ab`, `dynamic_beta` or `dynamic_uv`) records the same statistic in `fit$eb$information` and withholds in the same way, with the same threshold for static factors and 1.3 per loading of a period for moving positions (the products of every period: `fit$UVPM`, `fit$Omat`, `fit$Oarr` and the positions with their standard deviations); it has no rank evidence, so its record has none.

Scale of a normal response. The iteration works in the units of the response: its tolerance is relative to the standard deviation of the response and the floors of the variances are relative to its variance. The priors of the factors are the samplers' and, like them, written in the unit of the response (`R/normal_scale.R`): the rows of a directed network have the typical size of the additive effects

(the square root of the mean variance of the row and column means of the centred response) as the scale of their prior, the interaction matrix of a bipartite network and the weights of an undirected one have the variance of the response as the variance of theirs. So the fit of `c * y` is the fit of `y` in other units: on simulated networks of every kind at ranks 0 and 2 the coefficients, the variance components, the effects and the product of `c * y` are `c` (or `c^2`) times those of `y` to 1e-13 for `c` of 1e-3 and 1e3, an undirected panel without factors to 5e-5 (its stop rule), and the standard errors, which carry two refits of the scale of the factors, to 1e-3. A response without variation, or with a standard deviation outside 1e-6 to 1e8, is refused with the remedy (classes `"lame_als_data"` and `"lame_als_scale"`).

The earlier normal fit. `bias_correct = "none"` returns the fixed-effects fit of the earlier releases unchanged: the least-squares fit by block coordinate descent (the global solution at `R = 0`, a local optimum of the non-convex low-rank objective otherwise), with the conditional sandwich covariance of the intercept and the dyadic coefficients, which holds the additive and multiplicative effects fixed.

Validation of normal fits. Against the samplers' posterior means (55 simulated networks of 30 to 50 actors a side, directed, undirected and bipartite, ranks 0 to 3, 38 against `ame` and 17 against `lame` fits, 12 of one period and 5 of four periods, of 20,000 scans after 3,000 of burn-in), the 245 coefficients lie within 0.28 posterior standard deviations (mean 0.04), the Laplace standard errors are 0.91 to 1.05 times the posterior standard deviations (mean 1.00; the posterior standard deviation of the intercept of a `lame()` fit is that of the level of the centred effects, 2.4 to 5.7 times smaller than the standard error of the population intercept, and is left out of this comparison), the variance of the noise is within 0.95 posterior standard deviations, the variances of the sender and receiver effects within 0.16 and the dyadic correlation within 0.45. On 103 simulated networks of 40 actors a side (the 55 above with 48 more of weak and moderate signal; 89 with a latent rank, 1 to 3, directed, undirected and bipartite; against `ame` and `lame` fits of 20,000 scans) the norm of the double-centred product is 0.993 to 1.146 of the sampler's posterior mean product where the fit has at least 2.1 noise variances of signal per loading (39 networks; correlation of the products 0.984 to 1.000). Below (50 networks) the undirected fits are 1.37 to 6.9 times the sampler's (0.31 to 1.73 per loading), the directed and bipartite ones 1.08 to 5.0 below 0.7 per loading and 1.01 to 1.25 from 0.7: the sampler's posterior mean is shrunk towards zero and the mean field is not. Every network from 1.99 per loading has a norm within 0.9 to 1.15 of the sampler's, and the largest that does not is at 1.73 (pooled with the binary and count networks below, the threshold is 2.1). Evaluating the update of the scale of the prior at the posterior mean of the inverse-Wishart (`options(lame.eb_scale_stat = "mean")`) or at the mean-field second moments does not close the gap (median 2.25 and 1.74 times the sampler's below one per loading, against 2.14). The coefficients of the fits below 2.1 lie within 0.30 posterior standard deviations of the sampler's, with standard errors 0.93 to 1.05 times its posterior standard deviations. On 118 binary and count networks of 40 actors a side with a latent product (directed, undirected and bipartite, ranks 1 and 2, against `ame` fits of 20,000 scans; 71 binary networks of 8 and 25 percent ties and 47 Poisson-lognormal ones) the training events per loading do not measure the information about the product: a bipartite binary fit with 5.6 events per loading had 4.4 to 5.7 times the norm of the sampler's product, and a count fit with 13 nonzero counts per loading 7.5 times it. The information-weighted signal does: below 2.1 (79 networks) the binary products are up to 6.2 times the sampler's (median 1.47; undirected up to 6.2, bipartite 5.7, directed 2.9) and the count products up to 7.5 (median 1.87), and these are withheld; from 2.1 the norm is 0.83 to 1.00 times the sampler's for a binary fit (19 networks: the variational posterior shrinks the product somewhat more than the sampler does) and 0.94 to 1.12 for counts (20 networks). The largest network outside the rule is at 2.03 per loading and the next at 2.17. In repeated sampling (21 designs of 100 to 200 networks, 4,100 networks and 19,700 coefficient fits: every kind, ranks

0 to 3, 20 to 80 actors, one and four periods, 20 percent missing cells, a rank above the generating one and a weak product) the 95 percent Laplace intervals cover 95.2 percent (dyadic coefficients 95.1, node covariates 95.9, intercept 94.3; designs from 94.1 to 96.7). The rank with the largest `fit$eb$log_marginal` is the generating rank in all of the networks (252 networks of 40 actors, generating ranks 0 to 2) with at least 8 noise variances of signal per loading, in 95 percent from 4 to 8, 89 percent from 2 to 4, 50 percent from 1 to 2, and in none below 1, where it picks a smaller rank; it picked a larger rank than the generating one in one network (a rank-0 network). Of 300 random designs (10 to 60 actors, one and three periods, up to 30 percent missing cells, fitted ranks 0 to 3 against generating ranks 0 to 2), all 300 fit: the two that the iteration did not settle at rank 3 (undirected panels of 14 and 10 actors over generating ranks of 0 and 1, a limit cycle of the scales of the columns) retire two columns (the rank-1 fit) and converge at rank 3 under the priors in the unit of the response.

Speed. The quadratures of the empirical-Bayes fit (the expected working problem of a cell and of a pair of cells, the cavity weights, the bivariate normal probability of the pair likelihood, the log-domain integral for its small values) and the sums over the panel that its equations need run in compiled code; the reference implementations are kept in R and agree with the kernels to rounding (typically 1e-12, and 1e-9 at worst on realistic predictors). The kernels split their cells and pairs over OpenMP threads where the platform has them: `options(lame.als_threads = k)` sets the number of threads (default 1), and the estimates do not depend on it. Do not combine threads with forked workers such as `parallel::mclapply()`. The dense solves of the effect step (a Cholesky inverse of the stacked sender and receiver effects, 600 by 600 at 300 actors) are the cost of a large panel and run in the BLAS of the R installation, whose speed sets that cost.

Standard errors. `vcov()` and `confint()` return the Laplace approximation to the covariance of the coefficients: the inverse penalised information of all the parameters (coefficients, effects and factors) at the cavity weights (for a normal fit the information of the Gaussian cell, **Standard errors of normal fits**), with the second moments of the factors in the curvature of the other side, plus the part of the uncertainty that comes from the scale of the prior of the factors. The logarithm of the scale has, to the second order, a normal posterior whose precision is the information of the data about it (the derivative of the log marginal likelihood in the log scale, $(\mathrm{tr}(\Sigma^{-1}SE) - np)/2$ at the cavity second moments, from two refits at the scale times $e^{\pm 0.15}$) plus that of the hyperprior; the coefficients move with it by $J = d\theta/d\log s$, and the covariance gains JJ'/c. This is one dimension of the uncertainty of the scale (the sampler's covariance of the rows is a full matrix, with 10 free parameters at $R = 2$); `options(lame.eb_scale_uncertainty = FALSE)` leaves it out.

A rank the data do not support. A sparse binary or count network at a rank above what it supports (or a network with no signal at all) leaves the iteration at that rank with no stationary point the data determine: a column the data barely inform keeps a live scale, its rows are fitted at the scale of the prior and their second moments are the prior's, so that the posterior mode drifts (the largest score of the stationarity certificate rises with every accepted step) or the variational iteration stops improving at a residual of a few thousandths, or enters a limit cycle in the scales of the columns. Such a fit is not refused. The five-node rule of the pair likelihood of a directed binary network integrates a dyad to less than the gain of a step near the fixed point where the rows are weakly informed, which is a limit of the quadrature as often as of the data: a stall of the pair rule is first refined, to seven and then to nine nodes a side, from the state it reached (`fit$eb$pair$refined`). An iteration that is still not settled retires its weakest column, the one with the smallest norm of the product $\|u_k\|\|v_k\|$ (the smallest $|l_k|$ of an undirected fit), and the fit of the next rank down starts from the other columns; this repeats while the iteration does not settle. The fit is the one of the rank it reached: `fit$R` is that rank, `fit$R_requested` the rank asked for, `fit$eb$retirement` the record, and the evidence, the standard errors and the factors are those of the effective rank; the

fit says so with a message of class "lame_als_retirement" when it is made, and print() and summary() say what was retired. A fit that still does not settle with no column left is refused with an error of class "lame_als_convergence" that names the rank asked for, and so is one that ran out of a max_iter below the default of 200 (a bound the caller chose: the iteration may only need more steps, so it is not reduced in rank). Three sparse binary networks of 50 actors at ranks 2 and 3, YX_bin at rank 4 and a network without a product at rank 2 settle at the rank asked for with the five-node rule, and a count network of 40 actors at rank 3 settles by retiring a column (rank 2). The data determine none of the six products (0.00 to 0.39 units of signal per loading), and every one is withheld.

Predictions. The response of a normal fit is its linear predictor plus Gaussian noise, so fitted() and predict() return the posterior mean of the predictor (fit$fitted), which is the mean of the response; the rest of this paragraph is about binary and count fits. fitted() and predict() of a binary empirical-Bayes fit return predictive probabilities: the link integrated over the posterior of the linear predictor rather than applied at its mean. The predictor of a cell is uncertain, so the plug-in probability of a held-out event can be orders of magnitude too small. A Poisson fit returns the plug-in mean count $e^{\eta+v_e/2}$ by default, as the samplers do: the posterior predictive mean of a count is not finite in this model (the inverse-gamma tail of v_e, and the inverse-Wishart tail of the scale of the factors for a cell the data do not inform), so the value of the integrated mean $e^{m+(v+v_e)/2}$ would be a property of the quadrature and of the prior; it stays available as plug_in = FALSE, and the predictive law of a held-out count (its quantiles and event probabilities, which are finite) is what the held-out scores below use. On the 170-state panel of Chapter 11 (10 periods, 15 percent of the cells held out, 39 and 45 events on the two masks) the log loss of the held-out cells at R = 1 to 3 was 0.00427, 0.00440 and 0.00450 (first mask) and 0.00506 at each rank (second mask, where every fit at R = 1 to 3 retires its columns and returns the rank-0 fit) under the plug-in values, and 0.003930, 0.003934 and 0.003972, and 0.004568, under the predictive ones (sampler, one chain, from its draws: 0.003902, 0.003886 and 0.003897, and 0.004480, 0.004448 and 0.004449); the smallest probability of a held-out event was 7e-7 to 3e-6 (first mask) and 7e-9 under the plug-in values and 4.5e-5 to 5.9e-5 and 5e-6 under the predictive ones. At R = 0 the predictive log loss, 0.004054 and 0.004568 on the two masks, is the sampler's (0.004049 and 0.004585). The sampler's own predictor, replaced by a Gaussian with its mean and variance, loses 0.003963, 0.003942 and 0.003947 on the first mask against 0.003902, 0.003886 and 0.003897 from the draws, so at ranks 1 and 2 the gap of the fast fit to the sampler is in the shape of the sampler's posterior of the predictor, not in its moments (at rank 3 the fast fit is 0.00003 above the Gaussian replacement). The predictor is $N(m, v)$: m is the fitted predictor, the posterior mean; v is the variance $g'Sg$ of the cell's additive predictor (the coefficients and the two effects, with g its gradient and S the inverse of their penalised information at the expected weights) plus the variance of the product. The probability is $\Phi(m/\sqrt{1+v})$ for a probit link and $\mathrm{plogis}(m/\sqrt{1+\pi v/8})$ for a logit link. plug_in = TRUE returns the link applied to the fitted predictor (what fit$fitted holds and simulate() uses). Forming the covariance costs one dense inverse, of the size of the one behind vcov(), per call.

Choosing the rank. fiteblog_marginal is the Laplace approximation to the log marginal likelihood of the data with the effects and factors integrated, at the posterior mode that the variational stage starts from (for a normal fit the mode at the fitted variance of the noise, **Choosing the rank of a normal fit**), with that mode's coefficients and scales: the deviance and the penalties of the effects and the factors, with the log determinant of the information of all of them, on the scale of the likelihood (an undirected network counts each dyad once; the hyperpriors of the scales are left out). The deviance falls with the rank; this does not, so the difference between two ranks fitted to the same data is an approximate log Bayes factor. The sign and label multiplicity of the factors is not counted, which favours the lower rank. On 1,040 simulated binary networks (directed, bipartite

and undirected; generating rank 0 to 3, fitted ranks 0 to 3; 60 and 100 actors, one and four periods, 7 to 25 percent ties and two sparser designs) 17 of the 4,160 fits were refused as not converged (class `"lame_als_convergence"`), all in the two sparsest designs without a latent term; of the 1,033 networks fitted at all four ranks, the rank with the largest value was the generating rank in all of the 309 with at least 10 training events per factor loading, in 96 percent of the 386 with 3 to 10, in 59 percent of the 234 with 1 to 3 (35 percent were too low) and in 38 percent of the 104 with fewer than 1 (27 percent were too high, 36 percent too low); the lowest rank within one unit of the largest value was right in 100, 96, 62 and 43 percent. With little information per loading the data do not determine the rank. Take the lowest rank whose value is within one unit of the largest: it named the generating rank more often than the largest value below 3 events per loading (62 against 59 and 43 against 38 percent) and as often from 3 to 10 (96 percent). The variational lower bound, `fit$eb$variational$elbo`, is not the value to use for this. A Gaussian posterior cannot hold the broad posterior of positions that the data barely identify, and the bound's slack grows with the rank: on the 1,033 networks its largest value was the generating rank in 35, 56, 96, 100 and 100 percent (fewer than 1, 1 to 3, 3 to 10, 10 to 30 and more than 30 events per loading), never too high but too low for 96 percent of the networks with a latent term and fewer than 1 event per loading, where the Laplace value found the rank in 38 percent (and was too high for 64 percent of the networks without a latent term and fewer than 1 event per loading). On the 170-state panel of Chapter 11 the Laplace value was -1137.8, -1119.3, -1117.2 and -1121.1 at `R = 0` to 3 on the first mask, so ranks 1 to 3 are preferred to 0 (by 17 to 21 units, most at rank 2), as the sampler's held-out log loss prefers them, while the lower bound falls with the rank (-1144.7, -1152.9, -1170.6 and -1195.9). On the second mask every fit at `R = 1` to 3 retires its columns and returns the rank-0 fit (Laplace value -1117.3, lower bound -1125.1), so neither value can prefer a latent term there, although the sampler's held-out log loss does. The value is the evidence of the independent-cell model at the posterior mode: a fit with a dyadic correlation (**Dyadic correlation**) reports the same `log_marginal` as the fit with `dcor = FALSE`. On 15 simulated networks of 60 actors with a latent correlation of 0.6 (generating ranks 0 to 2) the rule above picked the generating rank in 10 and a higher one in none.

What picks the rank where the Laplace value cannot is the held-out predictive log score: mask a share of the observed cells (15 percent), refit at each rank, and score the masked cells with `fitted()` (the integrated predictive; a variational fit's predictive is not shrunk to the plug-in probability, and its surplus columns are not shrunk away, so the score falls above the generating rank). On the 1,033 networks it found the generating rank in 51, 72, 85, 91 and 92 percent (the Laplace value in 38, 59, 96, 100 and 100), and where there are fewer than 3 events per loading it was right for 64 percent of the networks without a latent term (the Laplace value 72, with the one-unit rule 87; the lower bound 100) and 66 percent of those with one (Laplace 45, the lower bound 32): no other criterion that needs no refit was right for both. The rule that uses the held-out score below 3 events per loading (`fit$eb$variational$information["per_loading"]` of the fit at the rank the one-unit rule names, rank 1 where it names 0) and the Laplace value with the one-unit rule above found the generating rank in 85 percent of the 1,033 networks (the Laplace value alone 83 and 84, the lower bound 82, the held-out score alone 81). On the 170-state panel the held-out log loss at `R = 0` to 3 was 0.004054, 0.003930, 0.003934 and 0.003972 on the first mask and 0.004568 at every rank on the second, whose fits at `R = 1` to 3 retire to rank 0 (the sampler's, one chain: 0.004049, 0.003902, 0.003886 and 0.003897, and 0.004585, 0.004480, 0.004448 and 0.004449; the sampler's loss is lowest at rank 2 on both masks, with rank 3 within 0.00001 of it).

The variational stage needs a median of 46 iterations (87 with fewer than 1 event per loading), and 9 in 10 fits finish within 130. The 170-state panel of Chapter 11 takes 29 seconds at `R = 0` and 1.5 to 3 minutes at `R = 1` to 3 (one start, one thread; 8 minutes at `R = 2` with `multistart = "full"`). The joint solve is dense, so the cost grows with the cube of the number of actors. An

actor without a tie (or without a non-tie) on a side keeps a finite, shrunken effect and stays in the fit, so there is no `fit$not_estimated`. The effects `a` and `b` are not re-centred: the intercept is the population intercept (the level of a dyad between two actors with zero effects, at the means of the node covariates) and the effects have mean zero a priori; without node covariates the posterior-mean effects of the variational fit sum to zero, because the intercept is stationary. `va`, `cab` and `vb` are the fitted covariance of the effects, the EM estimate of Σ_{ab}, not descriptive summaries of the estimated effects.

Accuracy. Against the truth, on the 1,040 simulated networks above (dyadic slope 0.7; sender and receiver sd 0.6, correlation 0.3; variance 0.6 a latent dimension), fitted at the generating rank (above zero), the relative bias of the slope was, by training events per loading (fewer than 1, 1 to 3, 3 to 10, 10 to 30, more than 30), -3.9, -2.4, -0.8, -0.1 and 0.4 percent, with root mean square error 10.4, 7.1, 4.8, 2.8 and 1.8 percent (the posterior mode: -0.7, 0.8, 1.1, 0.7 and 0.8; 9.8, 7.2, 4.9, 2.9 and 1.9). One rank above the generating one the bias was -0.5, 1.4, 2.2, 1.3 and 1.0 percent (mode 0.5, 5.1, 5.2, 2.6 and 1.8), and with no latent term in the data and a latent rank fitted, with fewer than 3, 3 to 10, 10 to 30 and more than 30 events per actor, it was 17.5, 5.7, 4.0 and 1.1 percent (mode 12.4, 8.5, 6.7 and 2.1). At the generating rank the fit withheld the product in 85, 70 and 16 percent of the fits with fewer than 1, 1 to 3 and 3 to 10 events per loading and in none above; the products it kept correlated 0.57, 0.78, 0.87, 0.93 and 0.96 with the generating one and had norm 43, 54, 57, 58 and 52 against the truth's 81, 72, 67, 63 and 54 (the mode, every fit: correlation 0.48, 0.68, 0.85, 0.93 and 0.96, norm 49, 60, 61, 62 and 54 against 69, 62, 63, 63 and 54), and the held-out log loss was no higher than the mode's in any band. Against the sampler's posterior mean, on 101 fits with fewer than 1 training event per loading and 96 with 1 to 3 (directed, bipartite and undirected networks of 60 and 100 actors, ranks 1 to 3, one `lame()` run of two chains each), the slope of the fit lay on average 0.09 (95 percent interval -0.001 to 0.18) and 0.10 (0.05 to 0.16) sampler standard deviations from it (relative to the slope 1.1 and 1.2 percent), and the standard error was 1.05 and 1.03 times the sampler's standard deviation. The fit withheld the product in 90 and 88 of these fits; of the 135 directed fits, 132 withheld it, two with fewer than 1 event per loading retired every column and one kept it. The 9 and 8 products it kept (bipartite and undirected, and the one directed) had 0.74 to 1.08 and 0.92 to 1.07 times the norm of the sampler's posterior mean product. For undirected networks the slope lay 0.40 and 0.34 standard deviations above the sampler's. Where the sampler finds no structure the variational mean keeps a sign pattern from the mode: in 4 of 24 networks of 60 actors its norm is 2.3 to 7.8 times the sampler's posterior-mean norm, which is the small remainder (0.11 to 0.37) of a cloud of mutually near-orthogonal, sign-symmetric draws of about the size of the fitted product (mean cosine between draws 0.004 to 0.11); a unimodal Gaussian posterior cannot report that mean, and the fit withholds these four products (0.30 to 1.18 units of signal per loading). The slope on those four ($|z| \leq 0.83$), its standard error (1.08 to 1.10 times the sampler's standard deviation) and the held-out log loss (0.011 below to 0.002 above the sampler's) are within the usual bands. For bipartite networks of 60 actors fitted at `R = 3` (5 networks, 0.65 to 1.22 events per loading) the slope lies within 1.52 sampler standard deviations of the sampler's ($|z| \leq 0.41$ on four) and the fit withholds the product, whose variational mean has 0.76 to 1.21 times the norm of the sampler's posterior mean product. The scales of the sender columns are the eigenvalues of a Wishart matrix, as in the sampler, so that a column the data do not inform falls to the small eigenvalues: on these 5 networks the variance of the predictor of the held-out cells is 0.57 to 1.02 times the sampler's, and the held-out log loss 0.003 below to 0.014 above its. In cells of 40 networks (directed 30 to 100 actors, bipartite and undirected 60 actors, one and four periods, ranks 0 to 2, 14 cells) the slope bias was -2.2 to 1.2 percent (mode -0.5 to 3.4), the 95 percent interval from `vcov()` covered the slope in 80 to 100 percent of the networks (mean 95) and the intercept in 88 to 98 (mean 92), and the standard error was 0.78 to 1.43 times the standard deviation of the

slope. On the 170-state panel the coefficients of the full-system fit (R = 2) lay within 0.31 standard deviations of the sampler's posterior mean, with standard errors 0.86 to 1.23 times the sampler's; the fit withholds the product (0.26 units of signal per loading, below 2.1), whose variational mean has norm 14.4 against the 15.9 of the sampler's posterior mean product (correlation 0.93). On the collinear-covariate design (40 actors, 8 periods, R = 2, truth 0.5, 24 networks) the covariate that follows the latent product was 28.0 percent too large (sampler 26.2, mode 13.5, no latent rank 93): the prior shrinks the product and the covariate absorbs what it shrinks. The independent-cell fit (dcor = FALSE) has no dyadic correlation. On a strongly reciprocal panel (the school network of Chapter 9, 32 actors, R = 2; ame() with the book's specification, two chains of 50,000 scans, puts rho at 0.67, posterior standard deviation 0.16) its coefficients lie within 0.72 sampler standard deviations of the sampler's, with standard errors 1.07 to 1.14 times the sampler's, and its post hoc correlation is 0.98, 2.0 posterior standard deviations above the sampler's. The pair model (**Dyadic correlation**), the default for a directed network, estimates rho at 0.74 (0.43 posterior standard deviations from the sampler's), and its coefficients lie within 0.74 sampler standard deviations, with standard errors 0.76 to 0.97 times the sampler's. Both fits withhold the product of this panel. Choose the rank as described above.

Without the empirical-Bayes prior (bias_correct = "none" and the two explicit corrections below) a binary fit is the maximum-likelihood fixed-effects probit (or logit) fit, penalised at R > 0: each reweighting step caps the working-response increment at 30 and gives a capped cell the weight at which it still contributes its score, and the loop runs to its fixed point (on 32 simulated networks of 60 and 120 actors the coefficient and deviance matched glm() to 5e-6 and 3e-8). In a fixed-effects fit an actor with a single outcome value on a side (a sender without a tie, a receiver with only ties, and so on) has its effect there at the boundary of the likelihood and its cells on that side carry no information about the other parameters: those cells are left out of the fit (and of fit$Y), the effect is NA in a / b with its linear predictor and fitted values, the actors are listed in fit$not_estimated with a message, and va / vb describe the estimated effects. On simulated 250-actor networks with 1 to 50 such senders the coefficient, va and vb stayed where they were without them.

Split-sample bias correction. A binary IRLS fit is a fixed-effects probit (or logit) fit: every additive effect and every latent position is estimated from the cells of its own actor, and maximising over all of them stretches the latent scale outward in finite samples (the incidental-parameters problem), while the factor penalty pulls the product term in. On simulated 30 x 20 bipartite panels with five periods and R = 2 (20 seeds), the uncorrected dyadic coefficient was 3.3\ 15\ too flat; at 60 x 40 the stretch was 2 to 5\ panels 2 to 6\ split-sample jackknife wherever the jackknife holds (see the size boundary below; the empirical-Bayes default needs no such correction). The observed cells are split into two interleaved halves, the model is refit on each half, warm-started from the full fit with the same settings, and every identified quantity is reported as $2\times$ full minus the mean of the two halves: a half carries twice the stretch of the full panel, so the combination cancels it to first order. A half that has not converged within max_iter is continued from its last iterate with five times that limit; if it still has not converged the call is an error of class "lame_jk_nonconvergence" (the class of the dynamic count jackknife), since the correction never combines a last iterate. The split is the checkerboard of the cells with $i + j + t$ even and odd (every dyad keeps the periods of one parity in each half, every actor about half of its cells); when it leaves some actor without a tie or without a non-tie in a half, a random split of the dyad-periods (drawn from a fixed internal seed; the caller's random number stream is left unchanged) and then the split into odd and even periods are tried, the latter with weights for its unequal halves. Corrected are the intercept and regression coefficients, the additive effects a and b, the double-centred product surface (U, V, L are the rank-R factorisation of the corrected surface) and va, vb, cab; EZ, the fitted values and the residuals are rebuilt from them; ve and the IRLS weights are those of the full fit, and rho is re-estimated

from the paired binary likelihood at the corrected marginal probabilities. On the panels above the corrected coefficient ratio was 0.990 (Monte Carlo standard error 0.010) at 30 x 20 and 1.002 at 60 x 40, the effect spreads within 1.5\ 2\ product slope 0.97 to 1.04. The correction costs two half-data refits, 1.6 to 3.4 times the time of the uncorrected fit, and widens the spread across samples slightly (standard deviation of the coefficient 0.043 to 0.047 at 30 x 20). `fit$raw` keeps the uncorrected coefficients, `a`, `b`, `VC`, product surface `O` and deviance (`fit$deviance` is that of the corrected `EZ`); `fit$bias_correct` records whether the correction was applied, the split used and the splits tried, the half fits' convergence flags and iteration counts, and the shrink factors of the effects and the product. `bias_correct = "none"` returns the uncorrected fit and `bias_correct = "jackknife"` requests the correction for another family.

A single cross-section at `R = 0` (`ame_als()`) is split by the checkerboard of its cells ($i + j$ even and odd, each actor keeping half of its cells in each half). Each half's observed graph then falls into two sender-receiver components whose relative level is not identified, so every cell's half value is taken from the half that observes that cell's block. On simulated directed networks (20 to 40 seeds) the corrected slope, `va` and `vb` were 0.990, 0.982 and 0.954 of the truth at 75 actors with 36\ 100 and 0.999, 0.979 and 0.996 at 150 actors with 24\ 1.017 to 1.029 and up to 1.09). At `R > 0` a single cross-section is not corrected: the checkerboard over-corrected wherever the uncorrected estimate needs it (100 actors, `R = 2`: slope 1.024 to 0.971, `vb` 1.090 to 0.950; 50 actors: `va` 1.209 to 0.791), and so did a random split. Such a cross-section is corrected by the parametric bootstrap instead (see **Parametric-bootstrap bias correction**).

Parametric-bootstrap bias correction. `bias_correct = "bootstrap"` simulates the bias instead of splitting the sample, which reaches the panels the split-sample correction does not (a single cross-section with `R > 0` or below 25 rarer-outcome cells per actor, a panel of fewer than five periods at `R > 0`, an actor without both outcomes in a half). Networks are drawn from the fitted model with fresh additive effects from the fitted `va`, `vb`, `cab` and the intercept, coefficients, node-covariate parts, product surface and reciprocity held at their fitted values, and are fitted by the same uncorrected estimator, warm-started from the fit. The bias of each coefficient, of `va`, `vb`, `cab` and of each cell of the product surface is the mean of its replicate values less the value it was generated with (the variance and covariance of the drawn effects for `va`, `vb`, `cab`); the additive effects are divided by the mean slope of the replicate effects on the drawn ones. The bias grows with the spread of the effects, so it is larger at the stretched estimate than at the truth, and one subtraction over-corrects small networks (30 actors, `R = 0`, 150 networks: slope -2.4\ subtracting the bias at the fitted effects themselves gave 0.85); a second round of replicates at the once-corrected parameters, drawn with the same random numbers, gives the slope s of the bias, and the estimate is corrected by $b_0/(1+s)$, the secant step toward the parameters whose simulated estimate equals the fitted one. Each round has 30 replicates from a fixed internal seed (the fit is reproducible and the caller's random number stream is left unchanged), so the correction costs about 60 refits. On simulated directed probit networks (mean -1, about 27\ ties, dyadic slope 0.7, sender and receiver sd 0.6, factor variance 0.6 at `R = 1`; 100 to 150 networks a cell) the relative bias of the slope went from 12.2\ at 30 and `R = 1`, 6.8\ 0.2\ 1.9\ panels of four periods at `R = 1`, and on 40 x 30 bipartite cross-sections (60 networks) 9.3\ 2.4\ `va` and `vb` went from 1.10 to 1.51 of the truth to 0.94 to 1.08 (Monte Carlo standard error 0.02 to 0.05), the intercept bias from -0.03 to -0.11 to within 0.02, and the slope of the product surface on the generating one from 0.71 to 0.79 (30 actors), 0.86 to 0.93 (50) and 0.94 to 0.98 (100); the rest of its flattening is the factor penalty. The 95\ interval estimate plus or minus 1.96 replicate standard deviations of the first round (`fit$bias_correct$replicate_sd`) covered the slope in 93 to 99\ bootstrap-corrected fits among 100-actor cross-sections at `R = 0`), against 72 to 92\ uncorrected estimate and 49 to 82\ (`vcov()`). The correction took 4 to 8 seconds at `R = 0` and 40 seconds (30 actors) to 9 minutes (200 actors) at `R = 1` with twelve such fits running at once, against

0.2 to 9 seconds uncorrected. `fit$bias_correct` records `method = "bootstrap"`, the replicates of each round that refit and converged, the first-round biases, the secant factors and the corrections applied; `fit$raw` keeps the uncorrected values. A refit of the corrected fit (`ame_als_refit`, and every replicate of `ame_als_bootstrap`) removes these stored corrections rather than simulating again. Fewer than 80\ and converging is an error of class `"lame_als_size"` (1 of 100 networks at 30 actors and `R = 1`), and so is an undirected fit with `R > 0`, whose fitted product is flattened beyond what the replicates reproduce: on undirected 50-actor networks with a rank-1 term of variance 1 (60 networks) the product slope was 0.68 and the correction moved the dyadic slope from 1.0\ variance 0.36 (50 networks) the factor penalty removed the product in every fit and the slope went from 6.5\ Undirected fits at `R = 0` are corrected (50 actors: slope 6.6\ -0.8\

Size boundary. A fixed-effects binary fit is supported where the bias of what it reports is within a tolerance, 2\ `va` and `vb` (the internal constant `.ae_bias_tol`; the floors below follow from it). The size that governs the bias is the number of observed cells per actor (over all periods, on the side with fewer: senders or receivers, rows or columns) times the share of the rarer outcome (ties in a sparse network), called rarer-outcome cells per actor below, and the bias also grows with the spread of the additive effects, measured on the fit as the larger of the sd of the estimated sender and receiver effects over the latent noise sd (1 for probit, $\pi/\sqrt{3}$ for logit). On simulated networks (single cross-sections of 60 to 500 actors at 7 to 40\ four periods, directed, bipartite and undirected, probit and logit, additive-effect sd 0.5 to 2, 3 to 10 seeds a cell) the uncorrected bias fell as $c/$ size, with c growing with the rank and the spread. The split-sample correction is used where it can be computed: panels with two or more periods at `R = 0` and five or more at `R > 0`, and single cross-sections at `R = 0`, in each case with every actor keeping a tie and a non-tie in both halves and, at `R = 0`, from 25 rarer-outcome cells per actor (below that it over-corrected `va` and `vb` to 0.75 and 0.87 of the truth at 14, and to 0.86 and 0.89 at 18). The empirical-Bayes default does not use this rule, which governs the fixed-effects fits of the families and settings the estimator does not cover: a fit the split-sample correction does not reach is reported uncorrected when its size is at or above the floor for its rank and spread, and refused below it (the parametric bootstrap does not reach an undirected fit with a multiplicative term); a fit whose spread is above 2.25, or whose rank is above 8, is refused, corrected or not, since neither the uncorrected nor the corrected estimate was measured there. The floors (rarer-outcome cells per actor; in brackets those at 3\ tolerance up to one Monte Carlo standard error:

rank	spread up to 0.7	0.7 to 1.25	1.25 to 2.25
`R = 0`	73 (46)	73 (46)	95 (60)
`R = 1, 2`	78 (50)	94 (63)	155 (98)
`R = 3, 4`	117 (74)	124 (83)	refused (155)
`R = 5 to 8`	117 (74)	145 (97)	refused (refused)

"Refused" marks a band where no measured size (up to about 175 cells per actor) met the tolerance; ranks above 8 are not measured and refused. For a directed cross-section with 36\ are about 204, 218 and 326 actors at `R = 0`, `1-2` and `3-8`, and half that with two periods. A fit above the floor records in `fit$bias_correct` `method = "none"` with the `size`, the `floor`, the `spread` and a `basis` giving the measured bias band there, and `print()` / `summary()` say that no correction is needed; there is no warning. Above the measured spread or rank, binary ALS stops with an error of class `"lame_als_size"` that names the spread or rank and the measured band; the MCMC `ame()` / `lame()` integrate the effects and positions out and are the calibrated route there. The parametric bootstrap also stops with that class when fewer than 80\ replicates refit and converge (networks simulated from the fit too sparse for the fixed-effects estimator). `bias_correct = "none"`

returns the uncorrected fit at any size, recorded as requested (`fit$bias_correct$requested`); `bias_correct = "jackknife"` requires the correction and is an error (class `"lame_jk_split"`) where it cannot be computed. A bootstrap of a binary fit that carries no correction is an error (class `"lame_als_bootstrap_bias"`), whatever its size: each replicate is refit at the biased estimate and reproduces the bias, which falls more slowly than the standard error (`ame_als_bootstrap`). The default empirical-Bayes fit and the fits corrected by the jackknife or the parametric bootstrap have one.

Rank and ordinal families (`"ordinal"`, `"cbin"`, `"frn"`) are not supported by ALS and raise an informative error; use `ame` or `lame` for those likelihoods.

Row/column (node) covariates. A node covariate broadcasts to a per-actor constant, which is collinear with the additive sender/receiver effect, so its coefficient is not identified by the objective alone. It is identified here by an explicit constraint: the additive effects are taken orthogonal to the node covariates, and `beta_row`/`beta_col` are the corresponding between-actor regression coefficients (the additive effects then carry only the residual heterogeneity). This is the standard estimand under the assumption that the additive effects are uncorrelated with the node covariates; if that assumption is doubtful the coefficient absorbs the covariate-correlated part of the additive heterogeneity. Time-varying node covariates are summarised by their per-actor mean; if a node covariate varies within actor over time the discarded within-actor variation triggers a warning.

Missing covariates. As in `ame` and `lame`, a cell whose dyadic covariate, sender covariate or receiver covariate is missing in its period is unobserved: it is set to `NA` in `Y` before the fit, a message gives the number of cells, and the coefficients are estimated from the complete cells. A missing value is not modelled as 0, which biases the coefficient of a covariate that is not centred. A missing value on the diagonal of a square unipartite covariate is not counted, since the diagonal is unobserved anyway. In an undirected fit (`symmetric = TRUE`) a dyad has one covariate value: a hole in one triangle of a dyadic covariate is filled from the other triangle, and a dyad missing in both is unobserved. The per-actor mean of a node covariate runs over the periods in which it is observed. An actor missing a node covariate in every period has no observed cell on that side: the empirical-Bayes fit leaves its effect there to the prior (zero when the actor has no cell on either side), and a fixed-effects fit reports it as not estimated (`NA` in `a` / `b`, listed in `fit$not_estimated` as `"no observed cell"`). A covariate that is missing everywhere stops the fit, as it does in `ame` and `lame`.

Identifiability of the additive and multiplicative terms. For `R > 0` the additive term $a_i + b_j$ and the multiplicative term $u_i' v_j$ are not separately identified by the objective alone: a broadcast (row- or column-constant) component can sit in either, since a pure sender effect $a1'$ is itself rank one. The estimator imposes the standard AME gauge — the multiplicative term is double-centered (zero row and column means), so all broadcast structure is carried by a, b — which makes the reported a, b, U, V unique given the fitted values and independent of the optimisation path. For a unipartite network the double-centering means are taken over the full matrix, which includes the structurally unobserved self-tie diagonal that the model fills in by its low-rank completion; the additive/multiplicative split therefore carries an $O(1/n)$ dependence on that completion. On a disconnected observed-dyad graph the additive effects additionally have a per-component level shift, which is pinned to a precision-weighted minimum-norm gauge so that `a`, `b` and the variance components remain reproducible. A dyadic covariate that is itself (near) low-rank can still be partially aliased with the multiplicative term, so keep `R` modest relative to the covariate structure. This residual aliasing is intrinsic to the AME model — the MCMC estimator resolves it only through its priors.

Choosing R. An empirical-Bayes fit (the default for normal, binary and Poisson responses) has a rank criterion, `fit$eb$log_marginal` (see **Choosing the rank**). For the fixed-effects fits there

is no automatic order-selection criterion (the working-response objective has no likelihood, so AIC/BIC do not apply). Fit a few values — e.g. `lapply(0:4, function(r) ame_als(Y, R = r, ...))` — and inspect `deviance` (the residual sum of squares for the `normal` family and the transform route, the binomial or Poisson deviance of the fitted values for an IRLS fit, in each case of the reported `EZ`; the optimiser's final objective, which adds the factor penalty of an IRLS fit, is `objective`): it falls steeply while real multiplicative signal is being captured and then flattens, so the "elbow" of that curve is a reasonable choice. With `R > 0` the objective is non-convex; use `multistart` to guard against local optima.

A completely separating unpenalized linear predictor has no finite estimate; the IRLS route refuses it with class `lame_als_separation`. Use `ame()` or `lame()` with proper priors for such data.

Uncertainty. An empirical-Bayes fit (the default for normal, binary and Poisson responses) reports Laplace standard errors for every coefficient (`vcov()`, `confint()`, `fit$eb$se`). For a fixed-effects fit (`bias_correct = "none"`) uncertainty is obtained separately, by the bootstrap; see `ame_als_bootstrap`. (The SIR paper's own primary standard errors are Hessian-based, classical and sandwich/robust. For AME a Hessian-based variance is awkward on two counts: without an explicit gauge fix the rotational invariance of the multiplicative factors leaves the joint Hessian rank-deficient, and even with a gauge fixed the Gaussian working-response approximation used for the non-normal families leaves a Hessian-based variance miscalibrated. The bootstrap side-steps both, so it is preferred here.)

Value

An object of class `"ame_als"`: a list with the point estimates. When `R > 0` the fitted multiplicative surface is exposed as `UVPM` (`U L U'}` for `symmetric fits, which also carry it as \code{ULUPM}; \code{U V'` otherwise), matching the slot names on MCMC fits. estimates `mu`, `beta`, `a`, `b`, `U`, `V` (`L` for symmetric models), the linear predictor `EZ`, response-scale `fitted` values, working-scale `residuals`, convergence information, and the variance-component vector `VC`. `R` is the rank of the fit; where the data did not support the rank asked for and columns were retired, `R_requested` (present only then) is the rank that was asked for (see **A rank the data do not support**). For IRLS, `irls_iterations` counts accepted outer reweighting steps; `iterations` counts the final inner solve (the reweighting steps, for an empirical-Bayes fit). `irls_diagnostics` records the score and variance checks, their threshold, the stopping reason, and each accepted step's improvement under a common factor penalty. These checks concern the uncorrected fit; a split-sample correction also records each half's checks in `bias_correct$irls_diagnostics`. A changing empirical factor variance means successive penalized objective values need not decrease; an empirical-Bayes step is also accepted when it lowers the residual of the estimating equations that the checks evaluate. A fixed-effects fit that does not pass the checks within `max_iter`, nor within four times it, is an error of class `"lame_als_convergence"`: no fit is returned. (An empirical-Bayes fit that does not pass them warns with that class and has `converged = FALSE`.) For an empirical-Bayes fit (`bias_correct = "eb"`) `eb` is a list: `sab`, the fitted covariance of the actor effects (a 2×2 matrix; an undirected fit has one variance and repeats it on the diagonal), `Sab0` and `eta0`, the scale and degrees of freedom of its inverse-Wishart prior, `gamma`, the coefficients of the node covariates in the prior mean of the effects, `sab_change`, the relative change of `sab` in the last update, `sig2_u` and `sig2_v`, the fitted scales of the factor columns (one value per column and side; the isotropic scale of a directed fit repeated, the unit receiver scale of a bipartite fit, one scale per column of an undirected fit), `mu_int`, `a_raw` and `b_raw`, the intercept and the effects as the posterior is built from them (the effects are not re-centred), `log_marginal`, the Laplace log marginal likelihood at the posterior mode (see **Choosing the rank** in Details), `vcov` and `vcov_cond`, the Laplace covari-

ance of the coefficients (intercept, dyadic and node-covariate) jointly with the effects and the latent factors and conditional on the factors, `se`, the square roots of the diagonal of `vcov`, `se_method` and `vcov_error`, the reason when the covariance could not be computed (`vcov()` then falls back to the sandwich), `estimator`, which estimator produced the reported values (`"variational"`, the default, or `"laplace"`, the posterior mode alone), `variational`, the record of the variational iteration (`converged`, `termination`, `iterations`, `residual`, the largest step a full update would still take, and `gain`, the objective's gain at the last step; `step`, the step length; `tolerance`; `elbo`, the evidence lower bound at the plug-in scales, with `ell`, `kl_actors` and `kl_factors`; `information`, the training events, the loadings and the events per loading of a binary or count fit with a latent rank, with `signal`, the double-centred fitted product squared and weighted by the fisher weight of each observed cell, `signal_per_loading` and `threshold`, 2.1, below which the product is withheld (a normal fit: `signal`, the squared norm of the double-centred fitted product summed over the periods, `loadings`, `per_loading`, the signal in noise variances per loading, and `threshold`); `scale`, the rule that set the scale of the prior of the factors (`"cavity"` or `"plugin"`), the scale `s` (the mean over the columns of a bipartite or undirected fit), the correlation `rho` of the sender and receiver position of an actor (directed) and `sd_log`, the standard deviation of the log scale that `vcov()` carries; `start`, `mode_log_marginal`), `factor_posterior`, the covariances of the posterior of the factor rows (arrays `Sig` and `Om`, or `Sig` and the factor `F` with its signs `sg` for an undirected fit; the cavity covariances, with the covariances of the fixed point of the updates beside them as `Sig_mf` and `Om_mf`) and the prior of the fit, `suv`, the fitted $2R \times 2R$ covariance of the factor rows of a directed network, `laplace`, the parts of `log_marginal`, and, for a directed binary probit fit with a dyadic correlation, `pair`: the correlation `rho`, its posterior standard deviation `sd`, the score and curvature of the expected log likelihood in it, the number of dyads with both cells observed `n_pairs`, the nodes a side of the quadrature `quad` (zero for a normal pair, which has none), the rules the quadrature was refined to after a stall `refined` (empty where it was not) and the expected cross weights of the pairs; `retirement`, present only when columns were retired: a list with the `requested_rank`, the `effective_rank`, whether the fit of the effective rank `settled`, and `steps`, one record per retired column (the `rank` it was retired at, its `column`, its `strength` and `share` of the strongest column, and the `stage` and `termination` of the iteration that did not settle); for a normal fit, `ve`, the variance of the noise (also `VC["ve"]` and `fit$s2`) and `ve_prior`, the degrees of freedom and the scale of its prior; and, for a Poisson fit, `ve`, the variance of the lognormal noise of the count (also `VC["ve"]`, `fit$s2` and `fit$pois_latent_var`), and `ve_search`, the search that found it: `ve`, `objective` (the Laplace log marginal likelihood plus the log prior density of `ve` at the maximum), `curve` (a data frame of the values tried: `ve`, `objective`, whether the mode there `solved` its equations and its `iterations`) and `evaluations`. A fit with `multistart = "cheap"` or `"full"` (and `R > 0`) also records, for each start, `multistart_converged`, `multistart_criterion` and `multistart_sse` (the value of that criterion), the kept start `multistart_selected` and, for IRLS fits, `multistart_deviance` and `multistart_distance` (the distance of the fitted predictor from the kept start's, in deviance units); and for every multistart fit `multistart_coefficients`, a starts by coefficients matrix of what each start reports (intercept, dyadic and node-covariate coefficients, with the names and on the scale of `coef(fit)`) whose row for the kept start is `coef(fit)` (the uncorrected `fit$raw$coefficients` of a bias-corrected fit, since the correction is made on the kept start only), and `multistart_product_distance`, the Frobenius norm of the difference between the `UVPM` each start reports (before any bias correction) and the kept start's, over the norm of the kept start's: zero for the kept start. A kept `UVPM` of no size, when the factors of the starts shrink to nothing, is measured against `sqrt(.Machine$double.eps)`, so that starts that agree there are at distance zero. The surfaces themselves are not stored. The variance-component vector has five descriptive entries:

va, vb variances of the sender and receiver additive effects. The empirical variance of the *estimated* effects carries their estimation error as well, so the first-order sampling variance of the additive block — ve times the centred trace of the inverse two-way weighted normal equations — is subtracted, floored at zero, and va/vb estimate the variance of the effects themselves, the quantity ame / lame report. The uncorrected empirical variances are kept as vc_raw and the correction as vc_noise; it is computed conditional on the fitted multiplicative term. For an empirical-Bayes fit they are the fitted variances of the effects themselves (fitebsab).

cab covariance of the *estimated* sender and receiver effects, uncorrected (NA for symmetric or bipartite models); for an empirical-Bayes fit, their fitted covariance.

rho dyadic reciprocity, distinct from the sender/receiver correlation cor(a, b). For a directed binary probit fit with the empirical-Bayes estimator and dcor = TRUE it is the correlation of the fitted model (**Dyadic correlation**), and rho_estimation holds its posterior standard deviation and method = "variational pair likelihood". For other directed binary IRLS fits, this is the Gaussian copula correlation estimated from the paired binary likelihood at the reported marginal probabilities. With a probit link it is the latent error correlation. The working-residual correlation is retained as rho_working; rho_estimation records the number of reciprocal pairs, likelihood, and boundary or unidentified state. Without informative reciprocal pairs rho is NA and conditional network simulation is refused. Poisson IRLS fits estimate latent correlation from the weighted count covariance identity $\mathrm{cov}(Y_{ij}/\mu_{ij} - 1, Y_{ji}/\mu_{ji} - 1) = e^{\rho v_e} - 1$. Independent Poisson noise contributes to the marginal variances but not this covariance. (This is the estimate of a fit without the pair stage, bias_correct = "none" or dcor = FALSE; a directed empirical-Bayes count fit estimates the correlation from the pairs, see **Dyadic correlation of counts**.) With zero latent variance, correlation is unidentified and independent count simulation remains valid. Normal and transformed-response fits use reciprocal residual correlation.

ve residual variance, on the model's own scale and degrees-of-freedom-corrected ($\mathrm{SSE}/(n_{obs} - \mathrm{df})$) so it matches the MCMC posterior-mean s2. For a binary IRLS fit the probit/logit model fixes the latent error variance at 1 by identification, so ve is reported as 1; the working-scale GLM dispersion (about 1 under correct specification) is kept separately as ve_working. For a poisson IRLS fit ve is the variance of the lognormal latent layer of the model ame / lame fit ($y \sim \mathrm{Pois}(e^z)$, $z \sim N(\eta, v_e)$); an empirical-Bayes fit estimates it by maximising the Laplace marginal likelihood (see **Counts**), and the fixed-effects fit (bias_correct = "none") from the count overdispersion $\mathrm{var}[y] = \mathrm{E}[y] + (e^{v_e} - 1)\mathrm{E}[y]^2$ and 0 for equidispersed counts; the working-response mean square is again ve_working.

Poisson level scale. The fixed-effects IRLS fit (bias_correct = "none") solves the marginal log-mean $\mathrm{E}[y] = e^{\eta}$, which is the latent predictor plus $v_e/2$. The reported intercept and EZ subtract that Jensen term, so they are on the same latent scale as ame / lame; the shift is stored as pois_latent_var and added back by fitted, predict(type = "response") and simulate, which therefore do not change. The dyadic slopes are unaffected: the term is constant across dyads. The empirical-Bayes fit works on the latent scale throughout and needs no shift.

These are summaries of the point estimates, not draws from a posterior. See ame_als_bootstrap for uncertainty and vcov.ame_als for a fast analytic covariance of the regression coefficients.

Coverage relative to ame / lame

The ALS estimator is a fast, frequentist point estimator. It covers most static AME workflows, and the top-level lame(..., method = "als") dispatcher covers several dynamic workflows, but

posterior-specific features still require the MCMC estimator:

- **Families.** ALS supports `normal`, `binary`, and `poisson`. For `ordinal`, `cbin`, and `frn`, fall back to `ame` / `lame`.
- **Dynamic effects.** `lame_als` itself fits a static model pooled across time slices. The top-level dispatcher `lame(..., method = "als")` routes supported dynamic requests to a dynamic estimator for normal, binary, and Poisson panels (an empirical-Bayes fit by default; see `bias_correct` in `lame`), including named panels where actors enter or exit: `dynamic_ab`, selected intercept/dyadic/node `dynamic_beta`, AR(1) `dynamic_uv` for directed, symmetric, and bipartite panels, and Student-t `dynamic_uv` for directed and symmetric panels (bipartite t requests are refused). `dynamic_uv_kind = "snap"` has no ALS route and is refused. Node-covariate coefficients use the same orthogonal additive-effect decomposition as `lame_als`; dynamic node coefficients use period-specific node values when selected by `dynamic_beta`, while static node coefficients use per-actor means. A time-varying bipartite interaction matrix (`dynamic_G = TRUE`) is refused on every route: use `dynamic_uv = TRUE` and read `fit$UVPM`. A dynamic normal fit whose latent product the data do not determine withholds it (`fit$eb$product_withheld`; see **Choosing the rank of a normal fit** in `ame_als`). Changing actor composition requires row and column names on every slice so actors can be aligned; smoothing penalties are broken across actor-entry gaps (the empirical-Bayes fit treats an absent actor-period as a free AR state). Rank/censored dynamic families remain on the MCMC path. Static `ame_als()` / `lame_als()` fits still use a single latent rank `R`; the dynamic bipartite ALS dispatcher honours separate `R_row` and `R_col` values.
- **Priors.** `ame_als()`, `lame_als()` and `lame(..., method = "als")` read one entry of `prior`, `s20`, the scale of the prior of the noise variance of a normal or Poisson fit, and the dynamic empirical-Bayes fits read the hyperprior entries of `lame` in their EM. Every other entry of the samplers' `prior` list and `g = ...` are MCMC-only; the dispatcher `ame(..., method = "als")` warns and ignores `prior` and `g` (pass `prior = list(s20 = )` to `ame_als()`). The empirical-Bayes fits apply the default additive-effect prior of `ame` / `lame` with its covariance estimated.
- **Posterior quantities.** No `$BETA` / `$VC` posterior draws on the point fit. Use `bootstrap = N` for a sampling distribution analogue; the draws are bootstrap, not Bayesian.
- **Multi-chain.** Not applicable: an ALS fit is one point fit, `chains` has no effect, and `n_chains` is ignored with a warning.
- **Convergence diagnostics.** No Rhat / ESS / `trace_plot`.

Features that work the same on ALS fits: `coef`, `vcov` (Laplace covariance of an empirical-Bayes fit, sandwich on the regression block of a fixed-effects fit), `confint` (auto-routes bootstrap intervals when present, Wald intervals otherwise), `predict`, `fitted`, `residuals`, `summary`, `print`, `nobs`, `simulate.ame_als`, `gof_plot.ame_als`, `ab_plot.ame_als`, `uv_plot`, `latent_positions`.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

References

Minhas, S. and Hoff, P. D. (2025). Decomposing Network Dynamics: Social Influence Regression. *Political Analysis*. The iterative block coordinate descent estimator adapted here originates with that work (implemented in `sir::sir_alsfit()`).

See Also

lame_als for longitudinal networks, ame_als_bootstrap for bootstrap uncertainty, ame for the full MCMC estimator.

Examples

```
Y <- matrix(rnorm(400), 20, 20); diag(Y) <- NA
fit <- ame_als(Y, R = 1, family = "normal", verbose = FALSE)
coef(fit)
```

---

ame_als_bootstrap *Bootstrap uncertainty for the fast AME estimator*

Description

Computes bootstrap standard errors and percentile confidence intervals for a fast AME fit produced by ame_als or lame_als. Two strategies are available:

parametric (default) Simulates fresh outcomes from the fitted model and refits. For normal and IRLS (binary, poisson) fits the simulation is on the calibrated response scale; for a non-normal transform fit – whose estimator is a Gaussian fit to a fixed transformed response – it is on the Gaussian working scale. The fitted residual variance is df-corrected (mean-model degrees of freedom) at fit time, so simulating with it keeps the replicates centred on the point estimate. Available for both cross-sectional and longitudinal fits.

block Resamples time slices with replacement. With block_length = 1 slices are resampled independently; block_length > 1 draws contiguous blocks (a moving-block bootstrap), which preserves short-run temporal dependence. Longitudinal fits only, and only when the panel supplies at least ceiling(T / block_length) >= 10 resampling blocks: below that the ends of the interval are the extreme replicates (a plain percentile interval covered the dyadic coefficient in 0.83 of the panels at five blocks), so the call warns and uses parametric instead. The returned object records the scheme actually used in type and the one asked for in requested_type. The interval end points are read at expanded percentile levels, and the scheme returns no standard error or interval for the parameters that vary over actors and not over time; see *Block bootstrap* below.

Every replicate is refit by ame_als_refit, warm-started from the original point estimate so that replicates do not drift to different local optima. An empirical-Bayes fit (fit$bias_correct$method == "eb", the default for normal, binary and Poisson fits; see ame_als) has every replicate refit with the same estimator, re-estimating the covariance of the actor effects from the fitted one, and draws the replicate effects uncentred, since the fitted intercept is the population intercept; the intervals then describe the empirical-Bayes estimator, and agree with the Laplace standard errors of vcov.ame_als where those are calibrated. A fit carrying the split-sample jackknife (fit$bias_correct$method == "jackknife") has every replicate corrected the same way, so the intervals describe the corrected estimator; a replicate whose split cannot be refit is dropped and counted like any failed replicate. A fit carrying the parametric-bootstrap correction (fit$bias_correct$method == "bootstrap") has the bias it estimated removed from every replicate rather than re-estimated,

so a replicate costs one refit; the bias moves slowly with the parameters, and the replicate spread is that of the corrected estimator to first order. A binary replicate in which an actor has a single outcome value on a side records that actor's effect there as `-Inf` (no tie) or `Inf` (only ties), its maximum-likelihood value, so the actor's interval is open on that side and its standard error is `Inf`; an actor the original fit leaves out is `NA` throughout. A dynamic ALS fit is refit dynamically, on the parametric route only: a block resample reorders the time axis the temporal penalties are defined along. Only a penalised normal-family dynamic fit (`bias_correct = "none"`) has a bootstrap, and its time-varying factors are not summarised. Every replicate redraws the additive paths (an actor level with the fitted level variance plus an AR(1) deviation), so the standard errors carry the variability of the actors: the intercept's interval is for the population intercept (it covered 0.90 and 0.82 of 100 panels of 20 actors and 8 periods and 40 of 30 and 12, against 0.08 and 0.07 when the fitted paths were held fixed, which covers the realized intercept only) and the dyadic coefficient's covered 0.91 and 0.95. The bootstrap of `lame` (`bootstrap = N`) draws its replicates the same way. The default dynamic empirical-Bayes fit of every family refuses a bootstrap, with an error of class `"lame_dynamic_eb_unsupported"`, and carries standard errors in `vcov()` and `confint()` instead (Laplace ones for a binary or Poisson fit, the exact Gaussian marginal ones for a normal fit). A penalised binary or Poisson dynamic fit refuses it with an error of class `"lame_als_bootstrap_unsupported"`. These refusals come before the replicate count is checked. Standard errors are replicate column standard deviations; confidence intervals use the percentile method.

Replicates. At least 10 valid replicates are needed (`R` below 10 is an error of class `"lame_als_bootstrap_replicates"`; `bootstrap = N` in `ame_als` and `lame_als` needs 30, and a 95 percent percentile interval is best read from 200 or more). A replicate whose refit has not converged is retried on the same data with four times the budget and, failing that, dropped: it never enters an interval. A replicate that errors, returns non-finite values or cannot carry the split-sample correction is dropped too, and every dropped replicate is counted (`n_failed`, with the reason in `fail_reason`). A dropped replicate is usually an extreme one, so the nominal level of a percentile interval can move by at most the share dropped; more than 5 percent of the `R` replicates dropped is an error of class `"lame_als_bootstrap_convergence"`, raised at the replicate that crosses the cap, with no interval from the rest.

One optimum per replicate. The replicates of a fit with a multiplicative term (`R > 0`) are refit from the point estimate and also from the start the original fit used (the cold spectral start and, for a fit with `multistart`, its random starts); the converged refit with the lowest objective is kept, so a replicate cannot sit at a worse local optimum than the original procedure finds on its data. The start each replicate kept is in `boot_refit_start`, and `optimum_check` says what was compared. Without a multiplicative term the optimum is unique and a replicate is refit once. (An empirical-Bayes refit starts from the variational state of the fit and is not compared with other starts.) The point estimate of every coefficient must lie inside the central 95 percent of its own replicates: if it does not, the replicates are distributed around another optimum, or the estimator is biased by more than two standard errors, and the call is an error of class `"lame_als_bootstrap_optimum"` that carries the bootstrap as its `boot` field. A variance component whose estimate lies outside its replicate interval is the bias of its estimator in a regime a percentile interval cannot describe (the variance of the lognormal layer of a count network without extra-Poisson noise is the usual case): its standard error and interval are `NA`, it is named in `vc_uncentred` with the estimate and the replicate range, and the rest of the bootstrap stands.

Block bootstrap. A statistic that varies over B resampling blocks has a percentile interval that falls short of its level: in the simulations below a 95 percent interval covered the dyadic coefficient in 0.90 of the panels over ten to thirty blocks. The end points of a block interval are therefore read at

the expanded percentile levels $\Phi(\sqrt{B/(B-1)}\, t_{B-1,\alpha/2})$ (Hesterberg, 2015): 0.9 and 99.1 percent for a 95 percent interval at ten blocks, 1.6 and 98.4 percent at twenty. `n_blocks` records B, and `confint()`, `tidy()` and the interval columns of `print()` and `summary()` use these levels. The number of replicates matters more here than for a parametric bootstrap, since the end points sit in the tails of the replicates: use 200 or more.

A block resample redraws the time slices and holds the actors fixed, so it carries the variability of the data over time and none of the variability over actors. The parameters whose sampling variability is that over actors have no standard error and no interval under `type = "block"`: they are `NA`, named in `withheld` with the reason, and `print()` and `summary()` say so. `type = "parametric"` gives them. They are the node-covariate coefficients, `va`, `vb` and `cab`, and, for an empirical-Bayes fit, the intercept (the population intercept) and the additive effects and factors, which are shrunk toward their prior mean: the replicates vary around the shrunk estimates, not around the effects. The residual variance `ve` has none either: a refit to a resample, which repeats some slices and omits others, absorbs more of the noise than the original fit, so its replicates sit below the estimate. The intercept of a fixed-effects fit is the mean of its linear predictor over the observed actors (the additive effects sum to zero), and its interval is for that quantity and not for a population intercept.

Coverage of the 95 percent interval was measured on directed normal-family panels of 15 to 40 actors and 10 to 30 periods with a time-varying dyadic covariate, static node covariates, random actor effects (standard deviation 0.6) and reciprocal errors (correlation 0.3), at block lengths 1 to 3, with 100 to 200 replicates a bootstrap and 80 to 150 panels a cell (1720 bootstraps of simulated panels in all, 530 of them with 200 replicates). Coverage of the 95 percent interval, over all panels with ten or more blocks (plain percentile), at the levels above with 200 replicates (expanded), and for the parametric scheme on the same panels:

parameter	plain	expanded	parametric
dyadic coefficients	0.90	0.95	0.93
intercept, fixed-effects fit	0.89	0.93	0.92
`rho`	0.90	0.94	0.94
additive effects	0.90	0.94	0.93
multiplicative term (`R = 1`)	0.91 to 0.92	0.94	0.93
`ve`	0.87	0.89	0.94
node-covariate coefficients	0.07 to 0.11	0.11 to 0.13	0.90
`va`, `vb`, `cab`	0.12 to 0.14	0.15 to 0.17	0.92 to 0.98

The first five rows keep their intervals (for a fit by the fixed-effects estimator, `bias_correct = "none"`). The parametric column is the interval of the parametric scheme with its own corrections. The intercept of a fixed-effects fit is the mean of its linear predictor, so a replicate's intercept carries the projection of the redrawn actor effects on the node covariates, which the realized intercept does not: the replicate intercept is taken relative to it (the interval was 2.6 times the sampling standard deviation, coverage 1.00, before; now 0.92 of 500 panels, standard error 0.94 of the sampling standard deviation), and the node-covariate coefficients keep that variation (0.90). The variances `va` and `vb` are biased and their replicates reproduce the bias (a percentile interval covered 0.82 to 0.89 of the fixed-effects panels and 0.65 to 0.75 of the empirical-Bayes ones), so their parametric intervals are bias-corrected: the basic interval on the log scale for a fixed-effects fit (0.92 to 0.98 at 15, 20 and 40 actors) and the basic interval for an empirical-Bayes fit, whose estimate is biased upward by an amount that does not scale with it (0.91 to 0.97 at 20 and 30 actors). On binary empirical-Bayes panels of 20 actors and 12 periods (80 panels) the block intervals covered the dyadic coefficient in 0.91 of the panels and `rho` in 0.94, and the intercept in 0.19, the node-covariate coefficients in

0.25 and 0.26, va, vb and cab in 0.43, 0.28 and 0.33 and the additive effects in 0.67 and 0.63 (0.93 for the parametric scheme). The same holds for the empirical-Bayes normal fit, the default for the normal family (20 actors, 12 to 20 periods, 160 panels, 200 replicates): the block intervals covered the dyadic coefficient in 0.96 to 1.00 of the panels and rho in 0.93 to 0.98, and the intercept in 0.03 to 0.13, the node-covariate coefficients in 0.05 to 0.10, va, vb and cab in 0.02 to 0.18 and the additive effects in 0.27 to 0.39; ve covered 0.91 to 0.95 and has no interval. The fixed-effects rows above are the fits made with `bias_correct = "none"`.

Fixed-effects fits. A binary fit by the fixed-effects estimator (`bias_correct = "none"`) has no bootstrap: an error of class `"lame_als_bootstrap_bias"`, raised before anything is fitted by ame_als and lame_als. The dyadic coefficient of such a fit carries the incidental-parameters bias, every replicate is refit at the biased estimate and reproduces it, and the percentile interval sits about twice the bias from the truth. A 95 percent interval covered the dyadic coefficient in 0.54, 0.72, 0.76 and 0.84 of 100 simulated panels of 30 actors and one period, 60 and two, 100 and two, and 30 and four; the bias relative to the standard error falls slowly with the number of actors (0.85, 0.71, 0.57, 0.40) and the fit passes the size rule of ame_als at 100 actors and two periods. The default empirical-Bayes estimator, or the split-sample or parametric-bootstrap correction, is the fit to bootstrap. The 95 percent interval of a corrected fit covered the dyadic coefficient in 0.940 (jackknife) and 0.940 (bootstrap correction) of 150 simulated panels of 30 actors and four periods, and in 0.945 and 0.953 of 145 and 150 panels of 60 actors and two (200 replicates; Monte Carlo standard error 0.019). Over those four cells the intervals for the dyadic coefficient, va and vb covered 0.944, 0.940 and 0.943 of 595 panels (standard error 0.010), and the standard error of the dyadic coefficient was 0.99 to 1.04 times its sampling standard deviation; 5 of the 600 panels were refused by the refit rule or the size rule. A replicate of a jackknife-corrected fit repeats the split-sample correction, and a replicate of a bootstrap-corrected fit has the stored bias removed. Scaling that stored bias by each replicate's own estimate changed the coverage of 60 panels in each of two cells by at most 0.02 and is not used; repeating the bootstrap correction in every replicate costs about 60 refits a replicate. A fixed-effects Poisson fit has no incidental-parameters bias in its dyadic coefficient (0.93 to 0.94) or its variance components, but its intercept does: it has no interval under either scheme (0.61, 0.85 and 0.73 of panels of 30 actors and one period, 60 and two, and 100 and two).

Usage

```
ame_als_bootstrap(
  object,
  R = 200,
  type = c("parametric", "block"),
  block_length = 1,
  seed = NULL,
  verbose = TRUE,
  max_iter = 200L,
  tol = 1e-06
)

boot_ame(
  object,
  R = 200,
  type = c("parametric", "block"),
  block_length = 1,
```

```
  seed = NULL,
  verbose = TRUE,
  max_iter = 200L,
  tol = 1e-06
)
```

Arguments

object	an ame_als object from ame_als or lame_als.
R	integer number of bootstrap replicates (default 200); at least 10. A block bootstrap reads the ends of its interval from the tails of the replicates, so use 200 or more.
type	"parametric" (default) or "block"; see Description.
block_length	block length for the block bootstrap: 1 (default) resamples time slices independently; an integer > 1 draws contiguous blocks of that many slices (a moving-block bootstrap), appropriate when the series has short-run temporal dependence. Capped at T; ignored for the parametric bootstrap.
seed	optional integer random seed.
verbose	logical; print progress (default TRUE).
max_iter	positive integer iteration budget for each refit (default 200). A replicate that does not converge is retried on the same data with four times this budget. A static Gaussian or working-scale refit that still fails also tries a fresh spectral start and retains it only if it converges with a lower objective. An unfinished low-rank Gaussian refit also tries seven starts from the fixed internal seed 6886, retaining only converged fits with no worse objective. A replicate that still has not converged is dropped, and more than 5 percent dropped is an error.
tol	positive convergence tolerance for each refit (default 1e-6).

Details

Choice of inference. The Social Influence Regression paper of Hoff & Minhas (2025) derives its primary standard errors from the observed Hessian (classical $-H^{-1}$ and the sandwich/robust estimator $H^{-1}SH^{-1}$). Two features of the AME model make a Hessian-based variance awkward here. First, without an explicit gauge fix the rank-R multiplicative term is identified only up to a full $R \times R$ rotation/reflection ($U \to UR, V \to VR^{-\top}$), leaving the joint Hessian rank-deficient. Second, even with a gauge fixed, the non-normal families are fit on a Gaussian working response, so a Hessian computed from that working objective is not calibrated to the family likelihood. The bootstrap side-steps both issues and is the recommended uncertainty tool here, mirroring sir::boot_sir().

Alignment-sensitive quantities. The regression coefficients beta, the additive effects a, b and the variance components are rotation-invariant and are aggregated directly. The multiplicative factors U, V are *not*: each replicate is Procrustes-aligned to the original fit before its standard errors are computed. The object stores both the raw and aligned replicate factors so the effect of alignment can be inspected. The alignment identifies a column only when its singular value is separated from its neighbours': two adjacent singular values (eigenvalues, for a symmetric model) count as separated when their gap in the point estimate is at least 5 bootstrap standard deviations of that gap (sv_separation$threshold; the rotation the alignment applies to the pair is then under 4 degrees

at the median and under 10 at the 90th percentile, against 9 and 25 when the two are equal). The U and V columns that are not separated from a neighbour have NA for the column-wise standard errors (se_U, se_V, and their unaligned _raw versions) and NA intervals in confint(). What is reported for every fit with a multiplicative term, separated or not, is the uncertainty of the subspaces the factors span, which a rotation of the columns does not change: the principal angles, in degrees, between each replicate's row-factor (and column-factor) space and the point estimate's (subspace). For a symmetric model the eigenvalues L of the multiplicative term are aggregated and reported as well.

The parametric bootstrap is a full parametric bootstrap of the AME model: each replicate draws fresh additive random effects a, b from their fitted dispersion (va, vb, cab) and fresh residuals, holding mu, beta and the multiplicative term fixed. Regenerating the additive effects (rather than holding the fitted a, b fixed) is what gives the intercept and node-covariate standard errors their actor-level sampling-variability component.

Assumptions. The block bootstrap treats the time slices as exchangeable replicates of the static-effects model — appropriate for that model, but not for strongly trended or serially dependent series (use the dynamic lame there). The block bootstrap needs at least ten resampling blocks, so parametric (the default) is substituted, with a warning, on a panel that supplies fewer than ten blocks. The finite-sample (incidental-parameters) stretch of a binary IRLS fit is removed from the point estimate and from every replicate by the empirical-Bayes estimator (bias_correct, on by default for binary, Poisson and normal fits) or, when requested, the split-sample jackknife or the parametric-bootstrap correction; a binary fit that carries none of them has no bootstrap (*Fixed-effects fits* above). The MCMC ame / lame path gives posterior summaries when that is the target.

Value

An object of class "boot_ame" with components including coefs (replicate intercept + regression coefficients), se, ci_lo, ci_hi, point_est, param_names; vc_* for the variance components; a_coefs, b_coefs, se_a, se_b; U_aligned, V_aligned, U_raw, V_raw, se_U, se_V (when R > 0 and the factors are static; NA for the columns whose singular values are not separated), sv_separation (the singular values of the point estimate, the gaps and their z-scores in bootstrap standard deviations, the threshold, and which columns are resolved) and subspace (the principal angles, in degrees, of the replicate factor spaces to the point estimate's, and their largest angle by side); n_valid, n_total, n_failed, fail_reason (per replicate: "", "nonconverged", "split", "error" or "nonfinite"), max_fail_share, boot_refit_start (the start each replicate kept), optimum_check, n_other_start (replicates that kept a start other than the warm one), vc_uncentred (the variance components with no interval because their replicates are not centred on the estimate, when there are any), withheld (a data frame of the parameters with no standard error or interval, their kind — "coefficient" or "vc" or "effects" — and the reason; NULL when every parameter has one), n_blocks (the number of resampling blocks of a block bootstrap, NA otherwise), estimator_eb (whether the fit is empirical Bayes, which sets the interval of va and vb), type, family.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

References

Minhas, S. and Hoff, P. D. (2025). Decomposing Network Dynamics: Social Influence Regression. *Political Analysis*. The block and parametric bootstrap follow the inference scheme of `sir::boot_sir()` developed for the SIR estimator.

Hesterberg, T. C. (2015). What teachers should know about the bootstrap: resampling in the undergraduate statistics curriculum. *The American Statistician* 69(4), 371–386.

See Also

`ame_als` and `lame_als`, both of which accept `bootstrap = N`, `bootstrap_type`, `bootstrap_block_length`, `bootstrap_seed` to do this work in a single call when you are about to fit the model anyway. `ame_als_bootstrap()` is the post-hoc path (bootstrap an *already-fit* object without refitting from scratch). `ame_als_refit` for warm-start refits; `confint.boot_ame` for the bootstrap CI extractor.

Examples

```
# twelve periods supply the ten resampling blocks the block scheme needs
Y <- replicate(12, { m <- matrix(rnorm(225), 15, 15); diag(m) <- NA; m },
               simplify = FALSE)
fit <- lame_als(Y, R = 1, family = "normal", verbose = FALSE)
# post-hoc bootstrap of an existing fit:
bt <- ame_als_bootstrap(fit, R = 50, type = "block",
                            seed = 1, verbose = FALSE)
# equivalent one-shot call:
# fit_b <- lame_als(Y, R = 1, family = "normal", verbose = FALSE,
#                   bootstrap = 50, bootstrap_type = "block",
#                   bootstrap_seed = 1)
print(bt)
```

`ame_als_refit` *Refit a fast AME model with a warm start*

Description

Refits an AME model by iterative block coordinate descent, initialised ("warm-started") from an existing `ame_als` fit. This is the workhorse of `ame_als_bootstrap`: every bootstrap replicate is refit from the original point estimate rather than from a cold random start, which prevents replicates from converging to different local optima or rotations and is essential for meaningful bootstrap standard errors.

Usage

```
ame_als_refit(
  object,
  Y_new = NULL,
  X_new = NULL,
```

```
  Z_new = NULL,
  max_iter = 30,
  tol = 1e-05,
  verbose = FALSE,
  bias_correct = NULL
)
```

Arguments

object	an ame_als object supplying the warm-start values (mu, beta, a, b, U, V) and the model configuration (family, mode, symmetric, R).
Y_new	optional canonical outcome array [n_row, n_col, T] to refit on. Defaults to object$Y.
X_new	optional canonical design array [n_row, n_col, p, T]. Defaults to object$X.
Z_new	optional canonical *working-response* array [n_row, n_col, T]. When supplied, the model is refit by a Gaussian block coordinate descent directly on Z_new, bypassing the family transform / IRLS reweighting. This is used by the parametric bootstrap of a non-normal transform fit, whose estimator is a Gaussian fit to a fixed transformed response. Y_new is still used for the observed-cell pattern.
max_iter	maximum block coordinate descent iterations (default 30; fewer are needed than for a cold start). On the dynamic route, left unset it keeps the original fit's iteration budget.
tol	convergence tolerance (default 1e-5). On the dynamic route, left unset it keeps the original fit's tolerance.
verbose	logical; print progress (default FALSE).
bias_correct	NULL (default) applies the bias correction object was fit with (object$bias_correct$method), so a refit of a jackknife-corrected fit is corrected the same way, a refit of a fit corrected by the parametric bootstrap has the bias that fit estimated removed (fit$bias_correct$stored is TRUE) and a refit of an empirical-Bayes fit ("eb") is an empirical-Bayes fit, which re-estimates the covariance of the actor effects starting from the fit's; "jackknife", "bootstrap" (re-estimating the bias), "none" or "eb" override it. On a refit the jackknife is computed or refused: a panel the correction cannot carry (a single cross-section at R > 0 or below 25 rarer-outcome cells per actor, fewer than five periods at R > 0, fewer than 25 rarer-outcome cells per actor at R = 0, or an actor without both outcome values in the panel or, under every split, in a half sample; see ame_als) raises an error of class "lame_jk_split". A refit of an uncorrected fit keeps its bias_correct record. Dynamic Poisson fits preserve their estimator (the empirical-Bayes estimator, or the penalised fit with the variance/covariance correction of directed panels) and override it in the same way (see bias_correct in lame); an override of the empirical-Bayes estimator by a penalised one starts cold.

Details

The estimation algorithm is the iterative block coordinate descent estimator of the Social Influence Regression model of Hoff & Minhas (sir::sir_alsfit()), adapted to the AME model, with each bootstrap replicate warm-started from the original point estimate.

A fit from one of the dynamic ALS routes (`lame(..., method = "als")` with `dynamic_uv`, `dynamic_ab` or `dynamic_beta`) stores trajectories rather than a single value per parameter. Such a fit is refit through the dynamic fitter that produced it, seeded from the stored trajectories, and the result is again a dynamic fit of the same class. `Z_new` is not available on that route, which runs its own working-response loop.

A static fit carrying the split-sample jackknife (see `ame_als`) is refit with it: the refit on the new data is followed by its own two half-sample refits, warm-started from it with the same `max_iter` and `tol`, and the corrected values are returned, with the refit's uncorrected values in `$raw`.

Value

An object of class `"ame_als"`; see `ame_als`.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

References

Minhas, S. and Hoff, P. D. (2025). Decomposing Network Dynamics: Social Influence Regression. *Political Analysis*. The iterative block coordinate descent estimator refit here originates with that work (implemented in `sir::sir_alsfit()`).

See Also

`ame_als`, `ame_als_bootstrap`.

Examples

```
Y <- matrix(rnorm(400), 20, 20); diag(Y) <- NA
fit <- ame_als(Y, R = 1, family = "normal", verbose = FALSE)
refit <- ame_als_refit(fit, verbose = FALSE)
coef(refit)
```

---

ame_memory_settings *Display memory usage information for AME models*

Description

Shows estimated memory usage for networks of given size. Memory optimization is automatic, so this is informational only.

Usage

```
ame_memory_settings(n_nodes, R = 2)
```

Arguments

n_nodes	Number of nodes in network
R	Rank of multiplicative effects (default: 2)

Details

Memory levers available in the package:

- Run `compact_ame()` on a fitted model to drop empty slots and, for genuinely sparse posterior means, use sparse storage via `use_sparse_matrices = TRUE`
- Increase `odens` in `ame()`/`lame()` to store fewer posterior draws
- Pass `posterior_opts = list(thin_UV = ..., thin_ab = ...)` to thin the stored latent-factor and additive-effect draws

Value

Invisibly returns memory estimates

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
ame_memory_settings(n_nodes = 12, R = 1)
```

---

ame_memory_usage *Calculate memory usage of AME model components*

Description

Calculate memory usage of AME model components

Usage

```
ame_memory_usage(fit = NULL, detailed = TRUE)
```

Arguments

fit	Fitted AME model
detailed	Logical; show detailed breakdown (default TRUE)

Value

Invisibly returns data.frame with memory usage

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
usage <- ame_memory_usage(fit, detailed = FALSE)
```

---

ame_options *AME model fitting options*

Description

Configure options for fitting AME models. Memory efficiency is now handled automatically based on network size.

Usage

```
ame_options(
  parallel_chains = 1,
  verbose = TRUE,
  odens = 25,
  use_sparse_matrices = FALSE
)
```

Arguments

parallel_chains	Number of parallel chains to run (default: 1)
verbose	Logical; print progress information (default: TRUE)
odens	Output density - save every odens iterations (default: 25)
use_sparse_matrices	Logical; use sparse matrices for storing results (default: FALSE). Set to TRUE if your network is actually sparse (many zero/NA entries) and memory is a concern.

Details

Memory optimization features:

- Redundant matrices (EZ, UVPM) are never stored - they can be reconstructed if needed
- use_sparse_matrices = TRUE: Converts large matrices to sparse format

- Posterior samples are always thinned appropriately

When to use sparse matrices:

- Your network has < 10\
- Memory usage is a critical concern
- You're willing to trade computational speed for memory efficiency

Note: For dense networks (most edges observed), sparse matrices will be slower and may use more memory than dense storage.

Value

List of options to pass to ame()

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Configure options
opts <- ame_options(verbose = TRUE, odens = 25)
opts
```

`ame_parallel` *Run AME model with multiple parallel chains*

Description

`ame_parallel(Y, n_chains = k, ...)` is `ame(Y, chains = k, ...)`: the same chains (seeds `seed + 1000 * (k - 1)`), the same drift check, between-chain gate and extensions, and the same pooled fit. It is kept for the older argument names; `combine_method = "list"` returns the fits of the chains instead of the pooled fit.

Usage

```
ame_parallel(
  Y,
  n_chains = 4,
  cores = n_chains,
  combine_method = c("pool", "list"),
  fitter = c("auto", "ame", "lame"),
  ...
)
```

Arguments

Y	Network data matrix
n_chains	Number of chains to run (default = 4)
cores	Number of CPU cores to use: the chains run forked on Unix (one core per chain at most) and one after the other on Windows. Default is n_chains. Use 1 for sequential processing.
combine_method	Method for combining chains: "pool" (default) or "list"
fitter	Which fitter to use: "auto" (default; pick lame when Y is a list / 3-D array with >1 time slice, else ame), "ame" (force cross-sectional), or "lame" (force longitudinal – required for dynamic_* flags).
...	Additional arguments passed to ame() or lame().

Value

If combine_method = "pool": A single ame object with pooled chains If combine_method = "list": A list of ame objects, one per chain. A call whose chains are not settled after the extensions stops with an error of class lame_chain_unsettled (see ame).

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

lame_multi for the unrelated multi-*panel* wrapper that fits K *distinct* networks with shared regression coefficients. ame_parallel / lame_parallel run K MCMC chains of the *same* model (for R-hat / ESS diagnostics); lame_multi runs one chain across K panels with pooled beta.

Examples

```
# Run 2 chains sequentially
data(YX_nrm)
fit_parallel <- ame_parallel(YX_nrm$Y, Xdyad = YX_nrm$X,
                             n_chains = 2, cores = 1,
                             nscan = 100, burn = 10, odens = 1,
                             verbose = FALSE)
```

---

array_to_list *Convert array to list.*

Description

Convert array to list.

Usage

```
array_to_list(arrayObj, actorList, sliceLabel)
```

Arguments

arrayObj	3d array object
actorList	list of actor names
sliceLabel	labels for array slices

Value

array in list format

Author(s)

Shahryar Minhas

Examples

```
n <- 12
arr <- array(rnorm(n * n * 3), dim = c(n, n, 3))
rownames(arr) <- colnames(arr) <- paste0("a", 1:n)
actors <- rep(list(paste0("a", 1:n)), 3)
Y_list <- array_to_list(arr, actors, sliceLabel = paste0("t", 1:3))
length(Y_list)
```

---

as_draws *Generic dispatcher for posterior::as_draws on lame fits*

Description

Lightweight S3 generic so calls of the form `as_draws(fit)` dispatch through R's S3 system even when the **posterior** package is not loaded. When **posterior** *is* loaded, its generic of the same name is resolved first by R's namespace search; this fallback only fires for bare-namespace use.

Usage

```
as_draws(x, ...)
```

Arguments

x	a fitted `ame` or `lame` object.
...	passed to the relevant method.

Value

An object dispatched by the relevant method (typically a `posterior::draws_array` for an `ame` / `lame` fit).

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
draws <- as_draws(fit)
```

`as_draws.ame` *Convert an AME / LAME fit to a posterior draws object*

Description

Reshapes the BETA + VC posterior draws stored on an `ame` or `lame` fit into a `posterior::draws_array` (or `draws_df`-equivalent) suitable for downstream packages that consume that representation. Multi-chain fits (`ame(..., chains = N)`, two chains by default) are reshaped with chain as a separate dimension; single-chain fits are returned as a single-chain `draws_array`.

Usage

```
## S3 method for class 'ame'
as_draws(x, include = c("beta", "vc"), ...)

## S3 method for class 'lame'
as_draws(x, include = c("beta", "vc"), ...)

## S3 method for class 'ame_als'
as_draws(x, ...)
```

Arguments

`x`	an `ame` or `lame` fit.
`include`	character; which parameter blocks to include. Default `c("beta", "vc")`; can also include `"rho_uv"`, `"rho_ab"` for dynamic LAME fits, and `"rho_path"` (the per-period residual correlation draws) for `dynamic_rho = TRUE` fits.
`...`	ignored.

Details

The reshape uses `fit$chain_indicator` (set when `chains > 1`) to keep chain identity separate so within-chain Rhat / ESS estimates and chain-coloured traceplots resolve correctly.

Value

A `posterior::draws_array` (when **posterior** is available) or otherwise a plain 3-D array with named dimnames.

as_lame_y *Convert a graph object to a lame-ready adjacency matrix*

Description

Convenience wrapper that takes an **igraph** or **network** object and returns the kind of matrix that ame / lame expect: a numeric adjacency matrix with NA on the diagonal (self-ties not modelled) and (if available) actor names preserved on the row/column dimnames.

Usage

```
as_lame_y(x, na_diag = TRUE)
```

Arguments

x	an igraph, network, matrix, or data.frame representing an adjacency / sociomatrix.
na_diag	logical: replace the diagonal with NA? Defaults to TRUE for square (unipartite) matrices and is ignored for rectangular (bipartite) matrices.

Details

Plain matrices and data.frames are accepted too; data.frames are coerced to a numeric matrix and only the diagonal is rewritten to NA.

Value

A numeric matrix suitable to pass as Y to ame / lame.

Examples

```
# plain matrices are returned with the diagonal set to NA
m <- matrix(rbinom(25, 1, 0.4), 5, 5,
            dimnames = list(letters[1:5], letters[1:5]))
as_lame_y(m)

if (requireNamespace("igraph", quietly = TRUE)) {
  g <- igraph::sample_gnp(8, 0.3)
  Y <- as_lame_y(g)
  dim(Y)
}
```

autoplot.ame_als *autoplot method for ALS fits*

Description

Coefficient point-and-interval plot for an ame_als or lame_als fit. The body builds the same frame autoplot.lame uses, sourcing the point estimate from coef(fit) and the interval from confint(fit) (sandwich or bootstrap, depending on which is available). which = "beta" is the only mode supported; which = "uv" / "ab" return an informative error.

Usage

```
## S3 method for class 'ame_als'
autoplot(object, which = c("beta", "uv", "ab"), conf.level = 0.95, ...)

## S3 method for class 'lame_als'
autoplot(object, which = c("beta", "uv", "ab"), conf.level = 0.95, ...)
```

Arguments

object	A fitted ame_als / lame_als object.
which	One of "beta" (currently the only supported value).
conf.level	Confidence level for the interval. Default 0.95.
...	Passed to vcov.ame_als / confint.ame_als (e.g. cluster).

Value

A ggplot object.

autoplot.lame *Ribbon plot of time-varying coefficients (or coefplot for static fits)*

Description

For a lame fit with dynamic_beta on, returns a faceted ggplot of the posterior mean coefficient path per period with a 95 percent credible-interval ribbon. For a static fit (no dynamic_beta), falls back to a tidy()-driven horizontal coefplot with posterior-mean point estimate and credible-interval bars so that autoplot(fit) returns a ggplot regardless of fit type.

Usage

```
## S3 method for class 'lame'
autoplot(
  object,
  which = c("beta", "ab", "uv"),
  probs = c(0.025, 0.5, 0.975),
  coefs = NULL,
  ...
)

## S3 method for class 'ame'
autoplot(
  object,
  which = c("beta", "ab", "uv"),
  probs = c(0.025, 0.5, 0.975),
  coefs = NULL,
  ...
)
```

Arguments

object	A fitted ame / lame object.
which	One of "beta" (default; coefficient plot – ribbon when dynamic, coefplot when static), "ab" (sender / receiver effects when dynamic_ab), "uv" (latent positions when dynamic_uv).
probs	Length-3 vector of quantiles to plot. Default c(0.025, 0.5, 0.975) for 95 percent intervals.
coefs	Optional character vector of coefficient names to subset.
...	Ignored.

Value

A ggplot2 object that can be further customised.

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X, family = "binary", R = 0,
            dynamic_beta = "dyad",
            nscan = 60, burn = 15, odens = 5, verbose = FALSE, chains = 1)
if (requireNamespace("ggplot2", quietly = TRUE)) {
  autoplot(fit)
}
```

boot_ame-no-fitted *fitted/residuals are not defined for a bootstrap object*

Description

A boot_ame stores replicate-level distributional summaries, not per-cell predictions or residuals. These methods exist only to refuse the call clearly (instead of falling through to stats::fitted.default / stats::residuals.default, which would silently return NULL). Apply fitted / residuals to the underlying ame_als fit instead.

Usage

```
## S3 method for class 'boot_ame'
fitted(object, ...)

## S3 method for class 'boot_ame'
residuals(object, ...)
```

Arguments

object	a boot_ame object.
...	ignored.

Value

Never returns; raises an error.

check_format *Validate input data format for lame function*

Description

Internal validation function that checks the format and consistency of input data for longitudinal AME models. Ensures that network data and covariates are properly formatted as lists with consistent dimensions across time periods.

Usage

```
check_format(Y, Xdyad=NULL, Xrow=NULL, Xcol=NULL)
```

Arguments

Y	a list of T network matrices, where T is the number of time periods. Each element should be an n x m matrix representing the network at time t.
Xdyad	an optional list of T dyadic covariates. Each element can be either an n x m matrix (single covariate) or an n x m x p array (p covariates). Must have the same length as Y if provided.
Xrow	an optional list of T matrices of row/sender covariates. Each element should be an n x pr matrix where pr is the number of row covariates. Must have the same length as Y if provided.
Xcol	an optional list of T matrices of column/receiver covariates. Each element should be an m x pc matrix where pc is the number of column covariates. Must have the same length as Y if provided.

Details

Validates input data for longitudinal network analysis:

- Verifying Y is a non-empty list of matrices
- Checking that all covariates (if provided) are lists of appropriate length
- Validating data types for all elements
- Warning about dimension inconsistencies across time periods

The function uses informative error messages via the cli package to help users identify and correct data formatting issues.

Value

Invisible TRUE if all checks pass. Throws an error with an informative message if any validation fails.

Note

This is an internal function primarily used by `lame()` but exported for advanced users who want to validate their data before model fitting.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
n <- 12
Y <- lapply(1:3, function(t) matrix(rnorm(n * n), n, n))
check_format(Y)
```

coef.als_dynamic_beta *Extract beta path from a penalised-ALS object*

Description

Extract beta path from a penalised-ALS object

Usage

```
## S3 method for class 'als_dynamic_beta'
coef(object, ...)
```

Arguments

object	An als_dynamic_beta object.
...	Ignored.

Value

The $p \times T$ matrix of estimates.

coef.ame *Extract model coefficients from AME model*

Description

Returns posterior means of regression coefficients from a fitted AME or LAME model.

Usage

```
## S3 method for class 'ame'
coef(object, ...)

## S3 method for class 'lame'
coef(object, ...)
```

Arguments

object	Fitted AME model (class "ame" or "lame").
...	Additional arguments (ignored).

Details

For a static fit (the default, and any model with `dynamic_beta = FALSE`), coefficients are returned as a named numeric vector computed as `colMeans(fit$BETA)`.

For a dynamic fit (`lame(..., dynamic_beta = ...)` where some coefficient is time-varying), `fit$BETA` is a 3-dimensional array `[n_stored, p, T]` and `coef.lame` returns a `[p, T]` matrix of per-period posterior means. Rownames are the coefficient names; colnames are the period labels (from `names(Y)` or `t1, t2, ...`). Static coefficients in a dynamic fit are constant across the columns.

For binary models, these are on the probit (latent) scale. Use `predict.ame` with `type = "response"` to get predicted probabilities.

What `coef()` does not return. The multiplicative latent positions U, V are not part of the coefficient vector; they live on `fit$U` and `fit$V` (or as 3-D arrays `[n, R, T]` when `dynamic_uv` is on). The additive sender / receiver effects a, b are on `fit$APM` and `fit$BPM`. For a tidy frame of latent positions use `latent_positions`; for sender / receiver lollipops use `ab_plot`.

Value

Named numeric vector (static fit) or `p x T` matrix (`dynamic_beta` fit) of posterior mean coefficients.

See Also

`vcov.ame` for the posterior covariance matrix, `confint.ame` for credible intervals, `summary.ame` for a full summary table

---

coef.ame_als *Extract coefficients from a fast AME fit*

Description

Returns the point-estimated regression coefficients (intercept first) of an `ame_als` fit. There is no posterior here; for uncertainty use `ame_als_bootstrap`.

Usage

```
## S3 method for class 'ame_als'
coef(object, ...)
```

Arguments

`object`	an `ame_als` object.
`...`	ignored.

Value

A named numeric vector of coefficients.

coef.boot_ame *Point estimates from a fast AME bootstrap*

Description

Returns the intercept and regression coefficient point estimates carried by a boot_ame object (the estimates the bootstrap quantifies). For their bootstrap standard errors and intervals see confint.boot_ame and vcov.boot_ame.

Usage

```
## S3 method for class 'boot_ame'
coef(object, ...)
```

Arguments

object	a boot_ame object.
...	ignored.

Value

A named numeric vector of coefficient point estimates.

coldwar *Cold War data*

Description

Positive and negative relations between countries during the cold war

Format

A list including the following dyadic and nodal variables:

- cc: a socioarray of ordinal levels of military cooperation (positive) and conflict (negative), every 5 years;
- distance: between-country distance (in thousands of kilometers);
- gdp: country gdp in dollars every 5 years;
- polity: country polity every 5 years.

Source

Xun Cao, Department of Political Science, Pennsylvania State University.

`combine_ame_chains` *Combine multiple AME chains*

Description

Combine multiple AME chains

Usage

```
combine_ame_chains(chain_list, diagnostics = TRUE)
```

Arguments

`chain_list`	List of ame fit objects from multiple chains
`diagnostics`	Logical; whether to compute convergence diagnostics (default TRUE)

Value

A single ame object with combined chains. Conditional sampler health counts are summed when every chain supplies them. The original conditional counts and joint-move diagnostics are retained in `sampler_diagnostics_by_chain`; move diagnostics from a single chain are not presented as pooled values. If a legacy chain lacks the numerical/bound decomposition, the combined `sampler_health` is `NULL`; its available per-chain records remain accessible.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# each fit is one chain (chains = 1); combine_ame_chains() pools them
fit1 <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, seed = 1, verbose = FALSE, chains = 1)
fit2 <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, seed = 2, verbose = FALSE, chains = 1)
pooled <- combine_ame_chains(list(fit1, fit2))
```

compact_ame *Optimize AME model output for memory efficiency*

Description

Optimizes AME model storage by converting matrices to sparse format.

Usage

```
compact_ame(fit, use_sparse_matrices = FALSE)
```

Arguments

fit	Fitted AME model
use_sparse_matrices	Logical; whether to use sparse matrix storage (default: FALSE)

Details

When use_sparse_matrices = TRUE, the posterior-mean matrices YPM and EZ are converted to sparse storage via the **Matrix** package, but only when a matrix's nonzero density is below 0.5 – converting a dense posterior-mean matrix would *grow* memory, so denser matrices are left as ordinary dense matrices. For lame fits, where YPM/EZ are per-period lists, the check and conversion are applied per element. The additive-effect summaries APM/BPM are named numeric vectors and are never converted. Names and dimnames are preserved.

Value

Memory-optimized AME model object

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
fit_small <- compact_ame(fit)
```

compute_mcmc_diagnostics *Compute MCMC convergence diagnostics for multiple chains*

Description

Computes R-hat and effective sample size (ESS) across two or more independently-seeded chains, for both the regression coefficients (BETA) and the variance components (VC). ESS is computed on the pooled draws, so genuinely non-converged chains report a low ESS rather than an inflated one.

Usage

```
compute_mcmc_diagnostics(chain_list)
```

Arguments

chain_list	A list of independently seeded fits from the same model: either ame/lame objects carrying BETA, or lame_robust objects. Do not mix fit classes.

Details

rhat is the split rank-normalised R-hat of Vehtari et al. (2021), the larger of the bulk and tail values, as posterior::rhat() reports it. It is the statistic of the between-chain gate of ame and lame, computed on the first min(n_samples) draws of every chain, so for a two-chain fit it equals fit$rhat for every monitored coefficient and variance component, and it is the Rhat column of summary(). It is NA for a quantity that is constant in every chain and for chains with fewer than 8 draws. rhat_classic is the Gelman-Rubin statistic of the whole chains, without splitting or rank normalisation. The printed verdict compares rhat with the gate's threshold, 1.05.

For lame_robust fits, the **posterior** package is required. Diagnostics include static coefficients, time intercepts, actor paths and levels, estimated hyperparameters, every off-diagonal latent product, and persistent pair precisions. Unused rank-zero factor hyperparameters and fixed Gaussian precisions are excluded. Chains must have identical inputs and priors, different seeds, and equal lengths of at least four retained draws. Robust pass requires both rhat and rhat_classic below 1.05 and finite positive ESS, bulk ESS and tail ESS for every included parameter. The individual R-hat and effective sample size values are reported for every included parameter. The existing ame/lame diagnostic scope remains coefficients and variance components.

Value

A list with rhat, rhat_classic, ess, param_names (over BETA and VC), n_chains and n_samples. For robust fits, additionally returns ess_bulk, ess_tail, a full diagnostics data frame, and pass.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# each fit is one chain (chains = 1); compute_mcmc_diagnostics() compares them
fit1 <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, seed = 1, verbose = FALSE, chains = 1)
fit2 <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, seed = 2, verbose = FALSE, chains = 1)
diag_out <- compute_mcmc_diagnostics(list(fit1, fit2))
diag_out$rhat
```

---

comtrade *Comtrade data*

Description

Ten years (1996 to 2005) of bilateral trade among 30 countries in six SITC Rev. 1 commodity classes aggregated at the first level (AG1).

Format

A four-way array of dimension 30 x 30 x 6 x 10 (exporter, importer, commodity class, year) with dimnames giving the country names, the classes (chemicals; crude materials, inedible, except fuels; food and live animals; machinery and transport equipment; manufactured goods classified chiefly by material; miscellaneous manufactured articles) and the years. Each value is the change from the previous year in the natural logarithm of exports measured in 2000 US dollars. For example, the 1996 slice records changes from 1995 to 1996. Values are growth measures, not log levels or raw trade volumes; negative values indicate decreases. The 1,800 missing cells are the within-country diagonals. The first index is the reporting (exporting) country.

Source

Hoff, P. D. (2011), Section 4 of *Separable covariance arrays via the Tucker product, with applications to multivariate relational data*, doi:10.1214/11BA606. Original processed array: https://www2.stat.duke.edu/~pdh10/Code/hoff_2011_ba/. Underlying trade and inflation data: https://comtrade.un.org/, https://www.measuringworth.com/

---

confint.ame *Bayesian credible intervals for AME model parameters*

Description

Returns posterior **equal-tailed quantile-based** credible intervals (not highest-posterior-density, HPD). Built directly from `quantile(object$BETA, c(alpha/2, 1-alpha/2))` and `quantile(object$VC, ...)`. These are Bayesian credible intervals, not frequentist confidence intervals.

Usage

```
## S3 method for class 'ame'
confint(object, parm = NULL, level = 0.95, ...)

## S3 method for class 'lame'
confint(object, parm = NULL, level = 0.95, ...)
```

Arguments

object	fitted AME / LAME model.
parm	character vector of parameter names, or numeric indices. When parm is character, both BETA names (e.g. "intercept", "x1_dyad") and variance-component names ("va", "vb", "cab", "rho", "ve") are accepted; on a unipartite binary, cbin or frn panel carrying fewer than one event per actor-period the "va", "vb" and "cab" intervals are NA, because the panel carries too few events per actor-period to estimate those variances (the draws stay in object$VC), unless the effects prior was supplied, when they are the intervals under that prior; under a saturated nomination cap the "intercept" and row-covariate intervals are NA with or without a supplied prior. NULL (default) returns intervals for the regression coefficients only, the parameters coef() reports; name variance components in parm to include them. Numeric indices select coefficient rows.
level	credible level (default 0.95).
...	additional arguments (ignored).

Value

Matrix with one row per parameter and two columns (e.g. "2.5%", "97.5%").

Note on interval type

These are equal-tailed quantile intervals, not HPD. For an HPD interval use e.g. coda::HPDinterval on the columns of object$BETA and object$VC directly.

---

confint.ame_als *Confidence intervals for a fast AME fit*

Description

Wald confidence intervals for the intercept and dyadic-covariate coefficients of an ame_als fit, built from the conditional sandwich covariance (vcov.ame_als); for an empirical-Bayes fit (the default for normal, binary and Poisson responses), for every coefficient from the Laplace covariance.

Usage

```
## S3 method for class 'ame_als'
confint(object, parm = NULL, level = 0.95, ...)
```

Arguments

object	an ame_als fit.
parm	character vector of parameter names, or integer indices; if NULL (default) all available coefficients are returned.
level	confidence level (default 0.95).
...	passed to vcov.ame_als (e.g. cluster).

Details

An empirical-Bayes fit (the default for normal, binary and Poisson fits) has Wald intervals from its Laplace covariance, which propagates the uncertainty of the actor effects and the latent factors and covers the node-covariate coefficients too.

The sandwich intervals of the other fits are a fast convenience. They are **conditional** (the additive and multiplicative effects are held fixed) and therefore anti-conservative, and they cover only the regression coefficients the sandwich covariance is defined for – not the node-covariate, additive or multiplicative parameters. For fully-propagated intervals on all parameters, use ame_als_bootstrap and confint.boot_ame.

Value

A matrix with one row per coefficient and lower/upper bound columns.

See Also

ame_als_bootstrap for bootstrap intervals on all parameters.

---

confint.boot_ame *Confidence intervals from a fast AME bootstrap*

Description

Bootstrap confidence intervals for the intercept and regression coefficients of a fast AME fit.

Usage

```
## S3 method for class 'boot_ame'
confint(
  object,
  parm = NULL,
  level = 0.95,
  ci_type = c("percentile", "basic"),
  which = c("all", "beta", "vc", "a", "b", "U", "V"),
  ...
)
```

Arguments

object	a boot_ame object.
parm	character vector of parameter names, or integer indices. If NULL (default), all coefficients are returned.
level	confidence level (default 0.95).
ci_type	interval type: "percentile" (default) uses the replicate quantiles directly (for a block bootstrap, those at the expanded percentile levels of ame_als_bootstrap) and matches the intervals reported by summary() and tidy() on the same fit; "basic" reflects the replicate quantiles about the point estimate ($2\hat{\theta} - q$), which corrects first-order bias and can be preferable for the IRLS binary/Poisson point estimators on small samples.
which	which uncertainty channel to return: "all" (default, intercept + regression coefficients + variance components + sender effects • receiver effects + multiplicative U/V), or a single channel: "beta", "vc", "a", "b", "U", "V".
...	ignored.

Value

A matrix with one row per parameter and lower/upper bound columns. The rows of the U and V columns whose singular values are not separated from a neighbour's, of a variance component with no interval (vc_uncentred), and of a parameter the bootstrap returns no interval for (withheld), are NA; see ame_als_bootstrap.

---

detect_change_point *Detect potential change points in a dynamic_beta posterior path*

Description

For each selected dynamic coefficient, compares the largest absolute first difference divided by its fitted innovation standard deviation $\sigma_\beta \sqrt{\Lambda_{jj}}$ with a simulated reference. Each reference path has the same number of periods as the fitted path, starts from the first state of the matched posterior draw, and follows that draw's persistence and innovation scale: it is the largest jump a path that begins where the fitted path begins makes by chance. The design-based innovation scale and the averaging of symmetric node coefficients are retained. This is a posterior-conditioned transition reference, not an unconditional prior predictive check or a marginal-likelihood Bayes factor.

Usage

```
detect_change_point(
  fit,
  coefs = NULL,
  n_prior_sims = 2000L,
  threshold_bf = 10,
  seed = NULL
)
```

Arguments

`fit`	A fitted `lame` object with `dynamic_beta`.
`coefs`	Optional character vector of coefficient names to check. Defaults to every dynamic coefficient on the fit.
`n_prior_sims`	Number of simulations for the transition reference. Default `2000`.
`threshold_bf`	Cutoff for the "warn" column. Default `10`, the the threshold at which the measured false-positive rate is below 5 percent for smooth and for in-model paths (see Measured properties).
`seed`	Optional seed for reproducibility.

Details

The `bf` column is the tail ratio $\Pr(M^{post} > m^*)/0.05$, where m^* is the reference's 95th percentile. Its range is zero to twenty, in steps of 20 over the number of stored draws. Values above `threshold_bf` flag unusually abrupt posterior paths for inspection; they do not establish a change point. The reference supports `"ar1"` and `"rw1"` on equally spaced periods. RW2, Matern, irregular periods and actor-specific coefficient laws are refused rather than assigned an AR(1) reference. Saved named training designs, the dynamic mask, and aligned transition draws are required.

Value

A data frame with one row per dynamic coefficient: `coef`, `bf` (the heuristic tail-ratio score), `m_post_mean` (posterior mean of M), `m_prior_q95` (95\ M), `t_hat` (period of the largest scaled jump), `warn` (logical, `bf` > `threshold_bf`).

Measured properties

The table gives the percentage of simulated panels with `bf` > `threshold_bf` at the default threshold, from 200 fits of each path type and panel shape. Every panel has one standard-normal dyadic covariate whose coefficient follows the named path, is fitted with `dynamic_beta = "dyad"` (`dynamic_beta_kind = "rw1"` for random-walk paths, `"ar1"` otherwise) and keeps 40 stored draws (12 for the binary panels, which have 50 actors; the others have 15). The outcome is normal, binary or Poisson. *Smooth* paths are a constant, a linear ramp and two stationary AR(1) paths (persistence 0.7, innovation standard deviation 0.15 and 0.4); *in-model* paths are random walks and AR(1) paths drawn from the prior of the fit; a *step* is a jump of 0.6 or 1.2 (from 0.2 to 0.8 or to 1.4) at the middle of the panel. Normal outcomes have residual standard deviation 0.4; binary outcomes are probit with intercept -0.3; Poisson outcomes have log-rate offset -0.5 and no AR(1) paths drawn from the prior. *Worst path type* is the highest rate among the smooth and in-model path types of the row.

T	outcome	smooth	in-model	worst path type	step 0.6	step 1.2
4	binary	1.2	2.5	5.0	2.5	20.5
5	normal	1.2	3.2	4.0	4.0	20.0
5	Poisson	0.8	0.0	3.0	2.5	20.5
8	normal	1.1	1.8	3.0	56.0	85.0
12	normal	1.9	3.0	5.5	100.0	100.0

Pooled over panel shapes, smooth paths were flagged in 1.2 percent of the fits and in-model paths in 2.3 percent. At a threshold of 9 the pooled in-model rate is 5.0 percent, with no margin, and at 10 it is 2.3 percent, so 10 is the default. The tail ratio judges a jump against the innovation scale the fit estimates from the same path, so a panel of four or five periods rarely shows a single jump as unusual (about a fifth of the panels with a step of 1.2 are flagged), while panels of eight or more periods flag it in most cases.

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X, family = "binary", R = 0,
            dynamic_beta = "dyad",
            nscan = 60, burn = 15, odens = 5, verbose = FALSE, chains = 1)
detect_change_point(fit)
```

`dutchcollege` *Dutch college data*

Description

Longitudinal relational measurements and nodal characteristics of Dutch college students, described in van de Bunt, van Duijn, and Snijders (1999). The time interval between the first four measurements was three weeks, whereas the interval between the last three was six weeks.

Format

A list consisting of a socioarray `Y` and a matrix `X` of static nodal attributes. The relational measurements range from -1 to 4, indicating the following:

- -1 a troubled or negative relationship
- 0 don't know
- 1 neutral relationship
- 2 friendly
- 3 friendship
- 4 best friends

Source

Originally available at `http://moreno.ss.uci.edu/data.html#vdb` (site no longer accessible).

`dynamic_beta_prior_summary` *Summarise the implied prior on a time-varying coefficient path*

Description

Draws `ndraws` sample paths of length `n_periods` from the `dynamic_beta` prior (given a kind, AR(1) hyperparameters, and an inverse-gamma on the innovation variance) and reports the implied distribution of common summary statistics: maximum absolute first difference, roughness (sum of squared first differences), path range, and correlation with time. Use this before fitting to confirm that your prior is loose / tight enough.

Usage

```
dynamic_beta_prior_summary(
  n_periods = 10L,
  ndraws = 5000L,
  kind = c("ar1", "rw1", "rw2", "matern32"),
  rho_mean = 0.8,
  rho_sd = 0.15,
  rho_lower = 0,
  rho_upper = 0.999,
  sigma_shape = 2,
  sigma_scale = 1,
  matern32_length_scale = 2,
  threshold = 1,
  seed = NULL
)
```

Arguments

`n_periods`	Length of the path (default 10).
`ndraws`	Number of paths to draw (default 5000).
`kind`	One of `"ar1"`, `"rw1"`, `"rw2"`, `"matern32"`. Default `"ar1"`. For `"matern32"` the path is drawn from a continuous-time Matérn-3/2 GP with length scale `matern32_length_scale` and marginal variance σ_β^2.
`rho_mean`, `rho_sd`	Prior mean/SD on ρ (AR(1) only). Used to parameterise a Beta prior on the standardised ρ.
`rho_lower`, `rho_upper`	Truncation bounds (AR(1) only). Default `[0, 0.999]`.
`sigma_shape`, `sigma_scale`	Inverse-Gamma shape / scale on σ_β^2. Defaults `shape = 2, scale = 1`.
`matern32_length_scale`	Length scale for the Matérn-3/2 covariance (`kind = "matern32"` only). Default 2.

threshold	Cutoff for the prob_max_diff_gt_threshold summary. Default 1.
seed	Optional integer seed for reproducibility.

Details

For kind = "ar1", rho_mean/rho_sd are translated to a Beta prior on the standardised $(\rho - \rho_{lower})/(\rho_{upper} - \rho_{lower})$. For kind = "rw1", $\rho = 1$ is fixed and only σ_β^2 is sampled. For kind = "rw2", the second-difference variance is sampled and the first two values use a diffuse $N(0, 10)$ initial prior.

Value

A list with components:

summary Data frame with quantile rows for max-first-diff, roughness, path range, trend correlation.

prob_max_diff_gt_threshold Empirical probability that the maximum absolute first difference exceeds threshold.

rho Length ndraws vector of sampled rho values (NA for kinds with fixed rho).

sigma Length ndraws vector of sampled sigma values.

paths ndraws x n_periods matrix of sample paths.

Examples

```
# Default prior: AR(1) with rho_mean = 0.8, rho_sd = 0.15
s <- dynamic_beta_prior_summary(n_periods = 10, ndraws = 2000, seed = 1)
s$summary
# what's the probability that consecutive beta_t differ by more than 1?
s$prob_max_diff_gt_threshold

# Tighter prior on innovation variance
s_tight <- dynamic_beta_prior_summary(sigma_scale = 0.1, seed = 1)
s_tight$summary
```

---

el2sm *Edgelist to sociomatrix*

Description

Construct a sociomatrix from an edgelist. Each row of el gives the two endpoints of an edge and, optionally, a weight in a third column (unweighted edges default to 1).

Usage

```
el2sm(el,directed=TRUE,nadiag=all(el[,1]!=el[,2]))
```

Arguments

el	a matrix in which each row contains the indices of an edge and possibly the weight for the edge
directed	if FALSE, then a relation is placed in both entry ij and ji of the sociomatrix, for each edge ij (or ji)
nadiag	put NAs on the diagonal

Value

a sociomatrix

Author(s)

lame authors

Examples

```
Y<-matrix(rpois(10*10,.5),10,10) ; diag(Y)<-NA
E<-sm2el(Y)
el2sm(E) - Y
```

---

evaluate_heldout *Held-out predictive evaluation for an ame / lame fit*

Description

Computes family-appropriate held-out predictive scores given a fit and a logical mask of cells to score. The function does not split the data for you: it expects you to have either (a) refit on a training subset and now want to score the held-out cells of the same matrix, or (b) have predicted probabilities you want scored. Both AUROC and PR-AUC are reported when applicable, alongside Brier and mean log density.

Usage

```
evaluate_heldout(y_obs, y_pred, mask, family = "binary", fit = NULL)
```

Arguments

y_obs	Observed outcomes. For longitudinal fits, a list of per-period matrices; for cross-sectional, a single matrix. Cells not in mask are ignored.
y_pred	Predicted probabilities / means on the response scale. Same shape as y_obs.
mask	Logical mask of the same shape as y_obs marking cells to score (TRUE = include in evaluation).
family	Family string; used to pick the scoring rule. Defaults to "binary".

`fit` For `family = "normal"`: the fit that produced `y_pred` (an `ame`/`lame` MCMC fit or an `ame_als`/`lame_als` fit of the normal family). Its residual variance sets the predictive density of `mean_logdens`; without it `mean_logdens` is `NA`. Ignored for the other families.

Details

Workflow. The typical pattern is: mask a random sample of dyads to `NA` in `Y`, refit (`lame()` handles `NA` internally via data augmentation), call `predict(fit, type = "response")`, then pass that prediction alongside the original `Y` and the held-out mask to this function. See the examples.

Dependencies. AUROC / PR-AUC use **precrec** when available; if not installed, only Brier and mean log-density are computed and a one-line note is emitted.

Value

A one-row data frame with columns appropriate to the family: `n_eval`, plus `auroc` + `auprc` + `brier` + `logloss` (binary / cbin), or `rmse` + `mae` + `mean_logdens` (normal), or `mean_logdens` + `rmse` (poisson).

For the normal family `mean_logdens` is the mean over the scored cells of $\log \phi(y; \hat{y}, v_e)$, the normal log density at the prediction with the fit's residual variance v_e: the posterior mean of `VC[, "ve"]` for an MCMC fit, the fitted `VC["ve"]` for a fast fit. It is `NA` when `fit` is not supplied.

Examples

```
set.seed(1)
n <- 25; Y <- matrix(rbinom(n*n, 1, 0.3), n, n); diag(Y) <- NA
rownames(Y) <- colnames(Y) <- paste0("a", sprintf("%02d", 1:n))
# mask 20% of dyads
mask <- matrix(FALSE, n, n)
obs_idx <- which(!is.na(Y))
set.seed(1)
mask[sample(obs_idx, floor(0.2 * length(obs_idx)))] <- TRUE
Y_train <- Y; Y_train[mask] <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y_train, R = 0, family = "binary",
           burn = 15, nscan = 60, odens = 5, verbose = FALSE, plot = FALSE, chains = 1)
y_pred <- predict(fit, type = "response")
evaluate_heldout(Y, y_pred, mask, family = "binary")
```

`fitted.ame` *Extract fitted values from AME model*

Description

Returns the stored response-scale summary (YPM). For binary models, these are predicted probabilities between 0 and 1. For normal models, these are predicted continuous values. For MCMC Poisson models, these are mean counts, $\exp(\eta + \sigma^2/2)$, at the posterior mean η of the linear predictor and the posterior mean σ^2 of the residual variance, the form the fast estimators report. Predictive quantiles and event probabilities come from `predict(type = "distribution")`.

Usage

```
## S3 method for class 'ame'
fitted(object, ...)
```

Arguments

object	Fitted AME model object (class "ame").
...	Additional arguments (not used).

Value

An n x n matrix (unipartite) or nA x nB matrix (bipartite) of fitted values on the response scale. Diagonal entries are NA for unipartite networks.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

predict.ame for predictions with type control, residuals.ame for residuals

---

fitted.ame_als *Extract fitted values from a fast AME fit*

Description

Returns response-scale fitted values: a single matrix for a cross-sectional (ame_als) fit, or a list of matrices for a longitudinal (lame_als) fit.

Usage

```
## S3 method for class 'ame_als'
fitted(object, plug_in = NULL, ...)
```

Arguments

object	an ame_als object.
plug_in	logical; for an empirical-Bayes binary or count fit, return the plug-in fitted values (TRUE) or the predictive means that integrate the link over the uncertainty of the predictor (FALSE). The default is TRUE for a Poisson fit and FALSE for a binary fit.
...	ignored.

Details

For a binary empirical-Bayes fit (the default) the fitted values are predictive probabilities: the expectation of the response over the posterior of the linear predictor, $\Phi(m/\sqrt{1+v})$ for a probit link, with m the posterior mean and v the posterior variance of the predictor (see `predict.ame_als`). For a Poisson fit they are the plug-in mean counts, $e^{\eta+v_e/2}$ at the fitted predictor η (the posterior mean) and the variance v_e of the noise of the count, as the samplers report them and as the dynamic empirical-Bayes fit does: the posterior predictive mean of a count is not finite in this model (the inverse-gamma tail of v_e, and the inverse-Wishart tail of the scale of the factors for a cell the data do not inform), so its value would be a property of the numerical integration and of the prior; the plug-in mean is finite everywhere. `plug_in = FALSE` returns the predictive mean of a count that the integration over the predictor gives, $e^{m+(v+v_e)/2}$, and `plug_in = TRUE` the plug-in values of a binary fit (the response mean at the fitted predictor); the plug-in values are what `fit$fitted` and `fit$YPM` hold and `simulate()` uses. `residuals()` follows `fitted()`. The mean of a normal response is its predictor, so a normal fit's fitted values are the posterior means of the predictor, with or without `plug_in`.

A dynamic fit (`lame(method = "als")` with `dynamic_ab`, `dynamic_beta` or `dynamic_uv`) follows the same rules. The empirical-Bayes binary fit returns the predictive probability, with v the variance of the cell's predictor: the additive effects given the coefficients (the prior variance for an actor with no observed cell), the product of the positions, and the coefficients' part at the design asked for; `plug_in = TRUE` returns $\Phi(m)$, which `fit$fitted` keeps. A fit stored without the variance record is an error of class `"lame_predictive_missing"`. The empirical-Bayes count fit returns the plug-in mean $e^{\eta+v_e/2}$, the normal fit the predictor, and a penalised fit (`bias_correct = "none"`) the response mean at its predictor ($\Phi(\eta)$, e^{η}). The dynamic route has no exposure and no forecasts (`predict(h > 0)` is an error).

Value

A matrix or list of matrices of fitted values.

`fitted.lame` *Extract fitted values from LAME model*

Description

Returns the stored response-scale summaries. For MCMC Poisson models these are mean counts, $e_t \exp(\eta + \sigma^2/2)$, at the posterior mean η of the linear predictor and the posterior mean σ^2 of the residual variance, with e_t the exposure of the period (`period_exposure`; one without); the form the fast estimators report. Predictive quantiles and event probabilities come from `simulate()` and from `predict(fit, h = 1, interval = "prediction")`.

Usage

```
## S3 method for class 'lame'
fitted(object, ...)
```

Arguments

object	Fitted LAME model
...	Additional arguments

Value

List of fitted value matrices (one per time point)

---

forecast_pit *Probability-integral-transform calibration check for h-step forecasts*

Description

For a fit with at least one dynamic component, evaluates how well the h-step posterior-predictive distribution covers the actually observed outcomes at the corresponding period(s). A well-calibrated forecast produces approximately Uniform(0, 1) PIT values.

Usage

```
forecast_pit(
  fit,
  y_future,
  h = NULL,
  n_draws = NULL,
  newdata = NULL,
  newdesign = NULL,
  newexposure = NULL
)
```

Arguments

fit	A fitted lame or lame_robust object trained on the first T - h periods (the function does not refit; the user is responsible for the train/forecast split).
y_future	A list of length h of held-out observed matrices, one per forecast period. The same shape as the fitted actor grid. Named periods are aligned to shared actors; unshared actors and missing outcomes are excluded. Unipartite self cells are excluded and symmetric networks score each unordered dyad once. Bipartite grids retain cells whose row and column positions coincide.
h	The forecast horizon (length of y_future). Inferred from length(y_future) when not supplied.
n_draws	Number of posterior draws to use for the forecast. Default uses all stored draws.
newdata	Future dyadic covariate arrays passed to the forecaster. Required for robust fits that contain dyadic covariates, as in predict.lame_robust().
newdesign	Full prepared future designs, including nodal roles. Supported only by ordinary lame fits.
newexposure	Future Poisson exposures, one per horizon. Defaults to the last fitted exposure. Zero exposure defines a degenerate zero-count law.

Details

Workflow: fit on periods `1:(T - h)`, forecast `h` steps ahead, score against the held-out `T - h + 1 : T` observations. Returns a numeric vector of PIT values (one per observed dyad in the held-out periods), plus a one-row summary with the Kolmogorov-Smirnov statistic against Uniform(0, 1) and a fraction-of-observed-dyads coverage diagnostic.

Families. For `family = "normal"`, the PIT averages the Gaussian CDF over matched predictor and residual-variance draws. For discrete families (`"binary"`, `"poisson"`, `"ordinal"`) the randomised PIT of Czado, Gneiting & Held (2009) is used. Poisson forecasts integrate the Gaussian observation effect and future exposure. Ordinal scoring requires saved explicit posterior cutpoints; data-induced joint rank simulation does not define a fixed future threshold law. For rank families (`"cbin"`, `"frn"`) PIT is not implemented; the function returns NULL with an informational note. Robust fits from `lame_robust()` use their genuine future latent states, matched persistent pair precisions and Gaussian or transient Student residual CDF. Fitted temporary observation precisions are never reused for future outcomes.

Plotting. The companion `plot()` method on the returned object renders a histogram with a uniform reference line. Use `ggplot2::ggplot(pit$pit) + geom_histogram()` for custom plots.

Value

A list of class `forecast_pit` with components `pit` (numeric vector), `ks_stat` (KS statistic against Uniform), `ks_p` (`NA`; a dependent-network null reference is unavailable), `cover_95` (fraction of PIT values in `[0.025, 0.975]`), matched `draw_index`, predictive `cdf_lower`/`cdf_upper`, and scored actor/period `cells`. Network cells need not be independent. The KS statistic, histogram and coverage are descriptive. No global model-adequacy p-value is reported without a calibrated joint network reference. `ks_inference` records this restriction.

References

Czado, C., Gneiting, T., & Held, L. (2009). Predictive model assessment for count data. *Biometrics*, 65(4), 1254-1261.

Examples

```
data(YX_bin_list)
Y_train <- lapply(YX_bin_list$Y, function(y) 1 * (y > 0))[1:3]
Y_test <- lapply(YX_bin_list$Y, function(y) 1 * (y > 0))[4]
X_train <- YX_bin_list$X[1:3]
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(Y_train, Xdyad = X_train,
            family = "binary", R = 0,
            dynamic_beta = "dyad",
            nscan = 60, burn = 15, odens = 5, verbose = FALSE, chains = 1)
pit <- forecast_pit(fit, y_future = Y_test)
pit$ks_stat
plot(pit)
```

formula.ame *formula() is not defined for an ame() / lame() fit*

Description

ame() and lame() use named arguments (Xrow, Xcol, Xdyad, symmetric, etc.) rather than a formula interface. The summary() output renders a pseudo-formula for diagnostic display, but it is not a valid R formula and cannot be passed back to a fitting function.

Usage

```
## S3 method for class 'ame'
formula(x, ...)

## S3 method for class 'lame'
formula(x, ...)
```

Arguments

x	an ame or lame fit.
...	ignored.

Value

Never returns; raises an error.

glance *S3 generic for* glance

Description

Light-weight fallback so glance(fit) dispatches through S3 even when **broom** or **generics** is not loaded.

Usage

```
glance(x, ...)
```

Arguments

x	An object to glance at.
...	Passed to the method.

Value

A one-row data frame.

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
glance(fit)
```

---

glance.ame *Glance method for fitted* ame / lame *objects*

Description

One-row data frame summarising model-level statistics, in the **broom** idiom. Used by `modelsummary::modelsummary()` and similar tabling tools to populate the lower goodness-of-fit panel of a regression table.

Usage

```
## S3 method for class 'ame'
glance(x, ...)

## S3 method for class 'lame'
glance(x, ...)
```

Arguments

x	A fitted ame / lame object.
...	Ignored.

Value

A one-row data frame with columns:

- nobs – number of observed dyads (NA cells excluded).
- n_actors – number of distinct actors; for bipartite fits this is the row-actor count (a 12 x 9 fit reports 12).
- n_row_actors, n_col_actors – row and column actor counts for bipartite fits (NA for unipartite).
- n_periods – number of time periods (1 for ame).
- n_stored – number of stored MCMC draws.
- family – outcome family, such as "normal" or "binary".
- mode – "unipartite" or "bipartite".
- R – latent-space dimension (or max of R_row, R_col for bipartite).
- dynamic_uv, dynamic_ab, dynamic_beta – logicals; whether each component is time-varying.
- elpd_loo – leave-one-out expected log predictive density. Populated only when a loo object has been cached on the fit: fit with save_log_lik = TRUE, then attach it via fit$loo <- loo(fit). NA otherwise (calling loo(fit) alone does not modify the fit object).

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X, family = "binary", R = 0,
            nscan = 100, burn = 20, odens = 5, verbose = FALSE, chains = 1)
glance(fit)
```

`glance.ame_als` *Glance method for fitted* `ame_als` / `lame_als` *objects*

Description

One-row data frame summarising an ALS fit, compatible with **broom / modelsummary**. Reports observation count, actor count, latent dimension, family, mode, ALS convergence flag, and iteration count.

Usage

```
## S3 method for class 'ame_als'
glance(x, ...)

## S3 method for class 'lame_als'
glance(x, ...)
```

Arguments

`x`	A fitted `ame_als` / `lame_als` object.
`...`	Ignored.

Value

One-row data frame with columns `nobs`, `n_actors`, `n_periods`, `family`, `mode`, `R`, `converged`, `iterations`, `se_source`.

Examples

```
data(YX_nrm)
fit <- ame_als(Y = YX_nrm$Y, Xdyad = YX_nrm$X[, , 2:3],
               family = "normal", R = 1, verbose = FALSE)
glance(fit)
```

gof *Compute GOF statistics from saved posterior samples*

Description

Computes goodness-of-fit statistics after model estimation by generating posterior predictive networks from the saved MCMC samples. This is useful when the model was fitted with `gof = FALSE` to speed up MCMC sampling, or when you want to evaluate custom GOF statistics without rerunning the model.

Usage

```
gof(fit, Y = NULL, custom_gof = NULL, nsim = 100, verbose = TRUE)
```

Arguments

`fit`	An ame model object that was run with posterior sampling enabled
`Y`	Original data. For ame objects, an n x n matrix (or nA x nB for bipartite). For lame objects, a list of matrices (one per time period). If NULL, extracted from the fit object.
`custom_gof`	Optional custom GOF function(s), each accepting one period's matrix. A function can return a named numeric vector; a list contains scalar-valued functions. Observed and replicated statistics are both averaged across the same fitted periods.
`nsim`	Number of posterior predictive simulations to generate (default 100). Posterior rows are sampled with replacement, so this may exceed the number of stored rows. If NULL, uses the number of post-initial rows.
`verbose`	Logical; print progress information

Details

Gaussian bipartite checks use weighted `four.cycle.dep`; a nonzero count would be fixed by the missing-cell pattern. Each replicate is evaluated on the same observed cells as its corresponding data slice, aligning actor labels before applying the mask. Named data periods are matched to the fit; unnamed periods must be in fitted order. All custom functions receive individual matrices for both observed and replicated data, and their values are averaged over periods with equal weight.

For the ordinal family the posterior predictive draws are quantile-matched to the observed category frequencies (the cutpoints are induced from the data), so the simulated marginal distribution always equals the observed one: the GOF statistics and `YPM` cannot show a marginal misfit for ordinal outcomes and only assess the dependence structure (degree spread, dyadic and triadic dependence).

This function requires joint posterior replication and refuses fits that lack aligned saved draws; it never substitutes conditional mean trajectories. This function requires that the model was estimated with posterior sampling of the parameters needed to generate posterior predictive datasets. Specifically, it needs:

- BETA: regression coefficients

- VC: variance components
- For models with random effects: samples of a, b
- For models with latent factors: U_samples, V_samples

To enable posterior sampling during model estimation, use: `posterior_opts = posterior_options(save_UV = TRUE, save_ab = TRUE, thin_UV = 1, thin_ab = 1)`. At rank zero, omit `save_UV`.

Computing GOF post-hoc has several advantages:

- Faster MCMC sampling (no GOF overhead)
- Can experiment with different GOF statistics without re-running model
- Can control number of posterior predictive simulations independently

Value

A `(nsim + 1) x n_stat` matrix of GOF statistics: the first row holds the observed statistics, the remaining rows the posterior predictive ones. For an `ame` fit this is the format of `fit$GOF`; a `lame` fit stores its own goodness of fit as a per-statistic list of period-by-draw matrices instead, so the two are not interchangeable. Attributes `replication_method` and `draw_index` record joint replication and the sampled MCMC rows.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

`ame`, `lame`, `gof_plot`

Examples

```
# Run model without GOF during fitting
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2, gof = FALSE,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE,
           posterior_opts = posterior_options(save_ab = TRUE, save_UV = TRUE,
                                              thin_ab = 1, thin_UV = 1), chains = 1)

# Compute GOF post-hoc
gof_result <- gof(fit)
```

gof_plot *Visualize goodness-of-fit statistics for AME and LAME models*

Description

Plots observed network statistics against their posterior predictive distributions to assess model fit.

Usage

```
gof_plot(
  fit,
  type = c("auto", "static", "longitudinal"),
  statistics = NULL,
  credible.level = 0.95,
  ncol = 2,
  point.size = 2,
  line.size = 1,
  title = NULL,
  ...
)
```

Arguments

fit	An object of class "ame" or "lame" containing GOF statistics
type	Character string: "auto" (default), "static", or "longitudinal". If "auto", determined by model class.
statistics	Character vector of statistics to plot, or NULL (default) for the standard panels plus any custom GOF columns. Unipartite names: "sd.row", "sd.col", "dyad.dep", "triad.dep", "trans.dep"; bipartite names: "sd.row", "sd.col", and "four.cycle.dep" for Gaussian outcomes or "four.cycles" otherwise. The internal GOF column names reported by names(fit$GOF) ("sd.rowmean", "sd.colmean", "cycle.dep") are accepted as equivalents; custom GOF columns may also be named. Unknown names are dropped with a warning.
credible.level	Numeric between 0 and 1; credible interval level for longitudinal plots (default 0.95)
ncol	Number of columns for faceted plot layout (default 2)
point.size	Size of points in longitudinal plots (default 2)
line.size	Width of lines in plots (default 1)
title	Optional title for the plot
...	Additional arguments forwarded to the ALS-specific gof_plot.ame_als when fit inherits from ame_als (nsim, seed). Its simulations condition on the fitted parameter estimates and resample outcome noise; they do not include parameter uncertainty. See ame_als_bootstrap for refit draws.

Details

Overview:

Goodness-of-fit (GOF) assessment is crucial for network models because standard residual diagnostics are often inadequate for capturing network dependencies. This function implements posterior predictive checking by:

1. Computing key network statistics from the observed data
2. Generating multiple networks from the model's posterior predictive distribution
3. Computing the same statistics on simulated networks
4. Visualizing the comparison to identify model inadequacies

Network Statistics Evaluated:

For unipartite (square) networks:

`sd.row` **(Out-degree heterogeneity)** Standard deviation of row means. High values indicate substantial variation in how active nodes are as senders/initiators. If the model underestimates this, it may be missing important sender effects or covariates.

`sd.col` **(In-degree heterogeneity)** Standard deviation of column means. High values indicate substantial variation in node popularity as receivers. Underestimation suggests missing receiver effects or popularity-related covariates.

`dyad.dep` **(Reciprocity/Mutuality)** Correlation between `Y[i,j]` and `Y[j,i]`. Positive values indicate reciprocity (mutual ties are more likely). The AME model captures this through the dyadic correlation parameter rho. Poor fit here points to adjusting the dcor parameter.

`triad.dep` **(Transitivity/Clustering)** Measures tendency for triadic closure (friend of a friend is a friend). Calculated as correlation between `Y[i,j]` and `sum(Y[i,k]*Y[k,j])/sqrt(n-2)`. AME captures this through multiplicative effects (U,V). Poor fit suggests increasing the latent dimension R.

For bipartite (rectangular) networks:

`sd.row` **(Type A activity variation)** Standard deviation of row means for Type A nodes. Indicates heterogeneity in how actively Type A nodes connect to Type B nodes.

`sd.col` **(Type B popularity variation)** Standard deviation of column means for Type B nodes. Indicates heterogeneity in how popular Type B nodes are with Type A nodes.

`four.cycles` **(Bipartite clustering)** Count of 4-cycles (rectangular paths A1-B1-A2-B2-A1). High values indicate that pairs of Type A nodes tend to connect to the same Type B nodes. Used for families with structural zeros, including binary outcomes.

`four.cycle.dep` **(Weighted bipartite dependence)** For Gaussian outcomes, averages products of centered, standardized cell values around fully observed rectangles. This measures weighted dependence; a nonzero-edge count would be fixed by the observed-cell pattern.

Interpretation Guide:

Histogram plots (static models):

- Dashed black vertical line: observed statistic value (dual-encoded on colour and linetype)
- Grey histogram: distribution from posterior predictive simulations

- Good fit: dashed line falls within the bulk of the histogram
- Poor fit: dashed line in the tail or outside the distribution

Common model inadequacies and solutions:

Observed sd.row/sd.col too high Model underestimates degree heterogeneity. Solutions: Add row/column covariates (Xrow, Xcol), enable random effects (rvar=TRUE, cvar=TRUE), or increase their variance.

Observed dyad.dep outside distribution Reciprocity not captured well. Solutions: For positive reciprocity, ensure dcor=TRUE. For negative, consider transformation or different family.

Observed triad.dep too high Clustering/transitivity underestimated. Solutions: Increase latent dimension R, add network covariates that capture homophily, or consider including community structure covariates.

Multiple statistics showing poor fit Fundamental model misspecification. Solutions: Change family, add missing covariates, or consider different model class.

Mathematical Details:

The posterior predictive p-value for statistic s is:

$$p = P(s(Y_{rep}) \geq s(Y_{obs})|Y_{obs})$$

where Y_rep is drawn from the posterior predictive distribution. Values near 0 or 1 indicate poor fit. The function visualizes the full distribution rather than just p-values for richer diagnostics.

Customization:

The function accepts custom GOF statistics through the model's custom_gof argument. These are automatically included in the plot. Custom statistics should capture network features important for your specific application.

Computational Notes:

GOF computation involves simulating multiple networks, which can be computationally intensive. The model uses the networks simulated during MCMC if gof=TRUE was specified. For post-hoc GOF, use the gof() function which generates new simulations.

Good model fit is indicated when:

- Observed statistics fall within the central 95% of posterior predictive distributions
- No systematic patterns of over/underestimation across statistics
- Custom statistics (if provided) also show adequate fit
- For longitudinal models, observed trajectories track within credible bands

Value

A ggplot2 object that can be further customized

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Fit an AME model
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit_ame <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, gof = TRUE,
               nscan = 100, burn = 10, odens = 1, verbose = FALSE, chains = 1)

# Basic GOF plot
gof_plot(fit_ame)

# Plot only degree-related statistics
gof_plot(fit_ame, statistics = c("sd.row", "sd.col"))
```

---

gof_plot.ame_als *Goodness-of-fit check for an ame_als fit*

Description

Bootstrap-style analogue of the MCMC gof_plot: draws nsim simulated networks from the fitted ALS model, computes the standard network statistics (sd.rowmean, sd.colmean, dyad.dep, cycle.dep, trans.dep for unipartite; sd.rowmean, sd.colmean, four.cycles for bipartite, replacing the count with weighted four.cycle.dep for Gaussian outcomes) on each replicate, and compares the observed value with the replicates.

Usage

```
gof_plot.ame_als(fit, nsim = 100, seed = NULL, ...)
```

Arguments

fit	an ame_als fit.
nsim	integer; number of replicates (default 100).
seed	optional RNG seed.
...	reserved.

Details

A cross-sectional fit gives one histogram per statistic with the observed value overlaid. A longitudinal fit gives, as gof_plot does for an MCMC lame fit, the statistics of *every* time slice: for each statistic, the observed value in each period against the median and the central 95 percent of the simulated values in that period.

Uses simulate.ame_als for the replicates and is therefore conditional on the fitted point estimates: the noise is resampled, but mu, beta, a, b, U, V are held fixed. The MCMC gof_plot.ame also averages over posterior movement in those parameters.

Value

A ggplot (or patchwork) object.

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
fit <- ame_als(Y, R = 1, family = "normal", verbose = FALSE)
gof_plot(fit, nsim = 20, seed = 1)
```

---

gof_stats *Goodness of fit statistics*

Description

Calculates goodness of fit statistics for relational data matrices, evaluating second-order (dyadic) and third-order (triadic) dependence patterns. Can handle both unipartite and bipartite networks.

Usage

```
gof_stats(Y, mode = NULL, custom_gof = NULL, family = NULL)
```

Arguments

Y	a relational data matrix. For unipartite networks, a square n x n matrix where Y\[i,j\] represents the relationship from node i to node j. For bipartite networks, an nA x nB matrix where Y\[i,j\] represents the relationship from node i in set A to node j in set B. Missing values (NA) are allowed and will be handled appropriately.
mode	character string specifying the network type: "unipartite" or "bipartite". If NULL (default), attempts to infer from matrix dimensions (rectangular = bipartite, square = unipartite). Note: square bipartite networks must specify mode="bipartite".
custom_gof	optional function or list of functions to compute custom GOF statistics. Each function should take Y as input and return a named numeric value or vector. Custom statistics will be added to the standard statistics.
family	optional outcome family, passed to bipartite diagnostics. Gaussian outcomes use four.cycle.dep, the mean standardized product around fully observed rectangles; see gof_stats_bipartite.

Details

The function computes network statistics that capture different aspects of network structure beyond simple density. These statistics are particularly useful for:

- Model checking: comparing observed statistics to those from simulated networks
- Model selection: choosing between models that better capture network dependencies

- Descriptive analysis: summarizing key structural features of the network

For bipartite networks with square dimensions (nA = nB), you must explicitly specify mode="bipartite" to ensure correct statistics are calculated.

Missing values in Y are handled by pairwise deletion for correlations and are excluded from matrix products in triadic calculations.

Value

A named numeric vector containing goodness-of-fit statistics. For unipartite networks:

sd.rowmean Standard deviation of row means. Measures the heterogeneity in out-degree centrality (sender effects).

sd.colmean Standard deviation of column means. Measures the heterogeneity in in-degree centrality (receiver effects).

dyad.dep Dyadic dependence/reciprocity correlation.

cycle.dep Cyclic/transitive triadic dependence.

trans.dep Transitive triadic dependence.

For bipartite networks:

sd.rowmean Standard deviation of row means (sender heterogeneity).

sd.colmean Standard deviation of column means (receiver heterogeneity).

four.cycles Count of four-cycles in the bipartite network.

If custom_gof is provided, additional statistics will be included with their user-specified names.

Naming note. The columns above are the internal names used by `gof_stats()` and stored on `fit$GOF`. `gof_plot` exposes a shorter user-facing alias for some of them; the mapping is:

internal (this column)	gof_plot alias
`sd.rowmean`	`sd.row`
`sd.colmean`	`sd.col`
`dyad.dep`	`dyad.dep`
`cycle.dep`	`triad.dep`
`trans.dep`	`trans.dep`
`four.cycles` (bipartite)	`four.cycles`

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
data(YX_nrm)

# Auto-detect unipartite
gof_stats(YX_nrm$Y)
```

```
# Bipartite (rectangular) networks are auto-detected; mode can also
# be given explicitly (required when a bipartite network is square)
Y_bip <- matrix(rnorm(120), 10, 12)
gof_stats(Y_bip, mode = "bipartite", family = "normal")

# Custom GOF function
my_stat <- function(Y) { c(my_measure = sum(Y > 0, na.rm = TRUE)) }
gof_stats(YX_nrm$Y, custom_gof = my_stat)
```

`gof_stats_bipartite` *Goodness of fit statistics for bipartite networks*

Description

Calculates goodness of fit statistics specifically designed for bipartite networks, evaluating degree heterogeneity and higher-order dependencies.

Usage

```
gof_stats_bipartite(Y, warn_square = TRUE, family = NULL)
```

Arguments

`Y`	a bipartite relational data matrix (nA x nB rectangular matrix) where Y\[i,j\] represents the relationship from node i in set A to node j in set B. Missing values (NA) are allowed and will be handled appropriately.
`warn_square`	logical; if `TRUE` (default) a warning is issued when `Y` is square (a possible unipartite matrix passed by mistake). Set `FALSE` for a genuinely square bipartite network.
`family`	optional outcome family. For `"normal"`, use weighted four-cycle dependence instead of a nonzero-edge count. Model fitting and post-hoc diagnostics supply this automatically.

Details

For Gaussian outcomes, `four.cycle.dep` is the mean product of four globally centered, sample-standardized values around each fully observed rectangle. It measures dependence among edge magnitudes. It returns `NA` when no complete rectangle is observed or the observed values have zero variance. Missing cells are not zeros. A nonzero-edge count is unsuitable for Gaussian predictive checks: continuous draws are nonzero almost surely, so fixing the observation mask fixes their four-cycle count regardless of model fit.

For bipartite networks, reciprocity and triadic closure are not meaningful concepts since edges only exist between the two node sets. Instead, this function focuses on:

- Degree heterogeneity in both node sets
- Four-cycles as the simplest higher-order dependence pattern

Value

A named numeric vector containing bipartite-specific goodness-of-fit statistics:

sd.rowmean Standard deviation of row means. Measures the heterogeneity in out-degree from set A nodes (sender effects). Higher values indicate more variation in how active A nodes are.

sd.colmean Standard deviation of column means. Measures the heterogeneity in in-degree to set B nodes (receiver effects). Higher values indicate more variation in how popular B nodes are.

four.cycle.dep For Gaussian outcomes, weighted dependence over complete observed rectangles, as defined below. Replaces the count.

four.cycles Count of four-cycles (also called 4-paths or squares) in the bipartite network. A four-cycle occurs when two nodes from set A (e.g., i and k) both connect to the same two nodes in set B (e.g., j and l), forming a closed path: i->j->k->l->i. This measures the tendency for pairs of A-nodes to share multiple common B-node connections, capturing a form of clustering specific to bipartite networks. High four-cycle counts indicate that connections are not random but show patterns of shared preferences or co-occurrence. For example, in a user-item network, many four-cycles suggest that users who like one item tend to also like other items that co-occur with it.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Create a random bipartite network
Y <- matrix(rnorm(10*12), 10, 12)

# Calculate GOF statistics
gof_stats_bipartite(Y, family = "normal")
```

---

gof_stats_unipartite *Goodness of fit statistics for unipartite networks*

Description

Calculates goodness of fit statistics specifically for unipartite (square) networks, evaluating second-order (dyadic) and third-order (triadic) dependence patterns.

Usage

```
gof_stats_unipartite(Y)
```

Arguments

Y	a square n x n relational data matrix where Y[i,j] represents the relationship from node i to node j. Missing values (NA) are allowed and will be handled appropriately. Diagonal values are typically NA for non-self-loop networks.

Details

This function computes network statistics that capture different aspects of network structure beyond simple density. These statistics are particularly useful for evaluating how well a model captures the observed network patterns.

The dyadic dependence statistic captures reciprocity - the tendency for relationships to be mutual. The triadic statistics capture different forms of triadic closure that are common in social networks.

Missing values in Y are handled by pairwise deletion for correlations and are excluded from matrix products in triadic calculations.

Value

A named numeric vector containing five goodness-of-fit statistics:

sd.rowmean Standard deviation of row means. Measures the heterogeneity in out-degree centrality (sender effects).

sd.colmean Standard deviation of column means. Measures the heterogeneity in in-degree centrality (receiver effects).

dyad.dep Dyadic dependence/reciprocity correlation. Measures the correlation between `Y[i,j]` and `Y[j,i]`, capturing reciprocity patterns.

cycle.dep Cyclic triadic dependence. Measures the tendency for directed cycles (i->j->k->i) in the network.

trans.dep Transitive triadic dependence. Measures the tendency for transitivity (if i->j and j->k, then i->k) in the network.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Create a random unipartite network
Y <- matrix(rnorm(100), 10, 10)
diag(Y) <- NA
gof_stats_unipartite(Y)
```

---

gof_temporal *Posterior-predictive temporal-trend test*

Description

For a fitted `lame` object, computes a network statistic (`density`, `reciprocity`, or `transitivity`) at each observed period, fits a least-squares linear trend on period index, and compares the observed slope to slopes from posterior-predictive replicates. The two-sided value is `min(1, 2 * min(p_up, p_down))`, where both empirical tails include ties. The value runs from 0 (observed slope in the extreme tail) to 1 (observed slope central in the predictive distribution).

Usage

```
gof_temporal(
  fit,
  stat = c("auto", "density", "mean", "reciprocity", "transitivity"),
  n_rep = 500,
  seed = NULL
)
```

Arguments

fit	A fitted lame object.
stat	One of "auto" (default: density for binary, cbin and ordinal fits, otherwise mean), "density", "mean", "reciprocity", or "transitivity". Density is the fraction of nonzero observed cells. Mean is the mean observed value. Unipartite checks exclude self ties; square bipartite checks retain diagonal cells. Reciprocity requires directed unipartite data; transitivity requires unipartite data. Gaussian density and nonzero transitivity are refused; use mean or weighted reciprocity for continuous outcomes.
n_rep	Number of posterior-predictive replicates to draw (each replicate is a full T-period network). Must be an integer at least 2.
seed	Optional RNG seed.

Details

Observed and replicated statistics use the same actor labels and observed cells in every period. Transitivity counts only fully observed triples. Joint posterior draws are required; missing storage is refused. Fit with posterior_options(save_ab = TRUE, save_UV = TRUE, thin_ab = 1, thin_UV = 1), omitting latent storage at rank zero. A constant observed series is still tested when the predictive slopes vary.

Value

A list with

- stat (chosen statistic name),
- slope_obs (observed slope of stat on period index),
- slope_rep (length-n_rep vector of replicate slopes),
- p_pp (two-sided posterior-predictive p-value),
- stat_obs_by_t, stat_rep_by_t (per-period statistics),
- n_rep (number of replicate networks drawn),
- degenerate (TRUE when the replicate slopes have no spread or a statistic is undefined, in which case p_pp is NA),
- reason and n_undefined describe an unestimated check,
- replication_method and draw_index identify joint draws

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X, family = "binary", R = 0,
            dynamic_beta = "dyad",
            posterior_opts = posterior_options(save_ab = TRUE, thin_ab = 1),
            nscan = 60, burn = 15, odens = 5, verbose = FALSE, chains = 1)
gof_temporal(fit, stat = "density", n_rep = 50)
```

`init_dynamic_ab_cpp` *Initialize dynamic additive effects with AR(1) structure*

Description

Initialize dynamic additive effects with AR(1) structure

Usage

```
init_dynamic_ab_cpp(n, Tn, rho_ab, sigma_ab, mean_a = 0, mean_b = 0)
```

Arguments

`n`	Number of actors
`Tn`	Number of time points
`rho_ab`	AR(1) parameter
`sigma_ab`	Innovation standard deviation
`mean_a`	Mean for row effects
`mean_b`	Mean for column effects

Value

List with initialized a and b matrices

IR90s *International relations in the 90s*

Description

A relational dataset recording a variety of nodal and dyadic variables on countries in the 1990s, including information on conflicts, trade and other variables. Except for the conflict variable, the variables are averages across the decade.

Format

A list consisting of a socioarray `dyadvars` of dyadic variables and matrix `nodevars` of nodal variables. The dyadic variables include

- total number of conflicts;
- exports (in billions of dollars);
- distance (in thousands of kilometers);
- number of shared IGOs (averages across the years);
- polity interaction.

The nodal variables include

- population (in millions);
- gdp (in billions of dollars);
- polity

Source

Michael Ward.

lame *AME model fitting routine for longitudinal relational data*

Description

An MCMC routine providing a fit to an additive and multiplicative effects (AME) regression model to longitudinal (time-series) relational data of various types. Supports both unipartite (square) and bipartite (rectangular) network structures. For cross-sectional (single time point) networks, use the `ame` function.

Usage

```
lame(
  Y,
  Xdyad = NULL,
  Xrow = NULL,
  Xcol = NULL,
  rvar = TRUE,
  cvar = TRUE,
  dcor = !symmetric,
  nvar = TRUE,
  R = 0,
  R_row = NULL,
  R_col = NULL,
  mode = c("unipartite", "bipartite"),
  dynamic_uv = FALSE,
  dynamic_ab = FALSE,
  dynamic_G = FALSE,
  dynamic_beta = FALSE,
  dynamic_rho = FALSE,
  dynamic_beta_kind = c("ar1", "rw1", "rw2", "matern32"),
  dynamic_uv_kind = c("ar1", "snap", "t"),
  family = "normal",
  intercept = !(family == "ordinal"),
  symmetric = FALSE,
  odmax = NULL,
  prior = list(),
  g = NA,
  seed = 6886,
  nscan = 10000,
  burn = 500,
  odens = 25,
  plot = FALSE,
  verbose = FALSE,
  gof = TRUE,
  start_vals = NULL,
  periodic_save = FALSE,
  out_file = NULL,
  save_interval = 0.25,
  model.name = NULL,
  save_log_lik = FALSE,
  posterior_opts = NULL,
  log_lik_path = NULL,
  log_lik_chunk_size = 10000L,
  keep_snap_draws = c("none", "summary", "draws", "chunked"),
  freeze_call = FALSE,
  dynamic_beta_pool = c("none", "rho", "sigma", "both"),
  dynamic_beta_per_actor = NULL,
  per_actor_covariate_idx = 1L,
```

```
  per_actor_identifiability = c("center", "exact_center", "drop_population"),
  keep_per_actor = c("auto", "draws", "summary", "none"),
  time_index = NULL,
  period_exposure = NULL,
  max_seconds = Inf,
  checkpoint_path = NULL,
  checkpoint_every = 100L,
  log_lik_method = c("observed_exact", "observed_ghk", "augmented"),
  ordinal_cutpoints = c("data_induced", "explicit"),
  method = c("mcmc", "als"),
  als_stability = c("none", "quick", "validation"),
  als_max_iter = 1000L,
  als_tol = NULL,
  als_lowrank_method = NULL,
  als_linear_solver = NULL,
  als_multistart = NULL,
  multistart = c("none", "cheap", "full"),
  bias_correct = NULL,
  bootstrap = 0L,
  bootstrap_type = c("parametric", "block"),
  bootstrap_block_length = 1L,
  bootstrap_seed = NULL,
  chains = .lame_chains_default(),
  cores = NULL,
  resume_from = NULL,
  print,
  sampler = c("gibbs", "tempered")
)
```

Arguments

Y	a T length list of relational matrices, or a 3D array of dimensions [n_row, n_col, T], where T is the number of time periods. Named lists may have changing actor composition; actors are aligned by row and column names. A longitudinal **netify** object is also accepted and converted with netify::to_lame(lame = TRUE). When family or mode is omitted, the value inferred by **netify** is used. See family below for data types.
Xdyad	a T length list of n x n x pd arrays of covariates
Xrow	a T length list of n x pr matrices of nodal row covariates
Xcol	a T length list of n x pc matrices of nodal column covariates
rvar	logical: fit row random effects (asymmetric case)? With method = "als" and a normal, binary or Poisson dynamic fit by the empirical-Bayes estimator, rvar, cvar, nvar and dcor are honoured (a disabled margin is a point mass at zero, as in the sampler; see dcor for the correlation); the penalised routes name them in the warning about ignored arguments.
cvar	logical: fit column random effects (asymmetric case)? See rvar for method = "als".

dcor — logical: fit a dyadic correlation (asymmetric case)? The correlation ρ of the two residuals of a dyad carries the prior $p(\rho) \propto 1 - \rho^2$ on $(-1, 1)$, which vanishes at the boundaries. Information about it depends on joint presence and absence of ties and the fitted means. With fewer than one mutual dyad per period, or fewer than 30 in all, the data inform it mainly through the absence of mutual ties, and a message gives the count. Its posterior can then stay near the prior or move well away from it: toward -1 when the fitted means predict more mutual ties than the panel has (on the 40-state MID panel, -0.58 with 95 percent interval -0.92 to -0.13). The additive effects, coefficients and latent positions all retain the same sampled correlation, including on panels with few mutual dyads. `dynamic_uv = TRUE` keeps the correlation: the per-period position kernel whitens each dyad's pair of residuals by the current ρ, exactly as the static-position kernel does, so the reciprocity is carried by ρ instead of being absorbed into the coordinates. A period-varying ρ (`dynamic_rho`) needs static positions. A Poisson residual variance below 0.1 leaves ρ to its prior in the same way, and a message says so. An exactly symmetric `Y` fitted with `symmetric = FALSE` has equal reciprocal residuals, so the posterior of ρ concentrates near one (with `dcor = FALSE` the two cells of a dyad count as two observations); a message says so and points at `symmetric = TRUE`, which fits the undirected model. A normal model with `dcor = TRUE` has no proper posterior once the reciprocal pairs of the panel outnumber what its mean parameters absorb by four (the bound ame applies, pooled over the periods: they share ρ, so the excess of every period adds, and a parameter they share absorbs its contrasts once): ρ runs to one, with static or dynamic positions or none, and the variance draw breaks down. Such a fit is an error of class `lame_symmetric_y`. With `method = "als"` a directed binary fit (the probit link of the empirical-Bayes default) models the correlation: the two cells of a dyad are one bivariate probit observation, the correlation is one number for all periods (`fit$VC["rho"]`; the record, with its standard deviation, in `fit$rho_estimation` and `fit$eb$pair`), and it is estimated with the other hyperparameters under the prior above. A directed Poisson fit takes the two counts of a dyad as one observation at the fitted state of independent cells: the correlation is the posterior mean, under the same prior, of the profile of the pair probability, and the information of the pairs enters the standard errors. A directed normal fit takes the two cells of a dyad as one bivariate normal observation, in closed form, and estimates the correlation under the same prior; a directed network whose reciprocal pairs are all exactly equal is an error of class `"lame_improper_normal_reciprocity"`, as in ame (fit it with `symmetric = TRUE` or `dcor = FALSE`). `dcor = FALSE` fits independent cells. Undirected and bipartite networks have no pair of cells. On the dynamic route the pair of cells of a binary fit is integrated over the uncertainty of its latent factors or moving positions, and the fit takes 1.4 to 2.7 times as long as the independent-cell fit (1.4 with additive effects only, 1.9 to 2.7 with static factors or moving positions); `options(lame.als_threads)` splits the work over threads and changes nothing else.

nvar — logical: fit nodal random effects (symmetric case)? See `rvar` for `method = "als"`.

R — integer: dimension of the multiplicative effects (can be zero). A binary fit whose multiplicative term reproduces the observed ties exactly is separated: the size of

the term is then set by its prior, the fit says so, and the signs and the fitted probabilities are what the data determine; a smaller R (down to zero for a panel whose ties barely change over time) reproduces them without saturating. With method = "als" and dynamic_uv = FALSE, the latent factors of a normal, binary or Poisson dynamic fit are static and have the variational posterior of the static estimator (ame_als): their posterior means are reported as U and V, the scale of their prior in fitebfactor_prior, and the standard errors are marginal over the factors and carry the uncertainty of that scale (fitebfactor_scale$sd_log; options(lame.eb_scale_uncertainty = FALSE) leaves it out).

R_row — integer: for bipartite networks, dimension of row node multiplicative effects (defaults to R)

R_col — integer: for bipartite networks, dimension of column node multiplicative effects (defaults to R)

mode — character: either "unipartite" (default) for square networks or "bipartite" for rectangular networks

dynamic_uv — logical: fit dynamic multiplicative effects (latent factors). The transition model is selected by dynamic_uv_kind. A directed fit with dynamic positions carries the dyadic correlation of dcor through the position kernel (see dcor). fitU / fitV hold the positions factored from each period's posterior mean product (see latent_positions); the per-iteration cubes are kept as fitU_draws / fitV_draws by the default posterior_opts (or with posterior_opts = list(save_UV_draws = TRUE)), and give the posterior standard deviations. Default FALSE. With method = "als" and a normal, binary or Poisson outcome the positions are estimated by the dynamic empirical-Bayes estimator (see bias_correct): fit$U and fit$V are the posterior means (n x R x T) with posterior standard deviations fit$U_sd and fit$V_sd, fit$Oarr (alias fit$UVPM) is the posterior-mean product, fit$rho_uv and fit$sigma_uv are the persistence and innovation scale, and fitebuv_scale and fitebuv_search are the stationary variance and the search that chose it. Directed, bipartite (R_row and R_col may differ: the interaction has rank min(R_row, R_col)) and undirected networks are covered.

dynamic_ab — logical: fit dynamic additive effects (sender/receiver effects) that evolve over time. When TRUE, the row effects (a) and column effects (b) become time-varying as a per-actor level plus an AR(1) deviation (see the model description in the details). This captures temporal heterogeneity in actors' baseline propensities to send and receive ties, allowing for smooth changes in activity levels and popularity over time. For example, an actor's tendency to form outgoing ties might gradually increase or decrease across observation periods. The AR(1) specification ensures temporal smoothness while allowing for actor-specific evolution patterns. Implementation uses conjugate updates where possible and C++ for computational efficiency. Default FALSE. Each dynamic effect is a per-actor level plus a zero-mean AR(1) deviation: the level is pooled over every period (its variance is the reported va / vb) and the deviation carries the persistence rho_ab and innovation scale sigma_ab. The marginal variance of a per-period effect is therefore va + sigma_ab^2 / (1 - rho_ab^2), and rho_ab describes the deviations, so on short panels it reads lower than the persistence of the raw effect path. Both parts are drawn from the likelihood: the levels given the deviations

and the deviations given the levels, so persistent actor heterogeneity lives in the effects. The regression coefficients are drawn conditional on the current effects, and the effects are drawn conditional on summing to zero in every period when an intercept is included. Under that constraint an individual actor's marginal variance has the additional factor `1 - 1/n`, where `n` is the number of actors in that margin.

The reported intercept and effects are the identified quantities when the fit has a multiplicative term; see the details.

A directed binary panel carrying fewer than 1.2 events per actor-period (two periods or more) reports `va` and `vb` as not estimated, and fewer than 0.6 (1.6 with a multiplicative term) its intercept. On simulated panels of 30 actors and 5 periods (static and dynamic effects, ranks 0 and 1, 64 to 252 panels a rate bin) the 95 percent intervals of `va` and `vb` covered 0.31 and 0.28 of the panels below 0.6 events per actor-period, 0.73 and 0.68 from 0.6 to 0.9, 0.91 and 0.81 from 0.9 to 1.2, and 0.91 to 1.00 from 1.2; the intercept covered 0.57 to 0.87 of the panels below its rate and 0.92 or more from it. The persistence `rho_ab` (coverage 1.00 at every rate) and the innovation scale `sigma_ab` (0.84 to 1.00) are reported at every rate, and so is `cab` (0.97 to 1.00). The draws remain in `fit$VC` and `fit$BETA`; `fit$events_per_actor_period` and `fit$events_per_actor` record the event rates. Supplying an effects prior (`prior$Sab0`, `eta0` or `etaab`) reports `va` and `vb` under that explicit prior, which the summary names. Count families keep reporting their variance components. The stored paths `a_dynamic` and `b_dynamic` contain the full level-plus-deviation effects. Static latent positions are centered for reporting, with the removed product components carried by the reported intercept and effects. Dynamic latent coordinates retain the coordinates drawn under their proper path prior. Compare fitted predictors and per-period products across chains; raw coordinates can differ through latent rotations.

`dynamic_G` logical. Not supported: `TRUE` is refused with an error on both the MCMC and the ALS route, and the argument is kept only so that existing calls and saved call objects still parse. With moving positions a per-period interaction matrix is not identified apart from them, because $U_t G_t V_t'$ and $(U_t G_t) V_t'$ describe the same products; with static positions it is identified but not implemented. The interaction matrix exists only in bipartite fits. Use `dynamic_uv = TRUE` and read the per-period product `fit$UVPM`. Default FALSE.

`dynamic_beta` logical, character, integer, or logical-vector flag selecting which regression coefficients evolve over time via independent AR(1) processes. Default FALSE keeps every coefficient static (the historical behaviour). Accepted forms:

- `FALSE` / `NULL`: no coefficient is dynamic.
- `TRUE`: every coefficient is dynamic.
- character vector of block shortcuts (`"intercept"`, `"dyad"`, `"row"`, `"col"`) or specific coefficient names from `colnames(BETA)` – those become dynamic.
- integer vector: 1-based column indices into `colnames(BETA)`.
- logical vector of length `p`: per-coefficient mask.

Each dynamic-coefficient block (one per distinct intercept / dyad / row / col label that has at least one dynamic coefficient) gets its own AR(1) parameters

ρ_β (truncated-Normal prior, default mean 0.8, bounded between 0 and 0.999) and σ_β^2 (inverse-Gamma prior, default shape 2 with a scale set from the response scale, see `prior`). The joint posterior over the time path β_t is drawn by Forward-Filter / Backward-Sample (FFBS) inside the MCMC loop, conditional on the current static-block beta and on (a, b, U, V). When the intercept or a nodal coefficient is dynamic, a sum-to-zero contrast basis is applied to the additive-effect sampler to keep $a_i + intercept_t$ identified. Requires at least 2 time periods. `dynamic_beta` composes with multiplicative latent factors (`R` > 0) and additive random effects for every family, across unipartite/bipartite and directed/symmetric panels and in combination with `dynamic_ab` / `dynamic_uv`. The state equation is a zero-mean AR(1) on the coefficient itself, $\beta_t = \rho_\beta \beta_{t-1} + e_t$, with no level term, so ρ_β is the persistence of the coefficient's level toward zero rather than the persistence of its movement around a level. A coefficient whose level is far from zero therefore has a persistence close to 1 whatever its movement: for a path with mean m, deviation sd s and deviation persistence r the value this state equation implies is $(m^2 + rs^2)/(m^2 + s^2)$, which is 0.95 for a coefficient of 0.7 whose deviation has sd 0.3 and persistence 0.7. Read ρ_β that way, and read the path `BETA[, k, ]` for how much the coefficient moves. With a single dynamic coefficient block and small T, only $T - 1$ transitions inform its AR(1) persistence, so the data inform ρ_β weakly and its posterior stays near the prior; the path `BETA[, k, ]` is read separately from the hyperparameters, and `dynamic_beta_prior_summary` shows the paths the prior implies. The hyperparameters of each block (ρ_β and σ_β, and the pooled scale and persistence of `dynamic_beta_pool`) are drawn from their exact conditionals given the path at every sweep (a draw costs well under a millisecond). Earlier versions drew them every 20th sweep; on normal and binary panels that left the scale and the persistence with 4 to 20 times fewer effective draws (panels of 5 to 170 actors; at 170 actors 357 of 1,600 draws of the scale on a normal panel and 304 on a binary one, where every sweep gives 1,599 and 1,335). An interweaving step also moves the hyperparameters together with the path, before the path is redrawn given them. It proposes new values of $(\sigma_\beta, \rho_\beta)$ by a symmetric walk on $(\log \sigma_\beta, \mathrm{logit}\, \rho_\beta)$, rebuilds the path from the same standardised innovations (the state equation divided by the innovation scale; `"rw2"`: the part of the path that its second differences leave flat stays fixed; `"matern32"`: the path scales with σ_β, the length scale being fixed) and accepts with the Metropolis ratio of the likelihood of the rebuilt path and the hyperprior (the pooled hierarchies included). It targets the funnel of hierarchical models: where the data say little about the path (few actors, many periods) and the prior of the scale is diffuse, the centred draws alone move the scale slowly, because the scale given the path is pinned at the spread of the path in hand. On a panel of six actors and 30 periods with `prior$sigma_beta_shape = 0.5` and `prior$sigma_beta_scale = 1e-4` the step raised the effective draws of the scale 2.8 times and those of the persistence 2.0 times over the draws at every sweep; at the default prior it raised them 1.0 to 1.5 times on the panels checked (5 to 30 actors, 12 to 60 periods), which about repays the time it takes there (10 to 60 percent of a sweep). Where the data pin the path (a dense panel) about nine proposals in ten are rejected and the step costs 3 percent of a sweep at 170 actors. `prior$beta_hyper_nc = FALSE` omits the step, and `fit$beta_hyper_move` records its attempts and ac-

cepts. With `method = "als"`, selected intercept, dyadic, row-node, and column-node coefficients are supported on normal, binary, and Poisson panels, including named panels where actors enter or exit. Dynamic ALS uses period-specific `Xrow` or `Xcol` values for selected dynamic node coefficients; static node coefficients still use per-actor means. Dynamic intercepts are unavailable when `family = "ordinal"`. Default FALSE.

`dynamic_rho` logical. For directed unipartite normal models, allow the residual dyadic reciprocity parameter ρ_t to vary by period instead of using one pooled dyadic correlation across the whole panel. It requires static latent positions (`dynamic_uv = FALSE`). MCMC fits store per-draw paths in `fit$RHO` and the posterior mean path in `fit$rho_path`. This option is not supported with `method = "als"`; use a supported MCMC configuration for period-specific reciprocity. Dynamic ALS fits use one pooled correlation, `fit$VC["rho"]`, and report period-specific residual diagnostics in `fit$residual_rho_path`. The legacy `fit$rho_path` field remains a diagnostic alias, labelled by `fit$rho_path_role = "residual_diagnostic"`; it is not a fitted covariance path. `als_start_vals()` uses the pooled ALS correlation, including for legacy ALS objects. Default `FALSE`.

`dynamic_beta_kind`

character: state-space prior on the dynamic coefficient block(s). `"ar1"` (default) gives mean-reverting AR(1) with a truncated-Normal prior on ρ_β. `"rw1"` gives a random walk (ρ_β pinned at 1, no truncation), appropriate when you expect permanent drift with no mean reversion – e.g. trade-gravity coefficients in a permanently changing world economy. The alias `"random_walk"` is also accepted for `"rw1"`. When the stationarity warning in `summary(fit)` fires under the default AR(1), refit with `dynamic_beta_kind = "rw1"` for cleaner inference. Decision tree: **mean-reverting?** AR(1). **Permanent / unit-root drift?** RW1. **Smooth with curvature?** `"rw2"` (second-order random walk). **Smooth with a known length-scale?** `"matern32"` (Matern 3/2). Note: `"rw2"` and `"matern32"` use an R-level joint Gaussian sampler and run substantially slower per iteration than the C++ FFBS used for `"ar1"` / `"rw1"` (roughly 2-4x on n = 100, T = 10). On `method = "als"` the coefficient paths are `"ar1"`, `"rw1"` or `"rw2"`, including the empirical-Bayes estimator; `"matern32"` is refused there and needs `method = "mcmc"`.

`dynamic_uv_kind`

Character string selecting the transition model for dynamic multiplicative latent positions when `dynamic_uv = TRUE`. `"ar1"` uses Gaussian AR(1) drift, `"snap"` uses a mixture of AR(1) drift and discontinuous reset transitions, and `"t"` uses heavy-tailed Student-t innovations represented by local transition scales. On the MCMC path, `"snap"` and `"t"` are available for directed unipartite fits; symmetric fits use `"ar1"` (the symmetric product's unit-variance coordinates do not hold their scale under the reset and scale-mixture transitions). Under `"snap"`, a reset lands on the stationary distribution of the position paths unless `prior$snap_kappa` fixes its scale, and `rho_uv` and `sigma_uv` are updated jointly with the latent reset indicators. At rank one, `summary()` and `print()` report `rho_uv`, `sigma_uv` and (for `"snap"`) `pi_snap`. At rank two or more, simulation studies found the `"snap"` chains settle where extra resets stand in for drift, so those three are not reported for it (the draws stay on the fit). A `"t"` chain starts from a dynamic ALS fit and then recovers `rho_uv` and `sigma_uv` at any rank; if that start is

unavailable they are not reported at rank two or more. Both kinds receive conditional sign updates that preserve the fitted products and help chains explore alternative trajectory orientations. With `method = "als"`, `"snap"` is refused. The ALS `"t"` kind supports directed and symmetric normal, binary and Poisson panels; bipartite Student ALS is refused. A normal, binary or Poisson fit with `bias_correct = NULL` is fitted by the dynamic empirical-Bayes estimator for `"ar1"` (Gaussian) and `"t"` transitions (directed and undirected networks): the weights of the scale mixture have a variational posterior, and the degrees of freedom are `prior$t_nu` (default 4). ALS Student-t dynamic UV fits attach the final local transition-weight matrices as `fit$lambda_u` and `fit$lambda_v` (n x T, one in the first period).

`family` character: one of "normal","binary","ordinal","cbin","frn","poisson" - see the details below

`intercept` logical: fit model with an intercept?

`symmetric` logical: is the sociomatrix symmetric? Not available for the `"cbin"` and `"frn"` families, whose per-actor ranked nominations are directed by construction. A symmetric fit needs a symmetric `Y` in every period and symmetric `Xdyad` slices, as in `ame`: the regression term of a symmetric fit is symmetric, so an asymmetric slice is an error (class `lame_symmetric_xdyad`) that names the slice and the first period; so is it with `method = "als"`. Replace it by `(X + t(X)) / 2`. A value missing in one triangle of a slice is not an asymmetry: it is filled from the other triangle.

`odmax` a scalar integer or vector of length n giving the maximum number of nominations that each node may make - used for "frn" and "cbin" families. A cap below a row's observed out-degree contradicts `Y` and is an error. A cap that most rows' observed outdegrees reach erases the cross-row spread in outdegree that the sender effects rest on: a row at its cap shows only that its level is high enough to fill the cap. The sampler pools the prior scale of the row side with the column side once 80 percent of the observed rows (60 percent in a single period) nominate the full `odmax`, counting a row with hidden cells by what a full cap thins to at its observed share; `fit$odmax_saturated_share` records the share and `fit$odmax_sat_threshold` the threshold. `va`, `vb` and `cab` are reported at every share: on simulated panels of 24 to 30 actors their 95 percent intervals covered the generating values in 0.88 to 1.00 of the panels at every share, with and without a fifth of the cells hidden and at generating sender sds of 0.25, 0.5 and 1 (at 0.25 the posterior mean of `va` runs up to 2.3 times the generating variance while the interval still covers). `va` and the level of the rows follow the design. A design of two periods or more with a cap share of 0.5 or more and a posterior-mean `va` from 0.5 to 1.5 reports the intercept and the row-covariate coefficients as not estimated: the rows below the cap carry too little of the spread of the senders and the posterior of `va` sits near its prior. In simulated panels (24 actors, 4 and 6 periods, one chain at the package defaults, 1440 fits at generating sender sds of 0.5, 1 and 1.5) the intercept was covered in 0.89 of the 312 panels in that region (0.76 at sd 0.5 and 1.5, 0.97 at sd 1) and in 0.96 of the 1128 outside it. `va` is reported in the region: its interval covered 0.91 of the 312 panels (0.79 at sd 0.5, 1.00 at sd 1, 0.72 at sd 1.5) and 0.99 outside it. The posterior mean of `va` ran 0.18 and 0.23 above the generating variance at sd 0.5 (shares 0.3 and 0.7), 0.08 and 0.00 above it at sd 1 and 0.02 and 0.18 below it at sd 1.5. In a single

period (ame() and one-period lame() fits) a row at its cap leaves the level of the rows unbounded above: with every row at the cap the intercept interval covered 0.64 of the panels, and with cells hidden and a share from 0.5 it covered 0.84, against 0.98 or more elsewhere. The intercept and any row-covariate coefficient are reported as not estimated there, with the draws kept in fit$BETA. On 200 simulated 170 by 100 bipartite panels (cap 10, shares 0.3 and 0.7) the intercept was covered in 1.00 of the panels, vb in 0.98 to 1.00 and va in 0.90 to 1.00, its posterior mean 0.09 to 0.15 below the generating variance; nothing is withheld there. When the effects prior is supplied (prior$Sab0, eta0 or etaab), the fit names the prior in the summary (fit$prior_supplied records it); a prior on the effects does not fix the level of a row, so the intercept stays withheld in those designs, and va is reported under that prior. A cap the design itself imposes cannot be relaxed: read APM / BPM as relative sender and receiver levels there.

prior — a list containing hyperparameters for the prior distributions. Available options and their defaults:

s20 Fixed positive prior scale for the dyadic residual variance v_e. The prior has one degree of freedom. By default, normal fits use the variance of the observed responses, computed once before sampling; constant responses use their mean square, or 1 if all zero. Poisson fits use 1 on the latent log-rate scale. The resolved scale is reported as fit$s2_prior_scale. It is never recomputed from the current residuals. A supplied value overrides this default. The coefficient prior is controlled separately by g.

uv_prior For sampler = "tempered", "hierarchical" (default) or the explicitly different "unit" coordinate prior. This option is refused on the Gibbs route. See Details.

Sab0 Prior scale matrix for the additive-effects covariance. A 2x2 matrix where Sab0\[1,1\] is the prior variance for row effects, Sab0\[2,2\] is the prior variance for column effects, and off-diagonals control correlation between row and column effects. For the "normal" and "poisson" families, and for unipartite "binary", "ordinal", "cbin" and "frn", this defaults to diag(2) scaled to the observed sender/receiver heterogeneity in Y (for the ordinal, cbin and frn families, of its probit-moment scores) rather than to a fixed diag(2); see ame for the rationale. For "cbin" and "frn" panels of at least two periods the two sides get their own scale, the row side read with the rows at their nomination cap as censored observations. This keeps the receiver spread from setting the sender's prior scale when capped outdegrees contain little sender information. Where the rows at the cap reach the reporting threshold of odmax the pooled scale is kept. Bipartite fits of the probit-scale families use diag(2). Pass diag(2) explicitly for a fixed unit-scale prior.

eta0 Prior degrees of freedom for the additive-effects covariance Σ_{ab} (default: round(4 * vdfmlt) for unipartite "binary", "ordinal", "cbin" and "frn" fits, where vdfmlt is a probit-moment variance multiplier estimated from Y, and round(4 + 3 * n/100) otherwise, where n is the number of actors). Higher values impose stronger shrinkage of the row/column effects toward the prior scale. The multiplicative-effects static unipartite prior is the one of ame: an inverse-Wishart covariance with the degrees of freedom and the data-scaled Suv0 of kappa0 and Suv0 in ame, read off every period of the

panel (the static factors are shared by the periods). `Suv0` and `kappa0` apply to `ame` of a unipartite network only; a bipartite network has no such covariance, and supplying either is an error of class `"lame_prior_unused"`.

etaab Prior degrees of freedom for covariance of additive effects (default: 4 + 3 * n/100). Controls shrinkage of row/column random effects.

ab_min_observed Observation count used to shrink sparse actor initial values (default: max(10, T)). Subsequent actor updates use the specified Gaussian prior without deterministic shrinkage.

rho_uv_mean For dynamic_uv=TRUE: Prior mean for UV AR(1) persistence. Gaussian bipartite AR(1) MCMC fits default to 0.5; other routes default to 0.9. Values close to 1 indicate high persistence.

rho_uv_sd For dynamic_uv=TRUE: Prior SD for UV AR(1) persistence. Gaussian bipartite AR(1) MCMC fits default to 0.2; other routes default to 0.1. Explicit settings override these defaults.

sigma_uv_shape For dynamic_uv=TRUE: Shape parameter for inverse-gamma prior on UV innovation variance (default: 2). Symmetric fits do not draw the innovation variance: their coordinate paths have unit stationary variance, so the innovation SD is $\sqrt{1-\rho_{uv}^2}$ and the diagonal `L` of the product carries the scale.

sigma_uv_scale For dynamic_uv=TRUE: Scale parameter for inverse-gamma prior on UV innovation variance (default: 1; for the normal family of a directed unipartite network the typical size of the additive effects, the scale of the factor prior above, so that the prior is the one of a response measured in other units. The positions of a bipartite or undirected network are unit free: the interaction matrix and the weights carry the unit).

uv_max_abs For dynamic_uv=TRUE: absolute bound on each free latent coordinate. Each actor-vector conditional rejects proposals outside this joint prior support. Default 8 for binary, ordinal, cbin and frn families and 50 for normal and Poisson families (the bound of a unipartite normal fit is multiplied by the square root of `sd(Y)`: its coordinates carry the square root of the unit of the response). Set to `Inf` to remove the restriction. A finite bound changes the prior; no deterministic coordinate normalization is applied. The box is part of the prior on the scale of the positions, which the data do not fix in a bipartite fit (the interaction matrix carries it) and which `uv_var_cap` already limits: on the validation panels (50 actors, 5 periods; bipartite 40 by 30) the largest coordinate drawn under the default box was 7.8 (the box is 8 on the probit scale) and 6.9 with the box removed, a proposal fell outside it in at most 1 of 10000 sweeps, and the coefficients, variance components, fitted values and posterior mean of the product UGV' with and without the box differed by about as much as two chains of the same fit do (at most 0.0014 root-mean-square for the product, and none for the cbin and Poisson panels, where no proposal came near the box). It matters when a fit separates the data: without it the product runs away.

G_max For bipartite fits with a static interaction matrix: absolute bound on entries of G. A joint Gaussian conditional proposal outside the bound is rejected. Default 20 for binary, ordinal, cbin and frn families with dynamic_uv=TRUE and 100 otherwise (multiplied by `sd(Y)` for the normal family: the interaction matrix carries the unit of the response). Set to `Inf`

to remove this prior restriction. On the validation panels the largest entry drawn was 2.7, a tenth of the default bound, and no proposal fell outside it.

sigma_uv_max For dynamic_uv=TRUE: rail on the innovation SD of the latent coordinate paths. Default 10 for bipartite fits, where G absorbs the scale, and none for unipartite fits. On the validation panels the posterior of the innovation SD stayed below 1.4, a seventh of the rail.

uv_var_cap For dynamic_uv=TRUE: bound on the stationary variance of a latent coordinate path. This is joint prior support for `sigma_uv` and `rho_uv`; both updates respect it. A finite cap also constrains the residual-variance update for normal responses. The bound is a multiple of the observation variance for the normal family and absolute on the latent scale otherwise: the binary, ordinal, cbin and frn families fix that variance at one, and for the Poisson family the latent scale is the log rate, whose overdispersion variance does not enter. Default 4 for the binary, Poisson, ordinal, cbin and frn families and `Inf` for the normal family. In a bipartite fit the interaction matrix carries the scale of the paths and the data do not identify it, so the posterior of the stationary variance lies against this bound (within 10 percent of the default in 29 percent of the sweeps of the validation panel, against none in the unipartite ones; without the bound its 1st to 99th percentiles run from 3 to 76). Between a cap of 4, a cap of 16 and none the coefficients and the variance components of the validation fit moved by at most 1.2 percent, while `rho_uv`, `sigma_uv` and the norms of `U` and `V` follow the cap: read a bipartite fit through the product UGV', not through its factors. For the binary, ordinal, cbin and frn families `uv_max_abs` also defaults to 8 and `G_max` to 20 rather than 50 and 100, so that once a fit separates the data the product UGV' is bounded on the latent scale instead of running away.

uv_temper_move Logical, default `TRUE`. For symmetric Poisson fits with `dynamic_uv` of the `"ar1"` kind: every `uv_temper_every` sweeps (a positive whole number, default 5; a fit with the transition and the `uv_nc_move` step costs 1.4 times one without them) make a tempered transition (Neal 1996) of one randomly chosen eigen-direction (its weight and coordinate paths, the latent array carried with the predictor), which crosses between the signatures of the weights that the other draws cross only in thousands of sweeps. Recorded in `fit$uv_temper_move`.

uv_nc_move Logical, default `TRUE`. For `dynamic_uv` of the `"ar1"` kind: in symmetric fits, draw the persistence `rho_uv` given the standardised innovations of the paths (the paths then follow the new persistence), and in bipartite fits draw the factor by which the paths, the innovation scale `sigma_uv` and the interaction matrix trade scale at no cost to the predictor. The conditional draws cross these directions slowly (30 to 70 effective draws of 400 for the persistence of a 50-actor symmetric panel). `FALSE` omits them.

cov_ridge_move Logical, default `TRUE`. For directed fits with a row or column covariate (a design slice constant along the columns or the rows) and static coefficients: each sweep also draws, exactly, the amount by which the coefficient and the additive effects of that margin (the levels and, with `dynamic_ab`, the deviation paths; the intercept takes the constant of the centring) trade against one another without changing any cell's predictor, up to

a constant of each period that the latent array absorbs. The conditional draws cross this ridge slowly (the coefficient of a period-varying nodal covariate had 38 to 99 effective draws of 800 on a 170-actor ordinal panel). `FALSE` omits it.

sparse_position_move For directed `"cbin"`, `"frn"` and `"ordinal"` fits with latent factors (`R > 0`, not `dynamic_uv`): each sweep also draws the position of every sender and receiver together with the latent values of its row or column, exactly. Moving u_i changes the predictor of row i by a different amount in each column; carrying the latent values along changes no residual, so the conditional of the position is its prior times the indicator that the shifted row is still a latent row the data allow, drawn by an elliptical slice step. Actors the data bound on one side only (no nominations, no nominators, only the lowest category) move across their wide posterior in a few sweeps instead of by random walk. Not applied to `"cbin"` / `"frn"` panels with hidden cells. Recorded in `fit$sparse_position_move`. The coordinate paths of `"cbin"` and `"frn"` fits with `dynamic_uv` of the `"ar1"` kind (bipartite or directed) move in the same way, each path as a whole, every second sweep unless `sparse_position_every` says otherwise. An ordinal fit that runs it visits the level translations of `actor_move` every sweep unless `actor_move_every` says otherwise.

sparse_position_every Sweeps between visits of `sparse_position_move` (default 1: every sweep).

uv_split_move For directed fits with static positions (`R > 0`, not `dynamic_uv`): draw the split of the scale between the two factors exactly (default `TRUE`). Multiplying U by c and V by $1/c$ leaves the product, so the likelihood, unchanged; with the covariance of the rows integrated out the factors have a matrix-t prior, and $\log c$ is drawn from its density along the orbit by a slice step. The conditional draws of the sweep moved the split by a few percent per sweep against a posterior sd of 0.2 to 0.5 (24 to 54 effective draws of 1,600 in the static directed cells of the acceptance sweep, 4 on a 170-actor ordinal panel); the coordinates of the actors the data say little about, and their products, followed it. `FALSE` omits the draw.

symmetric_node_tie For symmetric binary and ordinal fits with a nodal covariate: draw the row and the column coefficient of the covariate as one (default `TRUE`). The design carries a nodal covariate twice, and the coefficients of its two slices enter the predictor of a symmetric network only through their mean, so the data leave their difference to its prior. The exact symmetric draws of the binary and ordinal families return a difference of the width of that prior, and the draws of the additive effects, of the positions and of the dynamic coefficients that read the predictor carried it as noise: a fit with `dynamic_ab` and a time-varying nodal covariate let the node coefficient and the effect paths walk off together (the innovation sd of the paths ran to 20 and more against 0.3), and the persistence of dynamic positions was pulled down. Each draw of the coefficients is now followed by replacing the pair by its mean, which keeps the predictor symmetric. Fits without dynamic effects, positions or coefficients, and the normal and Poisson families, are not affected. `FALSE` restores the earlier draws.

rho_ab_mean For dynamic_ab=TRUE: Prior mean for additive effects AR(1)

parameter (default: 0.8). Controls temporal smoothness of sender/receiver effects.

rho_ab_sd For dynamic_ab=TRUE: Prior SD for additive effects AR(1) parameter (default: 0.15).

sigma_ab_shape For dynamic_ab=TRUE: Shape parameter for inverse-gamma prior on additive effects innovation variance (default: 2).

sigma_ab_scale For dynamic_ab=TRUE: scale parameter of the inverse-gamma prior on the innovation variance of the effect deviations. It is set once before sampling to the prior mean of `Sab` (that is `etaab * Sab0 / (etaab - 3)`, the mean of the inverse Wishart the covariance is drawn from) times `1 - rho_ab_mean^2`, so that a priori the stationary variance of an actor's deviation path matches the variance of the levels and per-period deviations are still pulled toward the actor's level unless the data insist; it is reported as `fit$sigma_ab_prior_scale`. This is a prior-scale convention. Because it is read off the prior of the additive-effect covariance rather than its current draw, it still follows the scale of the outcome while leaving the `sigma_ab` and `Sab` draws as conditionals of one joint density. Set a value to override it.

ab_var_cap For dynamic_ab=TRUE: optional bound on the stationary variance of an additive-effect path, in the same units as `uv_var_cap`; the innovation scale is then drawn so that `sigma_ab^2 / (1 - rho_ab^2)` stays at or below it. Default `Inf` (no bound). Set it, for example to 4, only when the additive effects themselves are running away; on a near-separable panel a bound here pushes the separation into the interaction matrix instead.

actor_factor_slice For supported directed binary rank-two AR(1) factor panels with dynamic actor effects and static coefficients, use paired actor/factor elliptical slice updates of the observed likelihood. Default `TRUE`; `FALSE` disables this additional transition. Actor centering and factor bounds are preserved.

bipartite_scale_move For Gaussian, binary, Poisson or ordinal bipartite panels (the last two with `actor_move`) with moving AR(1) factors and a static interaction matrix: jointly rescale both factor paths, their innovation scale and the interaction matrix (default `TRUE`). This preserves fitted values and the original posterior target, including all prior bounds. Set `FALSE` to retain the conditional updates alone.

marginal_move Run the joint observed-data updates and associated persistence, scale and translation moves for supported binary configurations (default `TRUE`); see Details.

binary_scale_move For `"binary"` fits (every mode, not with `dynamic_rho`): draw the factor by which the coefficients and the static additive effects are multiplied, exactly and with the latent array integrated out, every `binary_scale_every` sweeps (default `TRUE`); `FALSE` omits it. See Details.

binary_scale_every For `"binary"` fits: number of sweeps between draws of that factor (default 5).

level_move_every For binary unipartite and bipartite fits: number of sweeps between visits of the scale move (default 30).

level_move_tries For binary unipartite and bipartite fits: proposals per visit of

the scale move (default 8). Raising either this or the frequency buys a longer effective sample along the level / spread ridge at a cost per sweep; the visit pays for the factor draw and the latent redraw once, so more proposals per visit are cheaper than more visits.

dev_move For binary unipartite dynamic_ab fits: walk the second direction of the scale move, the per-period deviations together with sigma_ab (default TRUE). See Details.

dev_move_tries Proposals per visit along that direction (default 8).

actor_move For "poisson" and "ordinal" fits (unipartite, directed or symmetric, or bipartite; static or dynamic_ab effects; static coefficients or, for the level translations, dynamic_beta of kind "ar1" or "rw1"): run the translations of the levels with their latent rows and columns, and the further updates listed in Details (default TRUE). For binary fits (unipartite: directed or symmetric, with any additive margin and static coefficients or dynamic_beta of kind "ar1" or "rw1", incomplete panels included; or bipartite with an additive margin or static positions, with static coefficients or dynamic_beta of kind "ar1" or "rw1"): run the collapsed per-actor update (default TRUE). For "cbin" and "frn" fits (unipartite or bipartite, static or dynamic_ab levels, static coefficients or dynamic_beta of kind "ar1" or "rw1" on the unit period grid, with or without dynamic_beta_per_actor): run the receiver-level translation, with an intercept the level update, and for directed fits with dcor = TRUE the reciprocity update; the sender translation of a dynamic_ab fit runs under the same switch (default TRUE). With dynamic_uv, Poisson and binary fits also redraw the position paths (actor_move_paths). See Details.

chain_extensions Number of times a chain that is still moving at the end of the run is continued before the call is refused (default 3; 0 refuses at the first flag). See the convergence policy.

actor_move_every Number of sweeps between visits of the collapsed per-actor update (default 10), of the translations of the Poisson (default 2) and ordinal (default 10) families or of the receiver-level translation and the level, period and room updates of "cbin" and "frn" fits, and of the sender translation of their dynamic_ab fits (default 1; their reciprocity update runs on every tenth visit).

actor_move_rho For directed fits with dcor = TRUE, also update the dyadic correlation with the latent responses integrated out at each visit of the collapsed per-actor update (default TRUE); for "cbin" and "frn" fits it switches the reciprocity update.

actor_move_period For "cbin" and "frn" fits with a dynamic_beta intercept: also translate the intercept of the periods that the rows with room do not tie together, period by period (default TRUE; FALSE keeps the common translation of the level update only). See Details.

actor_move_room For "cbin" and "frn" fits with a dynamic_beta intercept: also move the intercept of each period that has rows with room, with the non-nominations of those rows integrated out (default TRUE). See Details.

beta_hyper_nc For dynamic_beta fits: run the interweaving step that moves the hyperparameters of the coefficient paths with the path (default TRUE);

FALSE omits it and keeps the draws at every sweep. See the dynamic_beta argument.

poisson_period_move For "poisson" fits with dynamic_beta: translate every period, and the whole path, of each dynamic coefficient with its latent slice at every sweep (default TRUE); FALSE omits them. See Details.

poisson_static_move For "poisson" fits with dynamic_beta and static coefficients: translate each static coefficient with the whole latent array every actor_move_every sweeps (default TRUE).

ordinal_scale_move For "ordinal" fits (unipartite and bipartite, static or dynamic coefficients and effects): draw, at every sweep, the scale of the whole state (the coefficients, the static additive effects and the latent array together, with the free cutpoints of ordinal_cutpoints = "explicit") and, with the default data-induced cutpoints, the shift and the scale of the latent array of each period alone, each exactly along a transformation that keeps every rank (default TRUE); FALSE omits them. See Details.

actor_hyper_nc For dynamic_beta_per_actor fits: run the interweaving step that moves the innovation variance and the persistence of the actor-specific paths with the paths (default TRUE); FALSE omits it and keeps the draws at every sweep.

actor_move_paths For Poisson and binary fits with moving positions (dynamic_uv = TRUE): redraw the whole position path of each actor at each visit (default TRUE); FALSE omits it and leaves the other updates of actor_move as they are. See Details.

ab_hyper_move For dynamic_ab fits with an additive margin (every family and mode): draw the level variances (their covariance for directed unipartite fits), the innovation scale and the persistence of the additive effects given their totals with the levels integrated out (default TRUE). See Details.

ab_hyper_move_tries Proposals per sweep of that update (default 8).

ab_period_move For "poisson" fits with dynamic_ab: move each actor-period effect together with the latent values of its row (column), in pairs when the effects of a period sum to zero, exactly and at every sweep (default TRUE). See Details.

uv_g_scale_move For bipartite fits with static positions: draw the scale split between the positions and the interaction matrix exactly (default TRUE). See Details.

als_start For directed unipartite dynamic_uv fits of the "ar1" or "t" kind (normal, Poisson or binary outcome): start the chain from a dynamic AR(1) ALS fit of the same model (default TRUE; see Details). FALSE keeps the cold start.

snap_kappa For dynamic_uv_kind = "snap": the sd of the position a reset lands on. Unset (the default), a reset is a fresh draw from the stationary distribution of the position paths, $\kappa^2 = \sigma_{uv}^2/(1-\rho_{uv}^2)$, recomputed every sweep, and the reset positions enter the rho_uv and sigma_uv draws as stationary draws. A positive number fixes the scale. A fixed scale well above the positions' spread makes a reset implausible for any ordinary jump; choose this scale in the units of the latent coordinates.

rho_beta_mean For dynamic_beta: Prior mean for the per-block AR(1) parameter on time-varying regression coefficients (default: 0.8). Closer to 1 = smoother evolution.

rho_beta_sd For dynamic_beta: Prior SD for the per-block AR(1) parameter (default: 0.15).

rho_beta_lower For dynamic_beta: Lower truncation bound on the AR(1) parameter (default: 0). Pass a negative value to allow negative autoregression.

rho_beta_upper For dynamic_beta: Upper truncation bound (default: 0.999). Closer to 1 admits near-unit-root behaviour.

sigma_beta_shape For dynamic_beta: shape parameter of the inverse-Gamma prior on the per-block innovation variance (default: 2). The conditional combines this proper prior with the coefficient path's quadratic form and effective dimension under its selected temporal law. With few transitions informing a coefficient block the posterior of the innovation variance stays near this prior.

sigma_beta_scale For dynamic_beta: scale parameter of the inverse-Gamma prior on the per-block innovation variance. It is set once before sampling from the response scale: the innovation covariance of the coefficient path is `sigma_beta^2 Lambda` with `Lambda` the g-prior scale matrix, so `sigma_beta` is the per-period innovation sd of a covariate's contribution to the linear predictor in the units of the response. The default prior mean innovation variance is 0.04 times the residual variance (the variance of the two-way centred outcome for the normal and poisson families, 1 for the probit and rank families, whose latent scale is fixed). It is reported as `fit$sigma_beta_prior_scale`. Set a value to override it.

sigma_beta_init For dynamic_beta: Initial value of the per-block innovation standard deviation (default: 0.25; `0.25 * sd(Y)` for the normal family). Affects mixing, not the stationary distribution.

beta0_mean For dynamic_beta: Mean of the Gaussian prior on the first observed coefficient state under AR(1) or RW1 (default: 0).

beta0_var For dynamic_beta: Variance of the Gaussian prior on the first observed coefficient state under AR(1) or RW1 (default: 10; `10 * var(Y)` for the normal family, in the unit of the response). This prior is independent of the sampled evolution parameters. RW2 uses its proper difference-precision prior and Matern 3/2 its covariance kernel over all observed periods instead.

Common usage: prior = list(Sab0 = diag(c(1, 1)), eta0 = 10) for stronger shrinkage, or prior = list(rho_uv_mean = 0.95) for higher temporal persistence, or prior = list(rho_beta_mean = 0.95, sigma_beta_scale = 0.01) for very smooth time-varying coefficients with tight innovations.

g optional scalar or vector for g-prior on regression coefficients. Default is `n^2`, where `n` is the number of actors (`nA` for a bipartite fit). The prior of the normal family scales with the response, for every unit of `Y`: a static unipartite fit has the coefficient precision `sum_t(crossprod(X_t)) / (T g)`, whatever the residual variance, and takes `n^2 * var(Y)`; the coefficients of a bipartite fit, and the static coefficients next to dynamic ones, have the prior `N(0, g s2)`, whose `s2` already carries the unit, and take `n^2`. The g-prior controls the variance of regression

	coefficients: larger values allow for larger coefficient values. Can be a vector of length p for parameter-specific control.
seed	random seed for the MCMC sampler (default 6886). The sampler is seeded internally with this value, so results are reproducible by default and an external `set.seed()` call has *no* effect on the chain – pass a different `seed` here to vary the draws (e.g. when running multiple chains). The caller's `.Random.seed` is restored on exit, so fitting never perturbs your RNG stream. Changing `nscan` or `burn` changes the whole chain, not only its length, because the sweep permutations are drawn up front; to lengthen a finished chain use `lame_resume`.
nscan	number of iterations of the Markov chain (beyond burn-in)
burn	burn in for the Markov chain
odens	output density for the Markov chain
plot	logical: plot results while running?
verbose	logical: print progress while running? Default FALSE.
gof	logical: calculate goodness of fit statistics?
start_vals	list of parameter starting values for the MCMC chain. Tempered fits accept their own `fit$start_vals` replica matrix as a warm start; use `lame_resume` for exact continuation. `dynamic_uv` fits of the `"ar1"` kind start from a dynamic ALS fit by default when this is `NULL` (see Details). Use `als_start_vals` to start MCMC from an ALS fit, or the `start_vals` element of a previous fit to continue from its final state. Recognised slots are `Z`, `beta`, `a`, `b`, `U`, `V`, `rho`, `s2`, `Sab` and, for dynamic fits, the scalars `rho_uv`, `sigma_uv`, `rho_ab` and `sigma_ab`; bipartite fits also take `G`, use a slot only when its shape matches the panel and warn about the ones they skip. The `rho_uv` and `rho_ab` values that `als_start_vals` emits are the penalty settings of the ALS fit rather than estimates (for an empirical-Bayes fit, the default for normal, binary and Poisson panels, they are estimates); they only seed the chain.
periodic_save	logical: indicating whether to periodically save MCMC results
out_file	character vector indicating name and path in which file should be stored if periodic_save is selected. For example, on an Apple OS out_file="~/Desktop/ameFit.rda".
save_interval	quantile interval indicating when to save during the post-burn-in period.
model.name	optional string for model selection output
save_log_lik	one of `FALSE` (default), `TRUE`, or `"chunked"`. When `TRUE`, stores the per-iteration pointwise log-likelihood matrix on `fit$log_lik` (an `[n_stored, n_obs]` double matrix). When `"chunked"`, streams the log-lik values to per-column-chunk binary files under `log_lik_path` so the in-memory cost during MCMC is just one chunk's width; the chunks are recovered later via `read_log_lik(fit)`. Required for `loo::loo(fit)` and `loo::waic(fit)`.
posterior_opts	optional list of posterior draw-storage options, either a plain `list()` or one built with `posterior_options` (which spells the position option `save_UV`). Recognised names (unknown names trigger a warning): `save_UV`, `save_UV_draws`, `save_ab`, `thin_UV`, `thin_ab`. `save_UV = TRUE` and `save_UV_draws = TRUE` are aliases on this path. With `dynamic_uv = TRUE` they store the per-iteration dynamic latent-position cubes at every stored (`odens`) iteration on `fit$U_draws` /

fit$V_draws, each an [actor, dim, period, draw] array. On a static fit with a positive multiplicative rank they store the per-iteration latent-position matrices on fit$U_samples / fit$V_samples ([actor, dim, draw] arrays, as ame does). Bipartite fits with positive latent ranks additionally store the interaction-matrix draws on fit$G_samples ([R_row, R_col, draw]). save_ab = TRUE stores identified additive-effect draws in a_samples / b_samples (actor by draw) for static effects, or a_draws / b_draws (actor by period by draw) for dynamic effects, with actor-level draws in a_level_samples / b_level_samples. thin_ab thins the retained coefficient draws further; ab_draw_index records their matching row indices. Compact joint terminal states are always retained on fit$forecast_state for posterior forecasts, even when full actor and position trajectories are not requested. thin_UV remains accepted for compatibility but is not applied here (odens already thins the stored chain). Requesting latent draw storage with a zero multiplicative rank warns and stores nothing. Default NULL.

log_lik_path
: directory to write log-lik chunks to when save_log_lik = "chunked". Default NULL creates a session-scoped subdirectory under tempdir().

log_lik_chunk_size
: column-width of each on-disk chunk when save_log_lik = "chunked". Larger chunks mean fewer files (and one readBin per chunk on read) but more memory during MCMC. Default 10000L.

keep_snap_draws
: controls storage of post-burn-in MCMC snap-shift indicators when dynamic_uv = TRUE and dynamic_uv_kind = "snap". "none" keeps only the posterior mean snap probabilities. "summary" stores cell-level count, mean, and variance summaries. "draws" stores retained binary indicator arrays as fit$snap_draws and, for directed fits, fit$snap_draws_v, with dimensions draw x actor x time and first period set to NA. "chunked" currently stores the same arrays in memory and labels the storage mode for downstream tooling.

freeze_call
: logical: if TRUE, store a snapshot of the evaluated Y, Xdyad, Xrow, Xcol on fit$data_snapshot so that a later update(fit, ...) refits against the same data even if the caller has mutated those objects in their workspace. Memory cost equals the size of the data; default FALSE.

dynamic_beta_pool
: one of "none" (default), "rho", "sigma", or "both". When two or more coefficient blocks are dynamic, non-"none" values update shared hyperparameters across blocks for the selected persistence and / or innovation-scale component. Persistence pooling uses a Gaussian hierarchy on the logit of (rho - rho_beta_lower) / (rho_beta_upper - rho_beta_lower), with a normal mean prior and inverse-Gamma variance prior. Its current mean and scale are retained in checkpoint state. With a single dynamic block the value is recorded but there is no other block to pool against. On method = "als" the empirical-Bayes estimator honours it: "sigma" shares the innovation scales across the dynamic blocks, "rho" pools the first-order persistences through the same logit-normal hierarchy as the sampler (a random-walk block has none to pool), "both" does both; the penalised routes name it in the warning about ignored arguments.

dynamic_beta_per_actor
: optional, one of NULL (default), "row", or "col". When non-NULL, fits actor-specific time-varying deviations for one dyadic covariate on the selected mar-

gin. The default `per_actor_identifiability = "center"` keeps the population coefficient in `coef(fit)` and enforces a per-period sum-to-zero constraint on the actor deviations. Their innovation variance defaults to an inverse-Gamma(2, 1) prior (scale `var(Y)` for the normal family: the unit of the response), configurable through `prior$sigma_actor_shape` and `prior$sigma_actor_scale`. The transformed persistence $(1+\rho)/2$ defaults to a Beta(2, 2) prior, with shapes `prior$rho_actor_shape1` and `prior$rho_actor_shape2`, restricted to $-0.99 < \rho < 0.99$, with a stationary initial state. Priors act on the centered subspace with one fewer dimension than the actor count. All four hyperparameters must be finite and positive. Choose the innovation scale for the outcome and covariate units. Stored fits record these choices in `per_actor_prior`. Draws are stored in `fit$THETA_ACTOR` when retained; summary mode stores `fit$theta_actor_mean` and `fit$theta_actor_sd`. The innovation variance and the persistence are drawn from their exact conditionals at every sweep from the first one, and an interweaving step moves them together with the paths (`prior$actor_hyper_nc`). Where each actor's data say little about its path, which is the usual case, the draws every 20th sweep of earlier versions left them with 18 to 75 effective draws in 2,000 and the draws at every sweep with 220 to 900; the step adds a factor of 1.1 to 2.4.

`per_actor_covariate_idx`
: positive integer; index into the dyadic covariate cube to slope on for the per-actor extension. Default `1L`.

`per_actor_identifiability`
: one of `"center"` (default), `"exact_center"` or `"drop_population"`. `"center"` preserves the population coefficient and constrains the per-actor deviations to sum to zero per period via pairwise contrast FFBS. `"exact_center"` draws each actor's path from its own conditional AR(1) posterior and conditions on all period-wise sum-to-zero constraints using the complete temporal covariance. Dyadic reciprocity couples actors; that case uses a joint Gaussian draw on the constrained subspace, at a higher cost per sweep. Actor-specific slopes require directed or bipartite data and unit-spaced periods. `"drop_population"` is reserved and currently falls back to `"center"` with a warning.

`keep_per_actor`
: one of `"auto"` (default; full draws when memory cost < 250 MB else streaming summary), `"draws"` (full `[n_iter, n_actors, T]` cube), `"summary"` (streaming posterior mean + sd only), or `"none"` (only hyperparameter chains).

`time_index`
: optional numeric vector of length T giving observation times. Default `NULL` treats periods as equally spaced. Strictly-increasing values are required when supplied. Unequal gaps scale the AR(1) / RW1 transition variance and are also used by the RW2 and Matern 3/2 dynamic-beta precision builders.

`period_exposure`
: optional non-negative numeric vector of length T giving period-level exposure offsets. **Wired for Poisson:** when supplied with any value != 1, the Poisson observation likelihood becomes $Y_{ij,t} \sim \text{Poisson}(e_t \cdot \exp(Z_{ij,t}))$, while the latent Z retains its existing semantic of "unexposed log-rate". For other families, non-trivial `period_exposure` is an error (rescale `Y` or use a covariate offset). `NULL` (default) or all-ones uses the unscaled likelihood path. Zero exposure is supported when all observed counts in that period are zero; its observed log-likelihood is zero.

`max_seconds` optional positive scalar; if the MCMC wall-clock time exceeds this many seconds, the chain terminates cleanly and `fit$terminated_early` is set to `TRUE`. The draw arrays hold the draws stored until then. When the time runs out before the first stored draw, `fit$no_draws_stored` is `TRUE`, every draw array (`BETA`, `VC` and the others) has no draws along its draw dimension, the posterior means are `NA`, `print()` says so, and `coef()`, `summary()` and the other methods that summarise draws stop with an error of class `lame_no_draws`; `lame_resume(fit, nscan_more = N, max_seconds = Inf)` continues the chains.

`checkpoint_path`
optional file path. When set, the chain periodically writes `BETA`, `VC`, the full sampler state, seed, completed iteration count, remaining burn-in and original call to this file. Use `lame_resume` or pass the same path back as `resume_from = path` to continue.

`checkpoint_every`
positive integer; iterations between checkpoint writes. Default `100L`.

`log_lik_method` one of `"observed_exact"` (default), `"observed_ghk"`, or `"augmented"`. Observed marginal scores are supported for normal, binary, Poisson-lognormal and explicit-cutpoint ordinal fits. Poisson integration uses an adaptive log-integral with concavity bounds on omitted tails. Capped nominations, fixed ranks and data-induced ordinal models do not have the cellwise observed likelihood claimed by `"observed_exact"`; such requests fail explicitly. The existing `"observed_ghk"` rank estimators are approximate diagnostics, and `"augmented"` stores Gaussian-on-Z latent diagnostics; `loo()` and `waic()` refuse both with an error of class `"lame_loo_factorization_unsupported"`, and `read_log_lik()` returns the scores. Even correct cell marginal scores require independent residual cells, or unique unordered dyads in symmetric fits, before LOO or WAIC can use them. Directed correlated cell marginals require joint dyad factors, which are not supplied by this interface. Inspect `fit$log_lik_contract`. `"augmented"` uses the augmented-data Gaussian-on-Z contribution. Note the GHK cost grows exponentially with the per-row dimension D (the number of observed dyads in a sender row): the per-row Monte Carlo budget escalates to `min(2048, max(64, 8 * 2^D))` draws across a bank of 8 randomized Halton sequences, once per stored draw, so `"observed_ghk"` on a unipartite `"frn"` fit with more than roughly 12 actors (where $D = n - 1$) is expensive; rows with $D > 15$ fall back to a fast pairwise-normal approximation. Power users can fix the per-row budget exactly via `prior$ghk_n_mc` (a single positive integer), which both raises the 2048 ceiling and lowers the `8 * 2^D` accuracy floor; lower budgets increase the downward Jensen bias of the log-lik estimate.

`ordinal_cutpoints`
character: cutpoint convention for `family = "ordinal"`. `"data_induced"` (default) uses the data-induced cutpoints; `"explicit"` samples ordered cutpoints with the first fixed at zero and unit latent residual variance. Independent directed and symmetric models use the Cowles (1996) marginal Metropolis-Hastings update. Directed models with `dcor = TRUE` use cutpoint Gibbs updates conditional on the correlated latent observations. Explicit cutpoints are supported for unipartite models; predictions retain the original numeric category labels and use each saved cutpoint draw. Ignored for other families.

`method` character: `"mcmc"` (default, the Bayesian MCMC fit) or `"als"` (the fast, MCMC-free point-estimation path). With no dynamic arguments, `method = "als"` uses the pooled-static `lame_als` estimator. For normal, binary, and Poisson panels, `method = "als"` also supports a dynamic estimator (an empirical-Bayes fit for normal, binary and Poisson panels, or with `bias_correct = "none"` the penalized point estimate; see `bias_correct`) for `dynamic_ab`, selected `dynamic_beta`, smooth AR(1) `dynamic_uv` in directed, symmetric, and bipartite panels, and Student-t `dynamic_uv` in directed and symmetric panels (bipartite t requests are refused). `dynamic_uv_kind = "snap"` is refused on this route (use `method = "mcmc"`). For a normal, binary or Poisson panel the dynamic route is by default an empirical-Bayes fit (see `bias_correct`): `rho_ab`, `sigma_ab`, `rho_uv`, `sigma_uv`, `rho_beta` and `sigma_beta` are then estimates under the hyperpriors of `lame()` (the variance components and persistences by EM, the stationary variance of moving positions by a profile of the marginal objective), not the prior means and penalty-implied scales of the penalised fit. The uncorrected penalised fit (`bias_correct = "none"` or `"jackknife"`) reports `rho_uv`, `rho_ab` and `rho_beta` as the AR(1) persistences its smoothness penalties used, which are the prior means unless `prior` sets them, not as estimates, and `sigma_uv`, `sigma_ab` and `sigma_beta` as the innovation scales implied by the fitted paths, which the penalty shrinks below the posterior scale. Either can seed a warm-started MCMC run via `als_start_vals`. Under `dynamic_ab` the reported `VC` entries `va` and `vb` are the variances of the actor *levels*, as in an MCMC fit: the empirical-Bayes fit estimates them by EM (`fit$eb$sab`), and the penalised fits take them from the cross-period covariance profile of the fitted paths, whose lag-k means follow $v_a + \tau^2\rho^k$, so the AR(1) deviation and the per-period estimation noise both decay out of the reported level variance. The mean per-period variance of the paths – level plus deviation – is kept as `fit$vc_period`, and the uncorrected variance of the time-averaged paths as `fit$vc_raw`. For a penalised normal-family fit (`bias_correct = "none"`), `ame_als_bootstrap` redraws the actor paths (level plus AR(1) deviation) in each replicate, so its intervals for `va` and `vb` carry the uncertainty of that split. Node-covariate coefficients use the same orthogonal additive-effect decomposition as `lame_als`; dynamic node coefficients use period-specific node values when selected by `dynamic_beta`, while static node coefficients use per-actor means. Named changing-composition panels are aligned to the union actor set; actor-entry gaps break the smoothing penalties of the penalised fits, and the empirical-Bayes fit treats an absent actor-period as a free AR state. Posterior draws and rank/censored families are not handled by the ALS routes. Dynamic ALS prediction evaluates the fitted periods, optionally with replacement dyadic covariates; `predict(fit, h = ...)` forecasting is not implemented and a positive horizon is refused.

`als_stability` character (only used when `method = "als"` and a dynamic ALS path is selected): `"none"` (default), `"quick"`, or `"validation"`. The non-`"none"` presets rerun the dynamic ALS fit from jittered starts and attach `fit$stability`, which reports fitted-surface and coefficient-path differences across starts.

`als_max_iter` positive integer (only used when `method = "als"`): maximum iterations for the selected ALS solver. Default `1000`. Dynamic binary and count fits count IRLS updates, each with at most 40 block-coordinate sweeps. Stalled Gaussian bipartite and directed Poisson fits with AR position paths and static coefficients

also allow this many joint gradient-refinement iterations (with static interactions in the bipartite case). They retain that result only when the same objective decreases and the likelihood-score convergence criterion passes. The dynamic empirical-Bayes estimator needs up to a few hundred passes on a weakly identified panel; the Gaussian-transition fit of a 40 x 10 panel takes about 150 passes at the scale its search settles on. A dynamic fit (empirical-Bayes or penalised) that has not met its convergence criterion in this many iterations is an error of class `"lame_als_convergence"`; its field `fit` holds the last iterate. The sweep limit of a penalised normal fit is raised once, to four times this value, before the fit is refused. A static fit that does not converge is refused in the same way, without a `fit` field (see `max_iter` in `ame_als`).

`als_tol` optional positive scalar (only used when `method = "als"`): convergence tolerance for the ALS objective and fitted values. `NULL` keeps each ALS estimator's built-in default. The dynamic empirical-Bayes estimator stops when every block's scaled score is below `sqrt(als_tol)` and, for moving positions, the objective has settled to within `als_tol` deviance units of the fixed point; for a Poisson fit the estimated distance of the noise variance `ve` to the limit of its iteration must be below the threshold as well.

`als_lowrank_method`
optional character (only used when `method = "als"`): inner solver for the low-rank factor block, `"mm"`, `"als"`, or `"hybrid"`. `NULL` (default) keeps each route's own default – `"mm"` on the static `lame_als` route, `"hybrid"` on the dynamic route. The dynamic empirical-Bayes estimator (the default for normal, binary and Poisson panels) solves the factors jointly with the other effects by Gauss-Newton and warns that the argument does not apply; `bias_correct = "none"` uses the alternating low-rank solvers.

`als_linear_solver`
optional character (only used when `method = "als"` with no dynamic arguments): linear solver for the static block updates, `"eigen"`, `"qr"`, or `"auto"`. `NULL` (default) keeps `lame_als`'s default (`"eigen"`). The dynamic ALS route uses its own path solvers and aborts if a value is given.

`als_multistart` optional character (only used when `method = "als"` with no dynamic arguments, and `R > 0`): `"none"`, `"cheap"`, or `"full"`, forwarded to `lame_als`. `NULL` (default) defers to `multistart`. Supplying both uses `als_multistart`.

`multistart` character (only used when `method = "als"` with no dynamic arguments, and `R > 0`): `"none"` (default), `"cheap"` (4 starts), or `"full"` (8 starts). Forwarded to `lame_als`, which reruns the block-coordinate fit from several starts and keeps the one with the lowest objective. The dynamic ALS route has no multi-start and ignores this argument with a warning; use `als_stability` there instead.

`bias_correct` character or `NULL` (only used when `method = "als"`): how the ALS point estimate handles the finite-sample (incidental-parameter) bias of a fixed-effects fit. For the static ALS point estimate it is forwarded to `lame_als`. `NULL` (default) fits a normal, binary or count static model by the empirical-Bayes estimator (`"eb"`: a Gaussian prior on the actor effects with its covariance estimated from the data, Laplace standard errors from `vcov()`, the exact Gaussian cell for a normal response and for counts the Poisson-lognormal model that `lame()` samples; see `bias_correct` in `ame_als`), for directed, bipartite and undirected networks,

with or without a latent rank. "eb" asks for the empirical-Bayes estimator explicitly.

A normal, binary or Poisson fit with dynamic_ab, dynamic_beta or dynamic_uv (of the "ar1" or "t" kind, with static or moving latent positions) and bias_correct = NULL is fitted by the dynamic empirical-Bayes estimator of the model lame() samples: the actor levels and zero-mean AR(1) deviation paths are random effects and the coefficients have a Gaussian prior; the variance of the cell-level noise (normal and Poisson), the covariance of the actor levels, the persistence and innovation scale of the additive paths (and of moving coefficients and positions) are estimated by EM under the lame() hyperpriors; the coefficients have Laplace standard errors (vcov(), confint(), fitebse; exact for a normal response); static factors are the variational posterior of ame_als() (options(lame.eb_scale) selects the rule that sets the scale of their prior, "cavity" by default) and moving positions have a mean-field Gaussian posterior whose product uncertainty is carried through the cell likelihood; and a directed panel gets the dyadic correlation (dcor).

A normal response has an exact Gaussian cell: the posterior of the coefficients, the levels and the paths at the hyperparameters is exact, the standard errors are the marginal ones given the hyperparameters, the dyadic correlation is an exact bivariate normal pair, and the reported intercept is the population intercept. A Poisson response is the Poisson-lognormal model: the correlation of a directed panel is taken at the fitted state, with the information of the pairs in the standard errors; the paths and levels are reported at their first-order posterior mean, the linear predictor (EZ) and the intercept are on the latent scale, and fitted() and predict() return exp(EZ + ve / 2).

A dynamic fit that did not meet its convergence criterion, or whose moving positions have no identified scale, is not returned: the call stops with an error of class "lame_als_convergence", "lame_dynamic_uv_scale_edge" or "lame_dynamic_uv_scale_unsettled" whose field fit holds the last iterate.

"none" returns the fixed-effects fit of the earlier releases at any size (on the dynamic route the penalised fit with fixed smoothness penalties). The penalised dynamic fit descends at every sweep and is returned only if it met its first-order criterion; the static factors of a binary or Poisson fit carry the level prior of the dynamic factors (precision 1). "jackknife" requires the split-sample correction of a static fixed-effects fit. On the dynamic route it is available for directed Poisson panels with static coefficients, at least four actors, and rank zero or AR(1) positions, and corrects the penalised fit for latent count variance and reciprocal covariance using two complementary dyad histories. The correction combines natural variances and covariances as twice the full estimate minus the observation-fraction-weighted halves. It preserves response means by shifting the latent intercept by half the variance change. Unrestricted moments, boundary projections and every half-fit attempt are retained in fit$bias_correct; fit$raw holds the original summaries. Each half uses als_max_iter, with one warm retry at five times that budget if necessary. All final halves must pass the same likelihood-score criterion. The correction removes the first-order term of the finite-sample bias of the latent variance and reciprocal covariance, the term that falls as the inverse of the number of observed cells per actor. Other dynamic specifications refuse an explicit "jackknife" request (class "lame_jk_split").

"bootstrap", the parametric-bootstrap correction, is for static binary fits.

On the dynamic route the empirical-Bayes estimator estimates the persistences and innovation scales, so it refuses the smoothness penalties of the penalised fit (prior$lambda_ab_als, prior$lambda_beta_als, prior$lambda_uv_als) with an error of class "lame_dynamic_eb_unsupported" (which also has class "lame_als_eb"), and a bootstrap (bootstrap > 0) with an error of class "lame_dynamic_eb_unsupported"; use bias_correct = "none" to fix the penalties. A bootstrap of a penalised dynamic fit exists for the normal family only.

bootstrap integer (only used when method = "als"): number of bootstrap replicates. 0 (default) skips the bootstrap; N >= 30 runs N replicates and attaches the result so that confint returns bootstrap intervals (a smaller positive N is an error, on the static and the dynamic route alike, raised before anything is fitted). The dynamic route bootstraps the penalised normal-family fit (bias_correct = "none") only. Each replicate is drawn as ame_als_bootstrap draws it (the additive paths redrawn, then the residual) and refit from the fitted trajectories; a replicate that has not converged within als_max_iter iterations is retried at four times that budget and otherwise dropped, never kept; more than 5 percent of the replicates dropped is an error of class "lame_als_bootstrap_convergence", as is a bootstrap of a fit that has not converged. A binary static network fitted with bias_correct = "none" has no bootstrap (an error of class "lame_als_bootstrap_bias"), nor has a penalised binary or Poisson dynamic fit (class "lame_als_bootstrap_unsupported").

bootstrap_type character (only used when method = "als"): "parametric" (default) or "block". The block scheme needs at least ten resampling blocks (ceiling(T / bootstrap_block_length)); on a shorter panel, and on any dynamic ALS fit, it warns and uses the parametric scheme instead. The block scheme returns no standard error or interval for the parameters that resampling time slices cannot vary (see ame_als_bootstrap); the parametric scheme gives them. A dynamic ALS fit has a bootstrap only when it is a penalised normal fit (bias_correct = "none"). The default dynamic empirical-Bayes estimator of every family refuses it with an error of class "lame_dynamic_eb_unsupported", and vcov(), confint() and summary() carry its Laplace standard errors; a penalised binary or Poisson fit refuses it too.

bootstrap_block_length integer: block length for the block bootstrap.

bootstrap_seed optional integer seed for the bootstrap.

chains integer: the number of independently seeded chains (default 2, or getOption("lame.chains")). Chain k runs with seed seed + 1000 * (k - 1). With chains > 1 the chains are checked, compared and pooled (see the convergence policy); chains = 1 runs one chain and the drift check only. ALS fits and sampler = "tempered" run one chain.

cores integer or NULL: workers for the chains. The chains run forked on Unix, at most one worker per chain; NULL uses getOption("lame.cores") if set and otherwise all cores but one. Windows runs the chains one after the other.

resume_from optional path to a checkpoint file produced by a previous lame(..., checkpoint_path = path) call. When non-NULL, lame() short-circuits all other input parsing and delegates to lame_resume with the user-supplied overrides forwarded. Pass nscan = K to request K additional post-burn sweeps on the continuation. Supply

checkpoint_path and max_seconds again to checkpoint or limit the continuation. Unfinished burn-in is completed before any new posterior draws are stored. Default NULL.

print — Deprecated. Use verbose instead.

sampler — MCMC transition method. Default "gibbs" retains the established sampler. "tempered" uses exact observed-probit replica exchange for symmetric binary networks with positive static rank, a static intercept and optional symmetric dyadic covariates, and shared dynamic additive effects on a common actor grid. Its default burn-in is 15,000 sweeps and default post-burn length is 200,000 sweeps. See Details for its explicit latent-coordinate prior and supported options.

Details

This command provides posterior inference for parameters in AME models of longitudinal relational data, assuming one of six possible data types/models. The model supports both unipartite networks (square adjacency matrices) and bipartite networks (rectangular adjacency matrices with distinct row and column node sets) across multiple time points.

Tempered symmetric binary sampling:

Set sampler = "tempered" for a common actor grid with family = "binary", symmetric = TRUE, dynamic_ab = TRUE, positive static rank, a static intercept and optional symmetric dyadic covariates. At least four actors and two equally spaced periods are required. The observed likelihood contains each unordered dyad once:

$$P(Y_{ijt} = 1) = \Phi(x_{ijt}^T \beta + a_{it} + a_{jt} + u_i^T \Lambda u_j).$$

Shared actor effects comprise a zero-sum level and a zero-sum stationary Gaussian AR(1) deviation. Their level variance has an inverse-gamma prior with shape etaab/2 and scale etaab * Sab0[1,1]/2; the innovation variance has shape sigma_ab_shape and scale sigma_ab_scale. Persistence has its specified normal prior truncated to $(-0.99, 0.99)$. Coefficients retain the panel-design Gaussian prior.

prior$uv_prior = "hierarchical" retains the static symmetric factor prior: signed weights are independent standard normals, and each raw coordinate column has variance $v_k \sim IG(1, 0.5)$. This is the default for the tempered sampler. The explicitly selected alternative prior$uv_prior = "unit" fixes every coordinate variance at one; it changes the model's prior, not just its sampling algorithm. Raw factors are centered for reporting, with compensating actor-level and intercept shifts that preserve every joint predictor.

The sampler uses bounded Langevin proposals with their complete forward and reverse Gaussian densities, a fixed ladder of 21 likelihood powers, replica exchanges, prior refreshes at power zero, and signed-weight reflections. Step sizes adapt only during burn-in. fit$tempering records transition counts, step sizes and replica sign traces; convergence diagnostics are computed from the stored draws of independently seeded fits.

This route accepts only its documented factor and additive prior settings; Sab0 must be a positive multiple of the two by two identity and ab_var_cap must be infinite. Nodal covariates, changing actor sets, time-varying coefficients/factors, exposure, and chunked likelihood output are not supported. save_log_lik = TRUE stores one column per observed unordered dyad-period, also the unit counted by nobs(). Posterior storage uses the same posterior_opts interface, and compact

joint terminal states are always retained for forecasts. LFO training windows need at least two periods. Tempered checkpoints restore every replica, the frozen model/prior and both random-number streams; only sampling length, output and time-limit settings can change on continuation. Built-in RNGs with serializable state are supported; Box-Muller and user-supplied RNGs are refused with an error.

Dynamic Effects Implementation:

The dynamic_uv and dynamic_ab parameters enable time-varying latent representations through autoregressive processes. These extensions are particularly useful for understanding how network structure evolves over time.

Dynamic Multiplicative Effects (dynamic_uv=TRUE):

The latent factors U and V evolve according to the transition selected by `dynamic_uv_kind`. The default `"ar1"` model uses Gaussian AR(1) drift:

$$U_{i,k,t} = \rho_{uv} U_{i,k,t-1} + \epsilon_{i,k,t}$$

where $\epsilon_{i,k,t} \sim N(0, \sigma_{uv}^2)$, i indexes actors, k indexes latent dimensions, and t indexes time. The parameter ρ_{uv} controls temporal persistence (values near 1 indicate slow evolution). This captures time-varying homophily, latent community structure, and transitivity dynamics. The `"snap"` transition uses a mixture of AR(1) drift and discontinuous reset transitions, while `"t"` uses heavy-tailed Student-t innovations represented by local transition scales. On the MCMC path, snap and t transitions are supported for directed unipartite models. With `method = "als"`, Student-t dynamic UV is supported for directed and symmetric normal, binary, and Poisson panels, including named panels where actors enter or exit; the snap kind has no ALS route.

Key references:

- Sewell & Chen (2015): Introduced dynamic latent space models with actor-specific evolution rates
- Durante & Dunson (2014): Nonparametric Bayesian approach allowing flexible evolution of network structure
- Hoff (2011): Hierarchical multilinear models providing theoretical foundation for temporal dependencies

Dynamic Additive Effects (dynamic_ab=TRUE): Each sender (a) and receiver (b) effect is a per-actor level plus a zero-mean AR(1) deviation:

$$a_{i,t} = \mu_i^a + d_{i,t}, \quad d_{i,t} = \rho_{ab} d_{i,t-1} + \epsilon_{i,t}$$

$$b_{j,t} = \mu_j^b + e_{j,t}, \quad e_{j,t} = \rho_{ab} e_{j,t-1} + \eta_{j,t}$$

where $\epsilon_{i,t}, \eta_{j,t} \sim N(0, \sigma_{ab}^2)$. The variance of the levels across actors is the reported `va` / `vb`; ρ_{ab} and σ_{ab} describe the deviations. This models time-varying individual activity levels (outdegree) and popularity (indegree).

Identified reporting: Stored static factor draws are centered only when both resolved actor margins are enabled. The removed row, column and grand-mean components of UGV' are transferred to the actor effects and intercept; for a symmetric shared actor without an intercept, the grand constant is divided equally between its two contributions. Otherwise, the saved draws retain their sampled coordinates because disabled effects cannot absorb the offsets. The reported map preserves the linear predictor and leaves the sampler state unchanged. Dynamic-coordinate reporting follows the supported fitted route; inspect fitted products and predictions when comparing latent geometries.

Dynamic additive effects are useful when sender activity and receiver popularity change over time rather than staying fixed across the whole panel. A `dynamic_ab` fit returns the posterior-mean paths on `fit$a_dynamic` / `fit$b_dynamic` (actor x period) and their per-period posterior standard deviations on `fit$a_dynamic_sd` / `fit$b_dynamic_sd`; `ab_plot(fit, plot_type = "ribbon")` uses the latter to draw a credible band around each actor's path.

Prior Specification for Dynamic Parameters:

- $\rho_{uv}, \rho_{ab} \sim TruncNormal(mean, sd, -0.99, 0.99)$: Ensures stationarity of AR(1) process
- $\sigma^2_{uv}, \sigma^2_{ab} \sim InverseGamma(shape, scale)$: Controls innovation variance
- Default priors (ρ_{uv} mean=0.9, ρ_{ab} mean=0.8) favor smooth evolution
- Adjust rho_*_mean closer to 1 for slower evolution, closer to 0 for more rapid changes

Computational Considerations:

- Dynamic effects add path updates; runtime depends on actor count, rank and number of periods
- Memory usage scales as O(n*R*T) for dynamic_uv, O(n*T) for dynamic_ab
- C++ implementation is substantially faster than a pure R loop
- Convergence diagnostics: Monitor rho and sigma parameters carefully
- Effective sample sizes typically lower due to temporal correlation
- A chain that is still moving when the run ends is continued, and refused if it keeps moving (see the convergence policy below)

Chains and convergence policy: `lame()` runs `chains` chains (default 2; `options(lame.chains = k)` sets another default). Chain k is seeded with `seed + 1000 * (k - 1)`, so `chains = 1` is the first chain of a two-chain fit. On Unix the chains run forked, on at most `cores` workers (all cores but one by default); on Windows they run one after the other. The draws do not depend on the number of workers. The result is the pooled fit of `combine_ame_chains`, with `fit$chains`, `fit$chain_seeds`, `fit$rhat` (the R-hat of the monitored set), the state each chain stopped in, and the `chain_indicator` that `as_draws()` and `trace_plot` use. Two chains cost twice the CPU time and about the same wall time on a multicore machine.

At the end of the post-burn run every chain is checked, and the chains are compared with each other. Within a chain, the first and last thirds of the stored draws of the intercept (or first coefficient), `va`, `vb`, `ve`, `rho`, `rho_ab`, `sigma_ab`, `rho_uv`, `sigma_uv` and `pi_snap` are compared with batch-means standard errors (ten batches per third); a quantity is flagged when the two means differ by more than 3 standard errors and by more than half a posterior standard deviation of the whole series. Between chains, the split rank-normalised R-hat (Vehtari et al., 2021) of the monitored set is compared with 1.05: every regression coefficient (each period's under `dynamic_beta`), `va`, `vb`, `cab`, `ve`, `rho`, the dynamic hyperparameters, and summaries of the additive and latent structure recorded at every stored draw (the rms of `a` and `b`, the largest actor effect, the rms of the latent product UGV', the norm of `G` of a bipartite fit and the split of scale between `U` and `V` of a directed fit, per period for dynamic fits). The dyadic products are not monitored. Quantities a fit reports as not estimated are left out, and a run with fewer than 60 stored draws per chain is not checked. The gate is computed on the draws the returned fit holds (the last block of every chain). `fit$diagnostics$rhat`, from `compute_mcmc_diagnostics`, is the same statistic of the same draws and equals `fit$rhat` for every monitored coefficient and variance component.

When a chain drifts or an R-hat exceeds 1.05, every chain is extended by a block of `nscan` sweeps from its own final state, as `lame_resume` does (the seeds advance by the sweeps already run); the flagged block counts as extra burn-in, its draws are discarded, and both checks run again. Up to `prior$chain_extensions` extensions are made (default 3, so at most four times `nscan` post-burn sweeps per chain). The returned fit has the same number of stored draws as a fit that needed no extension, and `fit$auto_extend` records the extensions, the sweeps discarded and the quantities that triggered each. If the last block is still flagged, `lame()` stops with an error of class `lame_chain_unsettled`: its `fits` field holds the chains, `fit` the pooled fit of the last block, `rhat` the monitored table and `moving` the offending quantities, and `lame_resume(e$fit, nscan = nscan)` continues every chain (`lame_resume(e$fits[[k]], nscan = nscan)` one). `prior = list(chain_extensions = 0)` refuses at the first flag; `chains = 1` runs one chain with the drift check only. There is no mode that returns a flagged fit.

Measured on chains that were already stationary (112 blocks of 400 draws from 28 simulated configurations, two chains each), the largest monitored R-hat had a median of 1.007 and a maximum of 1.040, and no block was flagged by either check (the 95 percent upper bound on the false-extension rate is 3.2 percent; three consecutive false flags have probability below 4×10^{-5}). Measured on the 13 simulated configurations of the replication study that failed a two-chain R-hat gate in earlier builds (four pairs of seeds each, 52 two-chain fits at the defaults, `burn = 500`, `nscan = 10000`, `odens = 25`), the policy settled 47 fits as run, extended 5 (four once, one twice) and refused none. 47 of the 52 pass the book's gate, an R-hat below 1.05 on every identified parameter including every dyadic product; the other five sit at R-hat 1.04 to 1.08 on dyadic products or one actor effect, which the monitored set leaves out.

The dyadic products are left out because their largest R-hat crosses 1.05 on settled chains. A fit has about 10^4 of them, and on stationary chains of 17 simulated configurations with a latent product (68 two-chain blocks) the largest product R-hat of a block reached 1.055; settled ordinal directed fits with `dynamic_uv` crossed 1.05 in 2 of their 4 blocks, so a product gate at 1.05 would extend and refuse settled fits. In the three acceptance configurations whose products failed the book's gate at the defaults (a Poisson undirected static rank-2 fit, and frn and ordinal directed fits with `dynamic_uv`), the smallest product effective sample size of a failing run was 19 to 87 of 800 draws. Run at four times the default `nscan`, all three passed, and the posterior-mean product `fit$UVPM` moved by at most 0.27 posterior standard deviations on any dyad. A study that reports individual dyadic products, rather than `fit$UVPM` or the monitored summaries, keeps the position draws (`posterior_opts = list(save_UV = TRUE)`) and continues the chains with `lame_resume()` until the R-hat of every reported product is below 1.05.

Model Selection Guidelines: Use both dynamic_uv and dynamic_ab when:

- Networks show clear temporal trends in density or clustering
- Individual node behavior changes systematically over time
- Community structure evolves (merging, splitting, drift)

Use only dynamic_uv when:

- Latent structure/communities change but individual effects are stable
- Focus is on evolving homophily or clustering patterns
- Network shows structural reconfiguration over time

Use only dynamic_ab when:

- Individual heterogeneity varies but overall structure is stable
- Actors' activity/popularity changes over observation period
- Focus is on individual-level temporal dynamics

Bipartite Network Models:

When mode="bipartite", the model handles rectangular adjacency matrices Y with dimensions n_A x n_B, where n_A and n_B represent the number of row and column nodes respectively.

Static Bipartite Case: The model uses separate latent factor matrices:

- U: n_A x R_row matrix of row node latent positions
- V: n_B x R_col matrix of column node latent positions
- G: R_row x R_col interaction matrix mapping between latent spaces
- Multiplicative term: U G V' captures bipartite community structure

Dynamic Bipartite Case: When dynamic_uv=TRUE for bipartite networks:

$$U_{i,k,t} = \rho_{uv} U_{i,k,t-1} + \epsilon_{i,k,t}$$

$$V_{j,k,t} = \rho_{uv} V_{j,k,t-1} + \eta_{j,k,t}$$

where i indexes row nodes, j indexes column nodes, k indexes latent dimensions.

A time-varying bipartite interaction matrix is not supported: `dynamic_G = TRUE` is refused on both the MCMC and the ALS route. With moving positions a per-period interaction matrix is not identified apart from them – $U_t G_t V_t'$ and $(U_t G_t) V_t'$ describe the same products, so a fit cannot separate movement in G_t from movement in U_t and V_t. With static positions, $U G_t V'$ is identified as a static G is, but it is not implemented. Use `dynamic_uv = TRUE` for a time-varying multiplicative term and read the per-period product `fit$UVPM`, which is the identified quantity; every bipartite fit still estimates a single interaction matrix and returns it as `fit$G`.

Key Differences from Unipartite Models:

- No dyadic correlation (rho): Bipartite edges are inherently directed
- Separate dimensions: R_row and R_col can differ for row/column spaces
- Rectangular structure: Network density patterns differ from square matrices
- Community interpretation: Captures affiliation patterns between node types

Missing covariates:

A cell whose dyadic covariate, sender (row) covariate or receiver (column) covariate is missing in its period is an unobserved tie, for unipartite and bipartite panels alike: it is set to `NA` in `Y` before the design is built, and the covariate coefficients are estimated from the complete cells. A message gives the number of such cells. A missing value is never modelled as 0, which would bias the coefficient of a covariate that is not centred. In a symmetric panel a dyad has one covariate value: a hole in one triangle of a dyadic covariate is filled from the other triangle (a message gives the number of values), so the slice stays symmetric and the dyad stays observed, and a dyad missing in both triangles is unobserved. A missing node covariate removes the actor's row and column in that period. A covariate that is missing in every period stops the fit. `ame` and the ALS routes apply the same rule and give the same messages.

Collinear covariates:

A regression design whose columns are exactly aliased over the observed cells of the whole panel (a covariate that duplicates another, or one that is a linear combination of others, counting the intercept and the sender and receiver covariates), static or with `dynamic_beta`, is an error of class `lame_design_rank`. It names the aliased columns and the combination each equals: the data cannot separate their coefficients, so the posterior along that direction would be the prior's alone. The periods are pooled, so a covariate that duplicates another in some periods only is fitted, and period indicators that span the intercept still drop it, with the warning. A column is aliased when it is a combination of the columns before it to a relative 1e-8 (as `lm()` reports them); a design that is only nearly collinear is fitted. `method = "als"` refuses the same designs with the same condition (fields `aliased`, `rank` and `n_col`) for every estimator and family.

Standard AME Model Types:

The following describes the six standard data types/models available:

"normal": A normal AME model.

"binary": A binary probit AME model.

"ordinal": An ordinal probit AME model. An intercept is not identifiable with data-induced cutpoints. Explicit cutpoints anchored at zero support an intercept.

"cbin": An AME model for censored binary data. The value of 'odmax' specifies the maximum number of links each row may have.

"frn": An AME model for fixed rank nomination networks. A higher value of the rank indicates a stronger relationship. The value of 'odmax' specifies the maximum number of links each row may have.

For both families a missing cell (`NA`) of an actor present in the period is an unobserved nomination or non-nomination: it counts against its row's cap while it holds a nomination, and the sampler draws it from its conditional given the row's observed cells. A row at its cap is bounded below only, so the sweep moves its level slowly; each sender's level is therefore also moved together with its latent row once per sweep, drawn from its conditional. On simulated designs (30 actors and one period, 24 actors and two to six, cap 10, more than 3,000 fits) the 95 percent intervals of `vb` and `cab` covered 0.97 to 1.00 of the panels at every share of rows at the cap, with and without a fifth of the cells hidden, and those of `va` 0.90 to 1.00 outside the region `odmax` describes and 0.91 inside it. Two designs withhold the level of the rows (see `odmax`): one period, where a row at its cap shows only that its level fills the cap, with every row at the cap, or with cells hidden and a share at the cap from 0.5 (the intercept interval covered 0.64 and 0.84 of the panels; `ame()` and one-period `lame()` report the intercept and any row-covariate coefficient as not estimated then); and two periods or more with a share from 0.5 and a posterior-mean `va` from 0.5 to 1.5. At a generating sender sd of 0.25 the posterior mean of `va` runs up to 2.3 times the generating variance while the interval still covers. `fit$hidden_cell_ess` records the effective sample size of the intercept and `va` of a fit with hidden cells. The posterior predictive (`YPM`, `fitted()`, the goodness-of-fit replicates and `simulate()`) lets a missing cell hold a nomination, as the sampler does, so `YPM` also predicts those cells; the diagonal and the cells of an actor absent from the period are left out.

"poisson": An overdispersed Poisson-lognormal model for counts (negative or non-integer observed values are an error): $Y \sim \mathrm{Poisson}(e \exp(z))$ with $z \sim N(\eta, \sigma^2)$. Given η and σ^2, a cell's mean count is $e \exp(\eta + \sigma^2/2)$. `YPM`, `fitted()` and `predict(type = "response")` report it, for the exposure e_t of the period (`period_exposure`; one without), at the posterior mean η of the linear predictor (`EZ`) and the posterior mean σ^2 of the residual variance, the form the fast estimators report. Predictive quantiles and event probabilities come from posterior predictive draws (`simulate()`, `predict(fit, h = 1, interval = "prediction")`). Zero exposure gives a structural zero count.

Joint scale move (binary, unipartite and bipartite):

A probit panel identifies the scale of the linear predictor only weakly: a higher intercept with less spread in the additive effects and smaller latent positions fits a sparse panel nearly as well, so the level of the predictor and the spread of the effects trade off along a ridge. The conditional sweep cannot step along it, because conditioning on the latent Z pins the scale of every block, and a cold start can take many thousands of sweeps to arrive. Binary unipartite fits therefore carry a Metropolis step that multiplies the coefficients and additive effects by a factor c and the latent positions by $\sqrt{c}$, with the variance parameters held fixed over the step and following it through their own draws. Under `dynamic_ab` the same visit walks a second direction: the per-period deviations of the additive effects together with σ_{ab}, whose amplitude relative to that innovation scale is a ridge of its own (the deviation draw pins the deviations given σ_{ab} and its draw pins σ_{ab} given them). Moving the two together leaves their AR(1) prior invariant, so only the likelihood and the inverse-gamma prior on σ_{ab}^2 resist; set `prior$dev_move = FALSE` to leave that direction out. The step integrates Z out (the dyad errors are split into a shared factor and an independent part; the factor is drawn from its conditional and the independent part integrated out), so it is accepted against the exact conditional posterior and leaves it unchanged; Z is redrawn from its conditional when the step is accepted. It is visited every `prior$level_move_every` sweeps (default 30) with `prior$level_move_tries` proposals per visit (default 8), adapts its step size during burn-in only, and reports its acceptance rate in `fit$level_move`. It is off for other families, symmetric fits and `dynamic_rho`. Changing coefficients with static factors are supported for ranks 1 and 2 when both actor margins and `dynamic_ab` are enabled, `dcor = FALSE`, and the coefficient law is AR(1) or RW1 on the regular grid, with zero initial means and diagonal innovation scaling. The static factor covariance is integrated in the acceptance density.

That configuration also receives a joint Hamiltonian update of coefficients, actor effects and factors with the binary latent responses integrated out. It requires at least three actors and one static coefficient. The update uses exact actor contrasts, keeps hyperparameters fixed, and adapts its step size during burn-in only. Bounded momentum kicks preserve reversal and phase-space volume; the Metropolis correction retains the original target. It runs every 20 sweeps; acceptance and numerical rejections are recorded in `fit$marginal_move`. Set `prior$marginal_move = FALSE` to omit this update. With static additive effects, an intercept, both actor margins and the same rank-one or rank-two changing-coefficient configuration, a joint observed-data update moves static actor contrasts, coefficients and factors. It integrates out the static factor covariance under its original prior. Bounded momentum updates and a Metropolis correction preserve the original target. The move begins after half of burn-in and runs every 20 sweeps; its step size adapts only during burn-in. Acceptance and numerical rejections are recorded in `fit$marginal_move`. Set `prior$marginal_move = FALSE` to omit this static-actor update.

Other changing-coefficient and static-factor configurations retain their existing updates.

Binary bipartite fits with an intercept, both margins, static coefficients and, when they have a multiplicative term, a static interaction matrix carry the same step on the whole predictor: the coefficients, the additive effects (and their deviations under `dynamic_ab`) and the interaction matrix G are multiplied by c with the latent positions held. The cells of a bipartite panel have no dyadic correlation, so no shared factor is drawn, and the prior terms of every block the factor touches (the coefficient prior, the effects, their AR(1) deviations, the $N(0,1)$ prior of the entries of G cut at `prior$G_max`) are included in the acceptance ratio.

Binary unipartite fits (directed or symmetric, with any additive margin in the model, and with static coefficients or `dynamic_beta` of kind `"ar1"` or `"rw1"`) also receive a collapsed per-actor update. The data-augmentation sweep draws an actor's parameters from a normal whose precision is the

number of cells the actor appears in, so an actor whose cells hold no events (a flat likelihood on a half line) moves by about $1/\sqrt{m}$ per sweep however wide its posterior is, and relaxes in about vm sweeps: thousands of sweeps for the additive level of an inactive actor, and 10^5 for static positions that have spread out, because the positions also feed the factor covariance they are drawn under. The update integrates the latent array out (as the scale move does) and draws, one actor at a time, an exact elliptical slice step of the actor's sender and receiver level translations and, for static positions, of its sender and receiver positions. The level translation adds a constant to one actor's predictor while the other levels and the intercept absorb the re-centring, so it keeps the sum-to-zero constraint and touches one row (or column) of the likelihood. The latent array is redrawn afterwards, so the posterior is unchanged; there is nothing to tune. It is available for `dynamic_ab` and static effects, ranks 0 to any with static positions (positions that move over time keep their own updates; only the levels are moved), and is visited every `prior$actor_move_every` sweeps (default 10). The positions are left to the sampler during the first half of burn-in, when it runs under a fixed diffuse prior that lets the positions of actors without information take any scale. In a symmetric fit an actor has one level, which enters both its row and its column, and the positions are those of the eigenmodel (u_i with the diagonal weights held fixed during the visit, drawn under the sampler's own precision prior); there is no dyadic correlation. Panels with absent actors (changing composition) are covered: only observed cells enter a likelihood, an actor with no observed cell in a block has the prior as its conditional there and the slice step moves it across that prior, the translation moves the level of absent periods with the rest, and the missing cells are redrawn from the model afterwards, as the sampler does. Set `prior$actor_move = FALSE` to omit it; its work is recorded in `fit$actor_move`. For directed fits with `dcor = TRUE` the same visit also updates the dyadic correlation with both latent responses of each dyad integrated out (an independent proposal from its $1-\rho^2$ prior, accepted on the exact pair likelihood, followed by an exact redraw of the latent array), because given the augmented array the correlation moves by about $1/\sqrt{\text{dyads}}$ and mixes slowly when few dyads are mutual; `prior$actor_move_rho = FALSE` omits that part.

The Poisson and ordinal families have the same slow direction, and the latent array cannot be integrated out, so their update carries it along: an actor's level is translated together with the latent values of its row (its column, or the pairs of a symmetric network), which leaves every residual, hence the Gaussian density of the latent array, unchanged. The shift is then an exact draw from the level prior (with the intercept that absorbs the re-centring) times the data factor of the row: the log-concave Poisson factor of the observed cells (an elliptical slice step) or, for the ordinal family, the indicator that the shifted values keep their ranks (or stay inside their cutpoint intervals), which cuts the prior at both ends and is drawn as a truncated normal. The intercept, the spread of the levels and the levels of the inactive actors, which the sweep alone crosses over thousands of sweeps on a sparse panel, then mix within a few visits. With static coefficients the Poisson family also translates each coefficient with the latent array, and moves the dyadic variance and the dyadic correlation together with the latent residuals (a Metropolis step whose ratio is the change of the Poisson factor and the priors), because given the array the residuals are prior draws wherever the counts say little, and, with static positions, redraws each actor's position with the latent values of its row or column carried along (an elliptical slice step under the factor prior the sampler draws under; the positions are left to the sampler during the first half of burn-in, when it runs under a fixed diffuse prior); the ordinal family draws the location of the latent array (the means of the levels, or the intercept), which the rank likelihood leaves free and explicit cutpoints confine to a strip. Visits are every `prior$actor_move_every` sweeps (default 10).

With moving positions (`dynamic_uv = TRUE` with a positive rank, of the `"ar1"` kind, or the `"t"` kind of a directed network) the sweep moves the position path of an actor whose cells say little (no events, or absent from most periods) by a step of about $\sqrt{s_2/m}$ per sweep, m the number

of its cells, however wide the posterior of the path is (s_2 is the dyadic variance, near 0.05 for a sparse Poisson panel and 1 for the probit families). The paths of such actors then wander at no likelihood cost, and the squared product of the positions, the persistence and innovation scale of the paths and the intercept follow them with autocorrelation times of hundreds to thousands of sweeps. Every `prior$actor_move_every` sweeps the update therefore redraws each actor's whole path, the senders' and then the receivers' (a symmetric network's once), with one elliptical slice step under the AR(1) prior of the path (stationary start; the local innovation scales of the `"t"` kind), the persistence and the innovation scale being held. For the Poisson family the latent values of the actor's row or column are moved with the predictor, which changes no residual, so the conditional of the path is its prior times the Poisson factor of the cells. For the binary family the latent array is integrated out as in the collapsed update above (the shared dyad factor of a directed network is drawn once per visit, each observed cell contributes a probit factor, and the latent array is redrawn at the end). Bipartite fits use their interaction matrix and symmetric fits the signed weights of their unit-variance coordinate paths. A directed binary fit with `dynamic_ab` at rank two already has the paired-actor slice step described below and is left to it; the `"snap"` kind and ordinal fits keep the sampler's own updates. A symmetric Poisson or binary fit also draws, every sweep, the scale ridge between its coordinate paths and their weights (scaling a coordinate of every path by c and its weight by $1/c^2$ changes no cell of the predictor, so the draw is a Metropolis step on $\log c$ with the ratio of the priors; it is recorded as `dpm_scale_attempts` and `dpm_scale_accepts`) and draws the persistence `rho_uv` with a proposal scale taken from the transitions of the paths, where the fixed scale of the sampler's own draw is 10 to 90 times the width of the conditional at persistences from 0.9 to 0.985 and moves `rho_uv` in one call of every ten to ninety. `prior$actor_move_paths = FALSE` omits this part only.

With `dynamic_beta` the prior of the intercept enters the density of a level translation through its path: the translation moves the whole path by the same constant, which changes the prior of the first state and, unless the path is a random walk, of the transitions. A static intercept next to moving coefficients enters through its own prior, and moving coefficients other than the intercept are not touched. A fit with no additive margin (`rvar = cvar = FALSE`, or a symmetric fit with `nvar = FALSE`) and static positions runs the position steps alone, and a fit with one margin translates the levels of that margin. Coefficient paths of the other kinds (`"rw2"`, `"matern32"`) or on a non-unit period grid keep their own updates.

The same update runs for bipartite binary fits (and in `ame` for a bipartite cross-section): each row and column actor's level translation, which moves the intercept with it and touches one row or column of the likelihood, and, for static positions, its position, as exact elliptical slice draws with the latent array integrated out; under `dynamic_ab` it also draws the deviation paths of random pairs of actors of a margin with their sum held. A bipartite cell has no partner cell, so there is no reciprocity update, and the positions have a fixed unit-variance prior, so there is no factor covariance to draw. Missing cells and the cells of an actor absent from a period carry no likelihood, and the update is never switched off by missingness. With `dynamic_beta` of kind `"ar1"` or `"rw1"` on the unit period grid the translation moves the whole intercept path by the same constant (a static intercept next to moving coefficients by the same amount), and the prior of the intercept enters the density of the translation through its path, as for unipartite fits.

A `"poisson"` fit translates the coefficients that move as well. The counts of a period fix the log of its mean to about $1/\sqrt{\text{events}}$, while given the latent array the intercept of the period is known to $\sqrt{s^2/\text{cells}}$, 10 to 20 times less on a sparse panel (0.015 to 0.022 against 0.0012 at 170 actors); the sweep moved the intercepts of the periods, and with them the coefficient of a dyadic covariate and the dyadic variance and correlation, by random walk. The gate panel of 170 actors and four periods had 3 to 14 effective draws of the period intercepts in 1,600 (split R-hat 1.5), 6 of the covariate co-

efficient, 6 of the correlation and 16 of the variance. Raising coefficient k of one period by δ and the latent values of the period by δx_{kt} leaves every residual unchanged, so the shift is an exact draw (an elliptical slice step) from the prior of the shifted path times the log-concave Poisson factor of the period's cells. The prior is the one of the path's kind (`"ar1"`, `"rw1"`, `"rw2"`, `"matern32"`, on any time grid), read from the precision the path sampler uses. The whole path of a coefficient is translated the same way, and so are the static coefficients next to moving ones, with the latent array of every period, at the visits of the translations of the actor levels (`prior$poisson_period_move = FALSE` and `prior$poisson_static_move = FALSE` omit them). Those fits, and every other `"poisson"` fit without `dynamic_rho`, then move the dyadic variance and correlation at every sweep with the latent residuals carried along their conditional law: given the counts the residuals of a dyad are close to a Gaussian with mean m and covariance V (the Gaussian prior times the first-order terms of the Poisson factors), the move maps $e' = m' + L'L^{-1}(e - m)$ for a proposed variance and correlation, and accepts with the Metropolis ratio of the joint density (the change of the Poisson factors, of the Gaussian density and the Jacobian of the map). The residuals of the cells with a count are tilted by about $s^2(y - \lambda)$, which follows the variance and not its square root, so scaling them as the translations of the actor levels do accepts steps of a few hundredths in the log variance, and this map accepts steps of 0.1 to 0.8. The visits of the translations of the actor levels are every second sweep for the Poisson family (every tenth left the least active actors with 30 to 80 effective draws of 2,000); `fit$poisson_period_move` records the moves. On the gate panel the period intercepts then have 1,370 to 1,450 effective draws of 1,600, the covariate coefficient 1,320, the correlation 650, the variance 850 and the slowest actor effect 400, and the split R-hat is 1.006; a sweep takes about a third longer.

An `"ordinal"` fit draws the scale of its latent array along the transformations that keep every rank. With the default data-induced cutpoints a cell of the latent array is drawn between the extreme values of the neighbouring categories, and those extremes are order statistics of thousands of cells: they move by steps of $1/\mathrm{cells}$ per sweep, while the scale of the whole configuration (the spread of the implied cutpoints against the unit residual variance) has a posterior of width $1/\sqrt{\mathrm{cells}}$, and given the latent array the coefficients, the additive effects and the dyadic correlation are tied to it. On 150 actors the coefficient of a dyadic covariate had 8 effective draws of 600 and the correlation 95; on 100 actors in four periods, whose rank windows are separate, 3 of 400 and 3. Multiplying the latent array, the coefficients and the static additive effects by a factor c keeps every rank, and c has an exact conditional (a group move whose invariant measure is dc/c: a gamma draw when the priors have mean zero and the predictor is fully scaled, an independence Metropolis step from its Laplace approximation otherwise). The shift and the scale of the latent array of each period alone keep that period's ranks and have exact conditionals too. Both are drawn at every sweep, and the same panels then have 600 and 400 effective draws of the coefficient and 400 and 270 of the correlation. The multiplicative effects, the dynamic additive effects and the dynamic positions stay unscaled parts of the predictor: the moves stay exact, and their efficiency is the share of the signal they scale. With explicit cutpoints the scale of the whole state carries the free cutpoints with it; the cutpoints of a directed fit with `dcor = TRUE` are drawn given the latent array, and on 50 actors the coefficient had 56 effective draws of 500 and the correlation 47, now 440 and 290 (`prior$ordinal_scale_move = FALSE` omits the moves; `fit$ordinal_scale_move` records them).

A `"binary"` fit draws the scale of its linear predictor with the latent array integrated out. When a covariate separates the ties (every tie lies above every non-tie on it) the likelihood increases without a bound along the ray that multiplies the coefficients and the additive effects by a factor $c > 1$, and the posterior of the coefficient is its prior times a likelihood that saturates only where the dyads nearest the boundary are classified. The data-augmentation sweep draws every block given the latent array and crosses that direction by steps of $1/\sqrt{\mathrm{cells}}$: on a 30-actor directed panel the separating

coefficient had 12 effective draws of 2,000 (steps of 0.03 against a posterior sd of 0.6), and on a bipartite panel, whose coefficients have the independent prior $N(0, g)$ with g the number of cells, it was still climbing after 8,000 sweeps (14.0 and 15.6 for two seeds, against a posterior mean of 59). Every `prior$binary_scale_every`-th sweep (default 5) the coefficients and the static additive effects are therefore multiplied by a factor drawn exactly from its conditional density, a group move whose invariant measure is dc/c: the dyad errors of a directed network are split into a shared factor, drawn from its conditional given the latent array, and independent parts that are integrated out, so that each cell contributes one probit factor, and the latent array is redrawn from its conditional afterwards (the device of the collapsed actor updates). The multiplicative effects, the dynamic additive effects and the actor-specific slopes stay where they are and the draw stays exact; its efficiency is the share of the signal it scales. The same panels then have 460 and 140 effective draws of 2,000 after 8,000 sweeps, two seeds agree, and the posterior mean is the same at 2,000, 8,000 and 32,000 sweeps. The draw runs in every binary configuration of `ame()` and `lame()` except `dynamic_rho`, next to the joint scale move and the Hamiltonian updates described above where those run: in the Hamiltonian configuration (changing coefficients and effects, rank one) the separating coefficient had 16 to 20 effective draws of 2,000 after 8,000 sweeps (the two seeds agreed on its mean to within the Monte Carlo error) and now has 1,000 to 1,200, and in the static directed configuration 140 to 200 and now 630 to 910. `prior$binary_scale_move = FALSE` omits it; `fit$binary_scale_move` records it.

The actor-period effects of a `"poisson"` fit with `dynamic_ab` are moved together with the latent values of their rows and columns. Given the latent array an effect has the precision of the cells it touches over the residual variance, so a Poisson panel of sparse counts moved the effects, and with them the innovation scale `sigma_ab`, the persistence and the levels, by small steps (on 50 actors in five periods `sigma_ab` had 6 to 34 effective draws of 800 and the effects of the first and last period, whose paths have one neighbour, R-hat up to 1.34). Each sweep moves an effect and the latent values of its row by one amount, the cell residuals staying where they are, so that the conditional is the prior of the effect given its neighbours and level times a Poisson factor that is two sums per row; the draw is an elliptical slice step, and with an intercept the effects of a period are moved in pairs, which keeps their sum. `sigma_ab` then has 216 to 311 effective draws (`prior$ab_period_move = FALSE` omits the move; `fit$ab_period_move` records it).

Under `dynamic_ab` (every family; bipartite, directed and symmetric unipartite networks) the effect of an actor in a period is a level plus a stationary AR(1) deviation, and the likelihood sees only their sum, so a short or sparse panel splits the common variance of the effects between the level variance and a persistent deviation almost freely, and the sweep, which draws the levels given the hyperparameters and the hyperparameters given the levels, crosses that ridge only over thousands of sweeps. Each sweep therefore draws the level variances (for a directed network their covariance, the pair of an actor's levels being correlated), the innovation scale and the persistence from their density given the totals, with the levels integrated out, before the levels are redrawn from their conditional. The update uses only the totals, so it needs no latent array and does not depend on the family, leaves the posterior unchanged and is recorded in `fit$ab_hyper_move`; `prior$ab_hyper_move = FALSE` omits it and `prior$ab_hyper_move_tries` sets the proposals per sweep (default 8). For bipartite fits with static positions the scale of the positions trades against the scale of the interaction matrix, which the unit priors anchor weakly. Each sweep also draws, exactly from its conditional, the factors by which the columns of the positions and the rows and columns of the interaction matrix are rescaled without changing their product (recorded in `fit$uv_g_scale_move`; `prior$uv_g_scale_move = FALSE` omits it).

Directed `"cbin"` and `"frn"` fits receive a receiver-level counterpart of the update their sender levels already have, which translates a sender's level together with its whole latent row. The data-

augmentation sweep alone moves a receiver's level and its latent column by about $1/\sqrt{m}$ per sweep, m the number of cells in the column, however wide the posterior is, and for a receiver nobody nominates (a column bounded above only) the effective sample size of the level falls roughly as $1/n$. One sweep of the update visits each receiver once and translates its level and its latent column by the same amount. Every residual is unchanged, so the amount enters the density only through the prior of the shifted level and the order constraints of the translated cells (a nominee stays above zero and above the non-nominees of its row, a non-nominee stays below the weakest nominee and, while the row has room, below zero, and the ranks of `"frn"` rows stay in order). It is drawn exactly from the prior of the level cut to the interval those constraints leave: there is no proposal and nothing to tune. With an intercept the levels are centred and the translation is compensated by the other levels and the intercept, with the prior of the intercept entering: the coefficient prior of a static intercept or, in a `dynamic_beta` fit of kind `"ar1"` or `"rw1"` on the unit period grid, the prior of the intercept's path, which the translation moves by one constant (a static intercept in a `dynamic_beta` fit enters through its own prior). Hidden cells are labelled by their latent values, as in the sampler, and keep their labels during the move. Panels with absent actors and bipartite fits are covered, and so are `dynamic_ab` fits, whose sender and receiver translations move the level together with its whole path (the deviations from the level are untouched), and `dynamic_beta_per_actor` fits, whose actor-specific slopes are a function of their own parameters and the design alone and so stay where they are while the levels, the intercept and the latent array move. The sender translation of a `dynamic_ab` fit, which a static fit has always made every sweep, runs under the same switch and schedule as the others. Symmetric `"cbin"` and `"frn"` fits are not available. The update runs every sweep (the compiled sweep adds roughly a tenth to the time of a sweep); `prior$actor_move_every` changes that, `prior$actor_move = FALSE` omits the update, and `fit$actor_move` records its work.

Two more updates serve the same fits. A row at its cap is bounded below only, but a row with room (fewer nominations than its cap) is pinned from both sides, its non-nominations below zero and its nominations above, and a sparse panel has a few of them (3 to 15 percent of the rows): translating the whole latent array together with the intercept has room for a step of the order of 0.01, so the intercept moved by about $1/\sqrt{m}$ per sweep however wide its posterior was (4 to 12 effective draws of 1,200 at 170 actors, and a level of the predictor that failed the convergence gate). The level update translates the rows that are free above (the rows at their cap in every period) by one amount and leaves the pinned rows where they are: the intercept takes the compensation (with centred levels it and the pinned rows' levels take it between them), every residual is unchanged, and the amount is drawn exactly from the prior of the levels and the intercept cut at the weakest nomination of the moving rows. It needs an intercept (a static coefficient, or the path of a `dynamic_beta` fit of kind `"ar1"` or `"rw1"` on the unit period grid) and the row effects, since the row effects are what keeps the rows with room in place while the other rows rise (with `rvar = FALSE` the receiver and reciprocity updates still run); a bipartite fit of `ame()` (levels free of centring) also draws, exactly, along the ridge the intercept shares with the means of its levels. The dyadic correlation of a directed fit with `dcor = TRUE` has the same trouble on a larger scale: the non-nominations of rows at their cap only have to stay below their row's weakest nomination, so their pair correlation is whatever ρ and the latent draws make it, and given the latent array ρ moves by about $1/\sqrt{m}$ per sweep, m the number of dyads, against a posterior a few tenths wide (4 to 8 effective draws of 1,200 at 170 actors). Every tenth visit the reciprocity update integrates the non-nominations and the cells no nomination can sit in out of the density of ρ, draws it by Metropolis steps given the nominations, and redraws the cells it integrated out from their conditionals given the nominations and the new value (`prior$actor_move_rho = FALSE` omits it; `fit$actor_move` records its proposals and accepts).

In a `dynamic_beta` fit of kind `"ar1"` or `"rw1"` on the unit period grid with a dynamic intercept, two more updates move the path of the intercept period by period. A period in which almost every

row is at its cap (0 and 1 rows with room of 170 in two of four periods of the simulated panels) has a level the data hardly inform, and the level update above, which moves one amount for every period, is stopped by the rows with room of the other periods: the intercepts of those periods had 4 to 7 effective draws of 1,600 (two chains of 800), the least mobile rows with room 13 to 24, and the innovation scale of the path, which follows its shape, 6 to 16; the two-seed convergence gate failed on it (split R-hat 1.10 for `"cbin"`, 1.27 for `"frn"`). The period update translates the periods that no row with room ties to another one (the connected components of the graph of periods and rows with room) by one amount each, the rows with room compensated by their own levels and the intercepts of the other periods, and draws the amounts exactly. The room update moves the intercept of a period that has rows with room by itself. A row with room is a probit row (its nominations above zero, its non-nominations below), so the Gaussian conditional of the intercept given the latent cells of those rows is narrower than its posterior by the information a latent cell loses when it is integrated out (0.13 to 0.37 per non-nomination). Given a shift of the intercept of a period, the latent slice of the period translates with it, the non-nominations of its rows with room are integrated out of the density of the shift and redrawn from their conditional afterwards, the dyadic correlation entering through the partner cell; the shift is a Metropolis step on that density, so that the intercept moves by its posterior width, not by the width of the Gaussian conditional. A period with a row whose hidden cells leave its room open (fewer observed nominations than the cap and an observed non-nomination) is left to the exact translation. `prior$actor_move_period = FALSE` and `prior$actor_move_room = FALSE` omit the two updates (`prior$actor_move = FALSE` omits every nomination update); `fit$actor_move` records their work.

Directed binary panels with rank-two Gaussian AR(1) position paths, dynamic additive effects, static coefficients, an intercept, both actor margins and the default unit period grid also receive joint Hamiltonian updates. These integrate both binary latent responses out and retain the full reciprocal-pair likelihood. Factor persistence and stationary scale move in the same Hamiltonian trajectory as their whitened paths, with their original priors and transformation Jacobians retained. The metric uses fixed prior references rather than either proposed endpoint. Additional updates move persistence with path scales, translate factors while integrating and redrawing actor levels, and update dyadic correlation with the latent responses integrated out. Hamiltonian step sizes adapt during burn-in only and survive checkpoints. Set `prior$marginal_move = FALSE` to disable this group. Other configurations retain their existing updates. These moves preserve the posterior target.

A directed unipartite `dynamic_uv` fit of the `"ar1"` or `"t"` kind with a normal, Poisson or binary outcome starts its chain from a dynamic AR(1) ALS fit of the same model, mapped through `als_start_vals`, unless `start_vals` is supplied or `prior$als_start = FALSE`. Bipartite fits keep the cold start: a bipartite ALS solution parks its scale in the interaction matrix. An ALS start places the initial path on the observed temporal structure; the chain then samples the paths and hyperparameters from there. The ALS fit is deterministic, so the start does not depend on `seed`; it is the penalised point estimate (`bias_correct = "none"`), not the empirical-Bayes fit. A start fit that is refused as unconverged (class `"lame_als_convergence"`) contributes its last iterate; a start with non-finite values, or positions beyond `prior$uv_max_abs`, is not used. `fit$als_start` records whether it was used.

Value

`auto_extend`	the record of the convergence policy: `max_extensions`, `extensions` (made), `discarded_sweeps`, `triggers` (a list with the flagged quantities of each extension) and the thresholds used.
`BETA`	posterior samples of regression coefficients. A 2-dimensional matrix `[n_stored,`

p] when all coefficients are static. When dynamic_beta flags any coefficient as time-varying, BETA is a 3-dimensional array [n_stored, p, T] whose third dimension is the time period. Static coefficients are still present with their values replicated across the third dimension. coef(fit) collapses this to a [p, T] posterior-mean matrix.

Migration from amen. Under amen::ame() the BETA slot was always 2-D. Scripts that compute apply(fit$BETA, 2, mean) on an amen fit will silently aggregate across periods when run against a lame() fit with dynamic_beta active (the second margin is the coefficient index in both shapes; the new third margin is time). Use length(dim(fit$BETA)) to detect the shape, or call coef(fit) which returns a [p, T] matrix in either case.

VC — posterior samples of the variance parameters. On a unipartite directed binary panel of two periods or more carrying fewer than 1.2 events per actor-period, the draws of va and vb are kept here but summary(), print() and confint() report them as not estimated, with the measured coverage. Under a supplied effects prior the variance draws are reported under it (see prior_supplied)

events_per_actor_period — for the families with an event rate (binary, cbin, frn, poisson) on a panel of at least two periods, the mean over periods of the events per row actor; the rate the reporting methods read

events_per_actor — on the same panels, the events per row actor over the whole panel (every period's events over the actors present in any period), quoted with the rate

prior_supplied — TRUE when prior carried Sab0, eta0 or etaab: where the data alone leave the effect variances to the prior (the sparse rule, a saturated cap) the reporting methods then show them under the supplied prior; the intercept and row-covariate coefficients a saturated cap withholds stay withheld

odmax_saturated_share — for the cbin and frn families, the share of observed rows whose observed outdegree sits at their odmax; the reporting methods read it alongside events_per_actor_period and stop reporting va, cab, the intercept and any row-covariate coefficient at 80 percent or above (60 percent for a panel of one period; odmax_sat_threshold records the one applied) (vb stays reported; see odmax)

APM — posterior mean of additive row effects a

BPM — posterior mean of additive column effects b

APM_sd, BPM_sd — posterior standard deviation of each additive effect (static effects), used by simulate() to draw the effects

U — multiplicative row effects u. For dynamic_uv = TRUE, a 3D array (n x R x T) whose periods factor the posterior mean product of each period at rank R and are aligned across periods; NA for an actor absent from a period (see latent_positions)

V — multiplicative column effects v (asymmetric case), with the same layout as U

U_sd, V_sd — for dynamic_uv = TRUE with saved latent draws (the default), the posterior standard deviation of each reported coordinate, same shape as U and V; latent_positions() reports them as posterior_sd

UVPM — on the union of actors; under changing composition the dynamic array has NA rows and columns for actors absent from a period, and a static fit gives one matrix for every pair. posterior mean of UV

ULUPM — posterior mean of ULU (symmetric case)

L — for symmetric fits, the diagonal of the product ULU': the eigenvalues of ULUPM when the positions are static; when they are dynamic, one diagonal for every period that carries the scale and sign of each direction, so that $U_t L U_t'$ is the best rank-R part of each period's ULUPM with those signs

EZ — posterior mean of the linear predictor Z, on the link scale for every family (the log rate for Poisson, the probit scale for the binary, ordinal, cbin and frn families); YPM is the response-scale counterpart. For mode = "bipartite", EZ is a list of per-period matrices with the row and column actors as dimnames, in the lexicographic sort of the input actor names (so "r10" sorts before "r2" unless you zero-pad).

YPM — Stored response-scale fitted value, one matrix per period: the average of the outcomes replicated at each stored draw, except for Poisson fits, where it is the mean count $e_t \exp(\eta + \sigma^2/2)$ at the posterior means of the linear predictor and of the residual variance; see the Poisson model details.

coefficient_prior — Recorded specification of the coefficient prior used by the fitting build, as displayed by prior_summary().

static_uv_prior — For static asymmetric unipartite factors, the inverse-Wishart covariance prior's degrees of freedom, scale and dimension, the data scale s of Suv0 = s I (data_scale) and the multiplier of $2R + 2$ in the degrees of freedom (df_multiplier; see kappa0 in ame). Older fit objects do not carry this record.

sampler_health — A data frame separating numerical failures from valid prior-bound rejections for each conditional sampler step. The compatibility counts in tryErrorChecks and mh_counters contain both. A numerical failure, in a conditional step or in a joint Hamiltonian, actor-factor slice or bipartite scale move, aborts the fit with an error of class lame_sampler_failure (the counts are in the health field of the condition), so a returned fit has none. A proposal outside the prior's box (prior$uv_max_abs, prior$G_max) is a Metropolis rejection of the model, counted here; summary() states it when it is on one percent or more of the sweeps. Joint Hamiltonian acceptance and rejection records are kept in their corresponding move fields. sampler_health_iterations gives the number of sweeps counted, including burn-in.

var_cap_hits — number of sweeps in which a stationary-variance cap (prior$uv_var_cap, prior$ab_var_cap) removed most of a conditional draw

level_move — for binary unipartite and bipartite fits, the attempts, accepts, acceptance rate and final step SD of the level direction of the joint scale move (absent when the move did not run)

actor_move — for Poisson and ordinal fits, the work of the translations: visits, the level translations drawn (level_calls) with the likelihood evaluations of the Poisson ones (level_evals) and the steps that failed to terminate (stuck) or that were left to the sampler because the current state lay outside the interval the translation

needs (skipped), the draws of the location of the latent array (location_calls, ordinal), the coefficient translations (coef_calls, coef_evals, coef_stuck; Poisson), the position steps (uv_calls, uv_evals, uv_stuck; Poisson with static positions) and the attempts and accepts of the scale move of the dyadic variance and correlation (scale_attempts, scale_accepts, and rho_attempts, rho_accepts of its correlation-only proposals; Poisson: the carried scale move, with dynamic_rho the one of the translations); for binary fits, the work of the collapsed per-actor update: visits, the slice steps of the level translations (level_calls) and of the static positions (uv_calls), the likelihood evaluations they used (level_evals, uv_evals), the steps that failed to terminate (stuck) and the attempts and accepts of the reciprocity update (rho_attempts, rho_accepts; zero for bipartite fits, which add the slice steps of the deviation paths of actor pairs under dynamic_ab, dev_calls and dev_evals); for "cbin" and "frn" fits, the work of the receiver-level translation: visits, the receivers moved (col_calls) and the receivers left to the sampler because the current state lay outside the interval the translation needs (col_skips), the sweeps in which the level update moved a set of rows (level_calls) and the rows it moved (level_rows), the sweeps in which the period update translated a set of periods (period_calls) and the sets it moved (period_comps), the proposals and accepts of the collapsed translation of the periods that have rows with room (room_attempts, room_accepts) and the moves refused because a redraw did not terminate (room_stuck), and the proposals and accepts of the reciprocity update (rho_attempts, rho_accepts); with moving positions, Poisson and binary fits add the path steps drawn (dpm_calls), the likelihood evaluations they used (dpm_evals) and those that failed to terminate (dpm_stuck), and symmetric fits the proposals and accepts of the scale ridge draw (dpm_scale_attempts, dpm_scale_accepts); absent when the updates did not run

beta_hyper_move
: for dynamic_beta fits, the sweeps, proposals and accepts of the interweaving step of the path hyperparameters and its acceptance rate; absent when the step did not run

poisson_period_move
: for "poisson" fits, the visits, the translations of the periods (period_calls) and of the whole paths (path_calls) and of the static coefficients (static_calls) drawn, the evaluations their slice steps used, and the proposals and accepts of the carried scale move (scale_attempts, scale_accepts, and for the correlation alone scale_rho_attempts, scale_rho_accepts); absent when the moves did not run

binary_scale_move
: for "binary" fits, the draws of the scale of the linear predictor (visits), the evaluations of its density they used (evals) and the draws that moved the state (moved); absent when the draw did not run

ordinal_scale_move
: for "ordinal" fits, the sweeps the scale moves visited (visits) and the sweeps at which the scale of the whole state moved (orbit_moved); absent when the moves did not run

pa_hyper_move
: for dynamic_beta_per_actor fits, the sweeps, proposals and accepts of the interweaving step of the actor-specific slope hyperparameters and its acceptance

rate; absent when the step did not run

`uv_nc_move` for `dynamic_uv` fits of the `"ar1"` kind, the calls and the draws that changed the value (`sym_calls`, `sym_moved`, `bip_calls`, `bip_moved`) of the non-centred persistence step (symmetric) and scale step (bipartite); absent when the move did not run

`uv_temper_move` for symmetric Poisson `dynamic_uv` fits, the tempered transitions made, accepted and those that changed the sign of the weight; absent when the move did not run

`ab_period_move` for `"poisson"` fits with `dynamic_ab`, the sweeps, the pairs (or single effects) moved and the slice evaluations of the translations of the actor-period effects; absent when the move did not run

`ab_hyper_move` for `dynamic_ab` fits, the proposals and accepts of the update of the additive-effect hyperparameters given the totals and the acceptance rate; absent when the update did not run

`uv_g_scale_move`
for bipartite fits with static positions, the number of sweeps and of rescaled coordinates of the draw along the scale ridge of the positions and the interaction matrix; absent when the update did not run

`marginal_move` For supported binary fits, joint-update attempts, accepts, acceptance rate, final step size and numerical rejections.

`bipartite_scale_move`
For supported Gaussian, binary, Poisson or ordinal bipartite fits, joint scale attempts, accepts, acceptance rate, prior-bound rejections and numerical rejections. The fixed proposal requires no adaptive tuning.

`actor_factor_slice`
For supported binary fits, completed paired actor/factor rounds, block and likelihood evaluation counts, prior-bound rejections, and separate numerical failures.

`dynamic_marginal_move`
For supported binary fits with moving factors, Hamiltonian attempts, accepts, acceptance rate, tuned step size, numerical rejections and prior-bound rejections.

`nc_ar_move` Attempts and accepts for the joint persistence and path-scale updates of the factors and actor effects.

`translation_move`
Number of visits to the factor/actor translation update, counting each margin separately.

`rho_collapsed_move`
Attempts, accepts and acceptance rate for dyadic correlation updates with binary latent responses integrated out.

`uv_scale_move` Attempts, accepts and acceptance rate for joint latent-factor and innovation-scale updates.

`dev_move` the same for the deviation direction, present for binary unipartite `dynamic_ab` fits

`als_start` for `dynamic_uv` fits, whether the chain started from a dynamic ALS fit (`FALSE` when start values were supplied, when `prior$als_start = FALSE`, for the snap kind, symmetric and bipartite fits and the rank families, or when the ALS fit failed or fell outside the range the sampler holds)

`sigma_ab_prior_scale`
: for `dynamic_ab` fits, the inverse-gamma scale of the deviation innovation variance, fixed before sampling

`sigma_beta_prior_scale`
: for `dynamic_beta` fits, the inverse-gamma scale of the coefficient innovation variance, fixed before sampling from the response scale

`G`
: for bipartite fits, the interaction matrix of UGV': the posterior mean of the gauged draws when the positions are static; when they are dynamic, a diagonal matrix fixed across periods that carries the scale, so that $U_t G V_t'$ is the best rank-R part of each period's `UVPM`

`GOF`
: a named list with one element per goodness-of-fit statistic (`sd.rowmean`, `sd.colmean`, `dyad.dep`, `cycle.dep`, `trans.dep`; for `mode = "bipartite"`, `sd.rowmean`, `sd.colmean`, `four.cycles`, with `four.cycle.dep` replacing the count for Gaussian outcomes). Weighted rectangle dependence is `NA` when no complete rectangle is observed or the values have zero variance; see `gof_stats_bipartite`. Each element is a `[T, 1 + n_stored]` matrix: one row per period, the first column the observed value and the rest the posterior predictive draws. For the data-induced ordinal family the predictive draws are quantile-matched to observed category frequencies and assess dependence conditional on those frequencies. Explicit ordinal fits instead use each draw's sampled cutpoints, allowing checks of category frequencies as well. See `gof` for post-hoc computation and `gof_plot` for visualization.

`a_level`, `b_level`
: for `dynamic_ab = TRUE`, the posterior means of the per-actor levels behind the effect paths; `predict()` and `simulate()` propagate the deviation around them

`a_dynamic`, `b_dynamic`
: for `dynamic_ab = TRUE`, the posterior mean effect paths (actors by periods), with `a_dynamic_sd` / `b_dynamic_sd` their posterior sds

`rho_ab`, `sigma_ab`, `rho_uv`, `sigma_uv`
: draws of the AR(1) persistence and innovation scale of the dynamic effects and positions, when fitted

`start_vals`
: Final parameter values from MCMC, can be used as the input for a future model run.

`model.name`
: Name of the model (if provided)

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

`ame` for cross-sectional models, `lame_als` for the fast MCMC-free point estimator, `als_dynamic_beta` for a regression-only penalised smoother on the time-varying coefficient path, `lame_resume` for resuming saved checkpoints, `gof` for post-hoc goodness-of-fit computation, `gof_plot` for visualizing GOF results, `latent_positions` for extracting latent positions as a tidy data frame, `procrustes_align` for Procrustes alignment of latent positions, `summary.lame` for model summaries, `coef.lame` for coefficient extraction. Long-format edgelists and covariates can be prepared with `netify::netify()` and passed directly to `lame()`, or converted explicitly with `netify::to_lame(lame = TRUE)`.

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit<-lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X,
          burn=5, nscan=5, odens=1, family="binary", chains = 1)
# you should run the Markov chain much longer than this


## Time-varying regression coefficients (dynamic_beta).
## Make every dyadic coefficient evolve as an AR(1):
fit_dyn <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X,
                family = "binary", R = 0,
                nscan = 60, burn = 15, odens = 5,
                dynamic_beta = "dyad", chains = 1)
dim(fit_dyn$BETA)        # [n_stored, p, T] -- 3-D when dynamic
coef(fit_dyn)            # [p, T] posterior-mean coefficient paths
confint(fit_dyn)         # per-period 95% credible intervals
summary(fit_dyn)         # prints a "Dynamic coefficients per period" block
```

---

lame_als *Fast (MCMC-free) AME estimation for a longitudinal network*

Description

Fits an additive and multiplicative effects (AME) model to a longitudinal (replicated) network by **iterative block coordinate descent**, producing a fast point estimate with no MCMC and no credible intervals. For normal, binary and Poisson data the default is an empirical-Bayes / variational estimator of the model that `lame` samples (see `bias_correct`); its `vcov()` and `confint()` are Laplace approximations. This is the longitudinal counterpart of `ame_als`; the estimated effects (`mu`, `beta`, `a`, `b`, `U`, `V`) are *static* (pooled across time), as in a non-dynamic `lame` fit.

The estimation algorithm adapts the iterative block coordinate descent estimator of the Social Influence Regression model of Hoff & Minhas (`sir::sir_alsfit()`) to the AME model; it is a port and adaptation, not original **lame** methodology. See `ame_als` for the algorithm details.

Usage

```
lame_als(
  Y,
  Xdyad = NULL,
  Xrow = NULL,
  Xcol = NULL,
  R = 0,
  family = "normal",
  mode = c("unipartite", "bipartite"),
  symmetric = FALSE,
  max_iter = 200,
```

```
  tol = 1e-06,
  lowrank_method = c("mm", "als", "hybrid"),
  non_normal_method = c("irls", "transform"),
  link = c("probit", "logit"),
  linear_solver = c("eigen", "qr", "auto"),
  multistart = c("none", "cheap", "full"),
  bias_correct = NULL,
  bootstrap = 0L,
  bootstrap_type = c("parametric", "block"),
  bootstrap_block_length = 1L,
  bootstrap_seed = NULL,
  verbose = TRUE,
  seed = 6886,
  dcor = !symmetric,
  prior = NULL
)
```

Arguments

`Y`	a list of `t` relational matrices, or a 3d array `[n_row, n_col, t]`. A named list may have changing actor composition; every slice must carry row and column names so actors can be aligned to the union panel. Unnamed lists and arrays are treated positionally and must have one fixed layout. A longitudinal **netify** object is also accepted and converted with `netify::to_lame(lame = TRUE)`.
`Xdyad`	a list of `T` dyadic covariate matrices/arrays, or `NULL`.
`Xrow`	a list of `T` row/sender covariate matrices, or `NULL`.
`Xcol`	a list of `T` column/receiver covariate matrices, or `NULL`.
`R`	integer dimension of the multiplicative effects (default `0`). **The covariate coefficients are conditional on this choice.** The multiplicative term $u_i' v_j$ is a flexible high-variance regressor that can correlate with the dyadic covariates, so the estimated `beta` can shift – and occasionally change sign – as `R` increases. Comparing against an `R = 0` fit is a useful check on whether the covariate story is being driven by the latent rank.
`family`	one of `"normal"`, `"binary"`, or `"poisson"`. The rank and censoring families are MCMC-only.
`mode`	`"unipartite"` (square) or `"bipartite"` (rectangular).
`symmetric`	logical; fit a symmetric (undirected) model. Unipartite only. Every dyadic covariate slice must be symmetric: an asymmetric one is an error of class `lame_symmetric_xdyad` that names the slice (and, for a panel, its first period), as in `ame` and `lame`.
`max_iter`	maximum number of block coordinate descent iterations (default 200), and maximum number of outer reweighting steps for IRLS. A fixed-effects fit that has not converged within it is refitted once with four times the limit, and is an error (class `"lame_als_convergence"`) if that run has not converged either, as is an empirical-Bayes fit that does not settle (without the second run); the limit also bounds each round of the joint refinement of a Gaussian fit. An empirical-Bayes reweighting step runs at most five sweeps (it refreshes the weights, the Laplace

correction and the variance updates after them). The variational stage of an empirical-Bayes fit runs at most `2 * max_iter` iterations. An empirical-Bayes fit whose iteration does not settle retires its weakest latent column and fits the next rank down (see **A rank the data do not support**); a `max_iter` below 200 disables that for an iteration that merely ran out of it.

`tol` convergence tolerance on the relative change in residual sum of squares (default `1e-6`). IRLS additionally requires the largest penalized likelihood score, scaled by the square root of its Fisher information plus penalty curvature, and the largest absolute log variance update to be at most `sqrt(tol)`.

`lowrank_method` inner solver for the multiplicative (low-rank) block: `"mm"` (default) weighted majorise-minimise; `"als"` alternating least squares; `"hybrid"` runs both and keeps the lower-objective result. `"als"`/`"hybrid"` converge faster than `"mm"` on strongly unbalanced longitudinal panels and are available for directed and bipartite models (symmetric fits always use `"mm"`). All three minimise the same objective, so the point estimate is unchanged for balanced data.

`non_normal_method`
for the non-normal ALS families (it does not apply to a normal fit), `"irls"` (default for `binary`/`poisson`) runs iteratively reweighted least squares, giving a fast GLM AME fit with coefficients on the requested link scale (Poisson log, binary logit/probit). `"transform"` fits one fixed Gaussian working response (`log(y+1)` for Poisson, rank-normal scores for binary); its coefficients are on an uncalibrated working scale and are mainly useful for direction/ranking checks. A directed `R > 0` IRLS fit uses the hybrid low-rank solver internally because the IRLS weights are unbalanced. Uncertainty for either path comes from the bootstrap or sandwich covariance.

`link` link for `non_normal_method = "irls"` with a `binary` family: `"probit"` (default; matches `ame`/`lame`) or `"logit"`. `poisson` always uses the log link; ignored otherwise.

`linear_solver` solver for the regression block: `"eigen"` (default) eigendecomposes the normal equations; `"qr"` uses a QR factorisation of the observed design, which is more stable for ill-conditioned covariates; `"auto"` picks `"qr"` when the design is ill-conditioned but full rank. All give the same answer for well-conditioned designs.

`multistart` for `R > 0` (a non-convex objective), `"none"` (default) fits from a single deterministic start; `"cheap"` (4 starts) and `"full"` (8 starts) also try random low-rank starts and keep the converged start with the lowest value of the criterion, evaluated at that start's own scales: for an IRLS fit the penalised deviance, which carries the factor penalty with its scale parameters (`"penalised deviance"`), and for an empirical-Bayes fit with a latent rank the Laplace marginal likelihood of the start, minus twice its logarithm (`"Laplace marginal likelihood"`; see **Choosing the rank**), which integrates the effects and the factors and does not move with the scale of a factor column the data do not use (with the default variational stage every start is refined, and starts of different modes end at the same variational fit, so they neither differ in `multistart_distance` nor warn). An empirical-Bayes start without it is rated by the penalised deviance plus the penalty of the actor effects, the log determinant of their information and the

hyperprior of their covariance ("empirical-Bayes objective"). The raw deviance is not compared: it always favours the least-shrunk optimum, the one with the largest scales. Other fits keep the lowest working SSE. The criterion (multistart_criterion), its value for each start (multistart_sse; for an empirical-Bayes fit it includes the hyperprior, which objective leaves out, or is the marginal likelihood), each start's observed-data deviance (multistart_deviance, IRLS fits), its distance from the kept start (multistart_distance, IRLS fits) and its convergence are recorded. The distance is the squared difference of the two fitted linear predictors, weighted by the mean of the two IRLS weights and summed over the observed cells: an approximation of the change in the deviance between the two fits. An IRLS fit warns of materially different local optima when a converged start is more than one deviance unit away from the kept one. Starts that end at the same fitted values can still differ in the penalised objective, which depends on the scales of the factors, and those settle more slowly than the fitted values (the Laplace marginal likelihood differs far less); that difference does not warn. Every multistart fit also keeps the coefficients each start reports (multistart_coefficients) and the distance of each start's multiplicative surface from the kept start's (multistart_product_distance), so that the spread of the estimates across the starts can be read from the fit. Reproducible given seed; the global RNG stream is left unchanged.

bias_correct — how an ALS fit deals with the finite-sample (incidental-parameter) bias of a fixed-effects fit. NULL (default) fits normal models, and binary and poisson models on the IRLS path, by the empirical-Bayes estimator ("eb"), for directed, bipartite and undirected networks, with or without a latent rank: a Gaussian prior on the additive effects with its covariance estimated from the data, coefficients that solve the Laplace-marginal equations, and Laplace standard errors from vcov() for every coefficient (see **Empirical-Bayes estimator**, **Normal data** and **Counts** in Details). A normal response is fitted as the model that ame and lame sample, with an exact Gaussian cell, and a count network as the Poisson-lognormal model that ame and lame sample. Other families are not corrected. "eb" asks for the empirical-Bayes estimator explicitly; it is an error (class "lame_als_eb") for another family and for a binary or count fit with non_normal_method = "transform". (The dynamic routes of lame(method = "als") have their own empirical-Bayes estimator for normal, binary and Poisson panels; see bias_correct in lame.) "none" returns the fixed-effects fit at any size and records the request (for a normal response, the least-squares fit of the earlier releases with the conditional sandwich covariance of the dyadic coefficients; for counts, the fit of the earlier releases, in which the effect of an actor without a count on a side runs to minus infinity); "jackknife" requires the split-sample correction of that fit, for any family, and is an error where it cannot be computed; "bootstrap" requires the parametric-bootstrap correction of a binary IRLS fit at any size. See **Split-sample bias correction**, **Parametric-bootstrap bias correction** and **Size boundary** in Details. A corrected fit keeps the uncorrected values in fit$raw and records the correction in fit$bias_correct; an empirical-Bayes fit records method = "eb" (default = TRUE when it was the default) and refits nothing.

bootstrap — integer: if > 0, additionally run bootstrap replicates of the parametric or block bootstrap (via ame_als_bootstrap) after the point fit and attach the result as

fit$bootstrap. The downstream accessors (confint.ame_als, summary, print) then surface bootstrap intervals instead of the anti-conservative sandwich Wald intervals. At least 30 replicates are needed: a smaller positive number is an error of class "lame_als_bootstrap_replicates" raised before anything is fitted. A binary network fitted with bias_correct = "none" has no bootstrap (an error of class "lame_als_bootstrap_bias", also raised before anything is fitted): see ame_als_bootstrap. Default 0 (no bootstrap) – bootstrap is expensive, so it is not imposed on a user who just wants a quick fit.

bootstrap_type character: "parametric" (default) or "block" – the bootstrap scheme to use when bootstrap > 0. The block scheme returns no standard error or interval for the parameters that resampling time slices cannot vary (the node-covariate coefficients and the variance components of the actor effects, among others; fit$bootstrap$withheld lists them with the reason), and the parametric scheme gives them; see ame_als_bootstrap.

bootstrap_block_length
integer: block length for the block bootstrap; only used when bootstrap_type = "block".

bootstrap_seed optional integer seed for the bootstrap (the point fit uses seed).

verbose logical; print progress (default TRUE).

seed random seed (default 6886) for the random starts of multistart = "cheap" / "full"; the caller's random number state is restored afterwards. The block coordinate descent itself is deterministic, so with multistart = "none" the point estimate does not depend on it.

dcor logical; whether a directed binary probit, Poisson or normal fit models the dyadic correlation of the latent errors of the two cells of a dyad, as ame and lame do with their dcor = TRUE. The default is !symmetric, the default of the samplers. With the empirical-Bayes estimator (the default of a binary fit) the correlation is part of the model that is fitted: the two cells of a dyad are one bivariate probit observation, fit$VC["rho"] is the correlation of that model, and the factors, the standard errors and the predictions are those of the model (see **Dyadic correlation** in Details). dcor = FALSE fits independent cells, the model of the earlier release, and reports the correlation of the pairs of cells at the fitted predictor as before. A directed count fit takes the pairs at its fitted state (see **Dyadic correlation of counts** in Details), and a directed normal fit takes the two cells of a dyad as one bivariate normal observation, in closed form (see **Dyadic correlation of normal data** in Details; it costs 1.3 to 1.6 times the independent-cell fit). A fit that has no pair likelihood (undirected and bipartite networks, the other estimators of a binary fit, and the fixed-effects normal fit) keeps its own estimate of rho; an explicit dcor = TRUE with the logit link is an error, because the latent correlation is that of normal errors. The pair likelihood of a binary or count fit costs 1 to 2.5 times the independent-cell fit for one period and 2 to 4.5 times for four, in proportion to the number of dyads and periods; dcor = FALSE is the fast path for very large networks. A directed normal network whose reciprocal pairs are all exactly equal (an undirected network given as a directed one) has an improper posterior for the correlation under the prior of the samplers, and is refused as ame refuses it (class "lame_improper_normal_reciprocity"): fit it with symmetric = TRUE, or with dcor = FALSE.

prior — NULL or a named list of priors. The estimators read one key, s20, the scale of the inverse-gamma prior of the variance of the noise of a Poisson or a normal fit (one degree of freedom on s20; the default is 1 for counts and the variance of the observed responses for a normal response, the priors ame and lame draw it under). The other keys of the samplers' prior list have no counterpart in the estimators; they are reported and ignored. The prior of the multiplicative effects of a directed network is the samplers' default (ame, Suv0 and kappa0).

Value

An object of class "ame_als"; see ame_als, including the per-start fields (multistart_coefficients, multistart_product_distance and the others) that a fit with multistart records.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

References

Minhas, S. and Hoff, P. D. (2025). Decomposing Network Dynamics: Social Influence Regression. *Political Analysis*. The iterative block coordinate descent estimator adapted here originates with that work (implemented in sir::sir_alsfit()).

See Also

ame_als, ame_als_bootstrap, lame for the full MCMC estimator, als_dynamic_beta for a regression-only smoother that estimates *only* a time-varying β_t (no a, b, U, V; not a special case of this function).

Examples

```
Y <- replicate(4, { m <- matrix(rnorm(400), 20, 20); diag(m) <- NA; m },
               simplify = FALSE)
fit <- lame_als(Y, R = 1, family = "normal", verbose = FALSE)
coef(fit)
```

---

lame_multi — *Multi-panel lame() with shared coefficients*

Description

Fits K independent lame() models (one per panel) and pools the per-panel beta posteriors into a precision-weighted shared posterior. Returns a list with the per-panel fits, the pooled beta posterior, and the panel-specific deviations.

Usage

```
lame_multi(Y_list, Xdyad_list, ...)
```

Arguments

`Y_list`	A list of length K, each element a list (or 3-D array) of T per-panel network observations.
`Xdyad_list`	A list of length K, each element a list of T dyadic covariate arrays.
`...`	Arguments forwarded to `lame()` (e.g. family, R, mode, nscan, burn, odens, dynamic_beta, dynamic_beta_kind). The pooled beta posterior is always returned, per-period when `dynamic_beta` is active (detected from the panel fits).

Details

This is an R-level wrapper: it fits each panel with its own MCMC and pools the results afterwards. The pooling is exact when the panels are conditionally independent given beta, which is the standard assumption.

Value

A list with

- `fits`: list of K per-panel `lame` fits.
- `beta_shared`: pooled posterior mean of beta (per-period when dynamic).
- `beta_deviations`: list of K panel-specific deviations from `beta_shared`.
- `K`: number of panels.

Class `"lame_multi"`.

See Also

`lame_parallel` for the unrelated multi-*chain* wrapper that runs K MCMC chains of the *same* model (used for R-hat / ESS diagnostics and pooled effective sample size). `lame_multi` is for K *distinct* panels with shared regression coefficients; `lame_parallel` is for K chains of one model.

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit_multi <- lame_multi(
  Y_list = list(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)),
                lapply(YX_bin_list$Y, function(y) 1 * (y > 0))),
  Xdyad_list = list(YX_bin_list$X, YX_bin_list$X),
  family = "binary", R = 0,
  nscan = 100, burn = 25, odens = 5, verbose = FALSE, chains = 1)
dim(fit_multi$beta_shared)
```

lame_parallel *Run LAME (longitudinal AME) with multiple parallel chains*

Description

Thin wrapper around ame_parallel that forces fitter = "lame". Use this for longitudinal data; in particular, multi-chain dynamic_beta / dynamic_uv / dynamic_ab fits are reached through here. The combined fit's $BETA is a 3-D array when dynamic_beta is on, and posterior::summarise_draws(as_draws(fit), "rhat") gives R-hat across chains for every per-period coefficient.

Usage

```
lame_parallel(
  Y,
  n_chains = 4,
  cores = n_chains,
  combine_method = c("pool", "list"),
  ...
)
```

Arguments

Y	Longitudinal network: list of T relational matrices, or a 3-D array [n, n, T].
n_chains	Number of parallel chains (default 4).
cores	CPU cores to use (default n_chains; 1 = sequential).
combine_method	"pool" (default) or "list".
...	Additional arguments forwarded to lame, including dynamic_beta, dynamic_uv, dynamic_ab, family, nscan, burn, odens, etc.

Value

Same as ame_parallel: a combined lame fit (when combine_method = "pool") or a list of fits.

Examples

```
data(YX_bin_list)
fit_pll <- lame_parallel(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), Xdyad = YX_bin_list$X,
                         family = "binary", n_chains = 2, cores = 1,
                         nscan = 50, burn = 10, odens = 5,
                         dynamic_beta = "dyad", verbose = FALSE)
dim(fit_pll$BETA)        # [iter * n_chains, p, T]
fit_pll$chain_indicator  # length iter * n_chains
if (requireNamespace("posterior", quietly = TRUE)) {
  posterior::summarise_draws(posterior::as_draws(fit_pll))
}
```

lame_resume *Resume a* lame() *MCMC run from a checkpoint*

Description

For a fit started with `lame(..., checkpoint_path = "X.rds")` that terminated early (either via `max_seconds` or an external interruption), continues the chain from the most recent checkpoint. The checkpoint holds the full sampler state (latent matrix, coefficients, additive and multiplicative effects, variance components, dynamic coefficient paths, auxiliary states and adapted proposal scales) together with the seed and the original call. The continuation re-invokes `lame()` with that state as `start_vals`, completing any unfinished burn-in, and with `seed` advanced by the number of completed iterations, so it does not replay the first run's random stream. The result is a fresh fit whose stored draws start where the checkpointed chain stopped; draws stored before the interruption are not carried over. `nscan_more` requests additional production sweeps; the saved thinning setting `odens` determines how many draws are retained.

Usage

```
lame_resume(path, nscan_more = NULL, ..., .envir = NULL)
```

Arguments

`path`	Checkpoint RDS file path, as written through `checkpoint_path`, or a fitted `lame` / `ame` object (for instance the `fit` carried by a `lame_chain_unsettled` error). A fit is continued from the state it stopped in, with the call that produced it, `burn = 0` and the seed advanced by the sweeps it ran. For a pooled multi-chain fit every chain is continued from its own final state, and the chains are checked and compared again (see `ame`); a single chain of a refused call is continued with `lame_resume(e$fits[[k]])`.
`nscan_more`	Optional integer, the new `nscan` for the continuation. If `NULL`, the original `nscan` is used.
`...`	Additional arguments forwarded to `lame()` (override the saved values).
`.envir`	Environment in which to evaluate the saved call's data arguments. Defaults to the caller's frame; used internally when `lame()` forwards a resume.

Details

Equivalent consolidated entry point. `lame(resume_from = path, ...)` short-circuits to this function with the user-supplied overrides forwarded; pass `nscan = K` on the resume call to request `K` additional post-burn sweeps. Both call shapes evaluate saved data arguments in the caller's frame, so objects referenced by the original call must remain available there.

Use `nscan_more = K` to override the `nscan` value to `K` for the continuation. A fit continued from the object runs with the arguments of its call, `max_seconds` included: a fit that stopped at its time limit stops again at the same limit unless `max_seconds = Inf` is passed. The continuation holds `nscan_more / odens` draws per chain (`chains * nscan_more / odens` pooled) when it runs to the end, plus one block of each extension. Other arguments can be overridden by passing them to

. . . . Older checkpoints without the complete transition state are treated as warm starts with a fresh burn-in and a warning. A `start_vals`, `burn` or `seed` passed there replaces the checkpoint's choice.

Tempered checkpoints restore every replica and the complete primary and auxiliary RNG streams, completing any unfinished original burn-in. The returned fit contains only newly retained draws, which exactly continue an uninterrupted run. Data, priors, seed, thinning and sampler changes are refused; start a new fit to change the model. Output settings, time limits and the number of additional production sweeps may be changed.

Checkpoints retain resolved priors independently of the original call. Where the checkpoint type permits a `prior` override, it can be supplied explicitly. Otherwise a continuation uses those settings even if a caller variable or a package default has changed. The original user settings are retained separately for training-window refits. Complete older Gaussian bipartite AR(1) checkpoints without frozen priors retain their former persistence defaults (mean 0.9, SD 0.1) when the saved call did not supply them.

Value

A fitted `lame` object.

Examples

```
ck <- tempfile(fileext = ".rds")
data(YX_bin_list)
# short run with very-aggressive max_seconds to force early termination
fit1 <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X,
             family = "binary", R = 0, nscan = 5000, burn = 50, odens = 5,
             checkpoint_path = ck, checkpoint_every = 50L,
             max_seconds = 0.5, verbose = FALSE)
if (isTRUE(fit1$terminated_early)) {
  # each chain wrote its own checkpoint (ck with _chain1, _chain2 before
  # the extension): continue one chain from its file, or every chain
  # from the returned fit
  fit2 <- lame_resume(sub(".rds", "_chain1.rds", ck, fixed = TRUE), nscan_more = 200)
  dim(fit2$BETA)
  # the time limit of the call applies again unless it is lifted
  fit3 <- lame_resume(fit1, nscan_more = 200, max_seconds = Inf)
  nrow(fit3$BETA)  # 2 chains * 200 / 5 = 80
}
```

`lame_robust` *Robust longitudinal models for directed continuous networks*

Description

Fit a longitudinal additive and multiplicative effects model with a residual precision shared across all periods of each directed pair. This permits persistent differences in volatility between pairs. All mean components and variance parameters are estimated jointly by MCMC.

Usage

```
lame_robust(
  Y,
  Xdyad = NULL,
  R = 2L,
  df = 4,
  burn = 5000L,
  nscan = 20000L,
  odens = 20L,
  seed = NULL,
  prior = list(),
  verbose = interactive(),
  actor_df = Inf,
  observation_df = Inf
)
```

Arguments

Y	List of at least two equally sized square numeric matrices. Rows are senders, columns are receivers, and list order is temporal order with equally spaced periods. Missing outcomes are allowed; diagonals are ignored. The actor set must be constant. When supplied, unique actor and period names are used to align inputs.
Xdyad	Optional list of sender-by-receiver-by-covariate numeric arrays, one per period. A matrix denotes a single covariate. Covariates must be finite off the diagonal, including for missing outcomes. No intercept is inserted into this design; the model contains a separate time intercept.
R	Nonnegative integer factor rank, smaller than the number of actors.
df	Degrees of freedom greater than two for the persistent pair precision prior. The default is four. Use `Inf` to remove persistent pair heterogeneity. With `observation_df = Inf`, this gives a homogeneous Gaussian residual model with the same mean specification and other priors.
burn	Number of initial iterations discarded.
nscan	Number of iterations after burn-in; must be a multiple of `odens`.
odens	Retain every `odens`-th iteration. Every retained iteration includes the complete joint state needed for prediction and simulation.
seed	Optional nonnegative integer sampler seed. An explicit seed preserves the caller's RNG state. With `NULL`, a seed is drawn from the caller's stream and recorded in the result.
prior	Named list overriding the proper priors described below.
verbose	Logical; print progress every 5,000 iterations.
actor_df	Degrees of freedom greater than two for independent sender-period and receiver-period innovation precisions. `Inf` (default) retains Gaussian actor innovations. A finite value permits periods with unusually large changes in sender or receiver activity.

`observation_df` Degrees of freedom greater than two for independent observation precisions. `Inf` (default) adds no temporary scale; finite values permit isolated residual shocks within otherwise stable pairs.

Details

For off-diagonal observations, the conditional model is

$$y_{ijt} = x_{ijt}^T\beta + \mu_t + a_{it} + b_{jt} + u_{it}^T v_{jt} + e_{ijt}, \qquad e_{ijt} \mid v_e, \lambda_{ij}, \omega_{ijt} \sim N(0, v_e/(\lambda_{ij}\omega_{ijt})).$$

For finite `df`, independent directed-pair precisions have $\lambda_{ij} \sim \mathrm{Gamma}(df/2, df/2)$ in shape-rate parameterization and persist through observed and forecast periods. With `observation_df = Inf`, each residual has a Student distribution after integrating out the pair precision; residuals for the same pair share that random scale across time. Opposite directions have independent precisions. For `df = Inf`, all pair precisions equal one.

For finite `observation_df`, independent precisions $\omega_{ijt} \sim \mathrm{Gamma}(observation_df/2, observation_df/2)$ add a temporary residual scale to each observation. Otherwise these precisions equal one. Conditional on the pair precision, integrating out the temporary precision gives a Student residual. When both degrees of freedom are finite, the fully marginal residual is a mixture over both scales and is not a single Student distribution. Replication and forecasting draw fresh temporary precisions; the persistent pair precision is retained.

The time intercept follows a zero-mean stationary AR(1). Sender and receiver effects each equal a persistent actor level plus an AR(1) deviation (stationary Gaussian when `actor_df = Inf`). Levels and deviations sum to zero across actors in every period. Sender and receiver levels have an estimated bivariate covariance; their deviations have independent innovations with a shared variance and AR coefficient. Factor coordinates follow independent zero-mean stationary AR(1) processes, sharing an innovation variance and AR coefficient. With finite `actor_df`, each actor role and period has an independent precision $\tau_{rt} \sim \mathrm{Gamma}(actor_df/2, actor_df/2)$. Conditional on this precision, centered actor innovations have covariance $(v_{ab}/\tau_{rt})(I - 11^T/n)$. The initial deviation uses covariance $v_{ab}(I - 11^T/n)/[\tau_{r1}(1 - \rho_{ab}^2)]$. This specifies the initial distribution for the finite panel; it is not asserted to be the stationary distribution of the Student-innovation process. Future periods draw new role precisions, while actor levels persist.

There are three AR coefficients: for the intercept, actor deviations, and factors. Each has a normal prior with mean 0.8 and standard deviation 0.3, truncated to (-0.995, 0.995).

Let `v` be the variance of observed off-diagonal responses, resolved once before sampling (one if that variance is undefined or nonpositive). The prior overrides are:

`ve_shape`, `ve_rate` Residual variance inverse-gamma shape and rate, defaulting to 2 and `v`. The density is proportional to $x^{-a-1}\exp(-b/x)$. The shape must exceed one.

`ar_shape`, `mu_rate`, `ab_rate`, `uv_rate` Inverse-gamma priors for intercept, actor-deviation, and factor innovation variances. The common shape defaults to 2 and must exceed one. Rates default to `0.1 * v`, `0.1 * v`, and `0.1 * sqrt(v)`, respectively.

`level_df`, `level_scale` Inverse-Wishart prior for the bivariate actor-level covariance. Defaults are 4 degrees of freedom and `v * I`. Degrees of freedom must exceed three; scale must be positive definite. The density is proportional to $|S|^{-(\nu+3)/2}\exp\{-\mathrm{tr}(\Psi S^{-1})/2\}$.

`beta_precision` Positive scalar or one value per covariate for independent zero-mean Gaussian coefficient priors. Default `0.01 / v` gives variance `100 * v`, in the unit of the response and independently of the residual variance.

All rates must be positive. The defaults are written in the unit of the response (a fit of `c * Y` is the fit of `Y` in other units), so only the units of the covariates matter for the priors; inspect `fit$prior` when comparing specifications.

This estimator supports directed continuous networks with stable actors. It does not estimate residual reciprocity, changing membership, bipartite structure, binary responses, dynamic covariate coefficients, offsets, or alternative latent laws of motion. It has its own `lame_robust` class and does not use `lame()`'s priors or inherit its fit class.

Run at least two independently seeded fits and pass them to `compute_mcmc_diagnostics()`. Diagnostics cover all retained identified parameters, including every off-diagonal latent product. Raw factor coordinates are rotation dependent. `simulate.lame_robust()` compares replicated networks with the observed statistics relevant to the analysis.

Value

An object of class `lame_robust`, with `draws`, aligned `Y` and `Xdyad`, resolved `prior`, `seed`, and sampling settings. In `draws`, `beta`, `mu`, and `hyper` have iterations in rows; `A` and `B` are actor-by-period-by-iteration arrays; `a0` and `b0` are actor-by-iteration matrices; `U` and `V` are actor-by-rank-by-period-by-iteration arrays; `lambda` is sender-by-receiver-by-iteration. With finite `actor_df`, `tau_ab` is role-by-period-by-iteration (sender, receiver). With finite `observation_df`, `omega` is sender-by-receiver-by-period-by-iteration. All components share the same iteration order. Methods include prediction, simulation, coefficient summaries, posterior covariance, credible intervals, and observation count.

See Also

`predict.lame_robust()`, `compute_mcmc_diagnostics()`, `lame()`

Examples

```
set.seed(21)
Y <- lapply(1:3, function(t) {
  y <- matrix(rnorm(25), 5, 5)
  diag(y) <- NA
  y
})
# Short execution example; use longer independent chains for inference.
fit <- lame_robust(Y, R = 0, burn = 10, nscan = 20, odens = 2,
                   seed = 1, verbose = FALSE)
dim(predict(fit, h = 1))
dim(simulate(fit, nsim = 2, seed = 2))
```

`lame_snap_als` *Removed: fast snap-shift ALS estimator*

Description

`lame_snap_als()` fitted a fast point estimate of the snap-shift dynamic latent-factor model. It was removed in lame 1.4.0 and now stops with a message: on simulated snap panels its per-period latent product tracked the generating one less closely than the `"ar1"` kind of dynamic ALS (correlation 0.72 against 0.80 on directed panels, 0.24 against 0.67 on bipartite and 0.44 against 0.72 on symmetric ones), and holding its drift at the `"ar1"` kind's smoothing, which closed that gap, missed planted resets (both ranked first in 15 of 20 panels against 20 of 20).

Fit snap-shift positions with `lame` and `dynamic_uv = TRUE, dynamic_uv_kind = "snap"` (the MCMC, directed panels). For bipartite and symmetric panels, or for a point estimate, use `dynamic_uv_kind = "ar1"` with `method = "als"` or the MCMC.

Usage

```
lame_snap_als(...)
```

Arguments

`...`	ignored: every call stops with the message above. A call with `mode = "bipartite"`, `R_row` / `R_col`, or `symmetric = TRUE` gets the message for that kind of panel.

Value

Does not return.

See Also

`lame`

`latent_positions` *Extract latent positions as a tidy data frame*

Description

Extracts multiplicative latent factor positions (U and V) from a fitted `ame`, `lame` or `ame_als` model and returns them as a tidy data frame suitable for plotting and analysis, with posterior standard deviations when posterior draws of the positions are available.

Usage

```
latent_positions(object, ...)

## S3 method for class 'ame'
latent_positions(object, align = FALSE, ...)

## S3 method for class 'lame'
latent_positions(object, align = TRUE, ...)

## S3 method for class 'ame_als'
latent_positions(object, align = FALSE, ...)
```

Arguments

object	A fitted ame, lame or ame_als model object with R > 0.
...	Additional arguments (currently unused).
align	Logical. For dynamic ame_als fits, apply Procrustes alignment across the time periods of the fit. It has no effect on ame fits (a single period) or on lame fits, whose dynamic positions are aligned when the model is fitted (see Details).

Details

The latent product of each period is identified, the positions only up to a rotation or sign change that leaves the product unchanged. The positions of a lame fit with dynamic_uv = TRUE therefore factor the posterior mean product of every period at the fitted rank: $U_t V_t'$ with the scale split evenly between U_t and V_t; $U_t L U_t'$ for symmetric fits, with fit$L fixed across periods; and $U_t G V_t'$ for bipartite fits, with a diagonal fit$G fixed across periods. Each period is matched to the one before by moves that keep its product. The posterior standard deviations come from the saved draws (posterior_opts = list(save_UV_draws = TRUE)), each draw factored the same way and matched onto the reported positions.

Alignment only ever happens across the periods of one fit. Nothing here aligns two different fits (two chains, or the same model with and without a covariate) to a common orientation, so their latent positions are not directly comparable. To compare fits that way, stack their U matrices into an [n, R, 2] array (same actors in the same order, same R) and call procrustes_align(U = that_array) on it.

Value

A data frame with columns:

actor Character. Actor name (from rownames of U or V).

dimension Integer. Latent dimension index (1 to R).

time Character. Time period label. Dynamic fits use the time labels from the input; static (cross-sectional) fits return "1" for every row so downstream filtering by time behaves the same in both cases.

value Numeric. The posterior mean latent position.

posterior_sd Numeric. Posterior standard deviation of the latent position, or NA if posterior draws are not available. To enable, fit the model with posterior_opts = posterior_options(save_UV = TRUE) (static positions) or posterior_opts = list(save_UV_draws = TRUE) (dynamic positions).

type Character. "U" for sender/row positions, "V" for receiver/column positions. Symmetric models have only "U".

Returns a zero-row data frame with correct column names if R = 0.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

procrustes_align for standalone Procrustes alignment, uv_plot for visualizing latent positions, posterior_options for enabling posterior sampling of U/V

Examples

```
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
           burn = 5, nscan = 5, odens = 1, verbose = FALSE, chains = 1)
lp <- latent_positions(fit)
head(lp)
```

---

lazegalaw *Lazega's law firm data*

Description

Several nodal and dyadic variables measured on 71 attorneys in a law firm.

Format

A list consisting of a socioarray Y and a nodal attribute matrix X.

The dyadic variables in Y include three binary networks: advice, friendship and co-worker status.

The categorical nodal attributes in X are coded as follows:

- status (1=partner, 2=associate)
- office (1=Boston, 2=Hartford, 3=Providence)
- practice (1=litigation, 2=corporate)
- law school (1=Harvard or Yale, 2=UConn, 3=other)

seniority and age are given in years, and female is a binary indicator.

Source

Originally available at http://moreno.ss.uci.edu/data.html#lazega (site no longer accessible).

`lfo` *Exact rolling-origin leave-future-out cross-validation*

Description

For a fitted `lame` object with T periods, refits the model on the first $t-1$ periods (for each t in `periods`) and computes the expected log predictive density (elpd) of period t under the refit. Returns the summed elpd across all leave-out periods, along with per-period contributions.

Usage

```
lfo(fit, periods = NULL, refit = TRUE, ...)
```

Arguments

`fit`	A fitted `lame` object.
`periods`	Integer vector of leave-out periods to evaluate. Default is the last 3 periods (`tail(seq_len(T), 3L)`). Each period t must satisfy $t \geq 2$.
`refit`	Logical; if `TRUE` (default), refits on the training window. If `FALSE`, scores each period under the original fit, which was estimated on every period including the one being scored, so the elpd is optimistic.
`...`	Passed to the refit `lame()` call (typically `nscan`, `burn`, `odens`, `verbose`).

Details

The one-step forecast is taken with the leave-out period's own dyadic covariates supplied as `newdata`. This matters because `predict(fit, h = 1)` called without `newdata` reuses the last fitted period's covariate array, which scores the held-out period against the wrong covariates whenever they vary over time.

Each dyad is scored by its posterior predictive log density, $\log \frac{1}{S} \sum_s p(y \mid \eta_s)$, averaging the family density over the posterior draws of the forecast linear predictor rather than evaluating it once at the posterior-mean predictor. For normal and Poisson models, each predictor uses its matching residual variance draw. Poisson scoring integrates the lognormal observation effect and uses held-out exposure. Supported families are normal, binary and Poisson; symmetric networks score each unordered dyad once. Refits retain the original covariate roles, selected dynamic coefficients and user priors, recomputing empirical defaults on training data only.

Value

A list with `elpd_lfo` (total summed elpd), `pointwise` (per-dyad log-density at each leave-out period; a list of numeric vectors, one per period – `unlist(pointwise)` gives a flat vector suitable for `loo::loo_compare()`-style stacking), `p_lfo` (`NA`; an effective parameter count is not estimated by this rolling-origin calculation), `per_period` (data frame with `period`, `elpd`, `n_obs`), and `periods` (the periods evaluated).

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X, family = "binary", R = 0,
            dynamic_beta = "dyad",
            nscan = 60, burn = 15, odens = 5, verbose = FALSE, chains = 1)
lfo_res <- lfo(fit, periods = 4L, refit = TRUE,
               nscan = 50, burn = 10, odens = 5, verbose = FALSE)
print(lfo_res)
```

---

list_to_array *Convert list to array*

Description

Convert list to array

Usage

```
list_to_array(actors, Y, Xdyad, Xrow, Xcol)
```

Arguments

actors	vector of actors
Y	dv in list format
Xdyad	dyadic covariates in list format
Xrow	sender covariates in list format
Xcol	receiver covariates in list format

Value

transforms Y, Xdyad, Xrow, and Xcol to arrays

Author(s)

Shahryar Minhas

Examples

```
n <- 12
actors <- paste0("a", 1:n)
Y <- lapply(1:3, function(t) {
  m <- matrix(rnorm(n * n), n, n)
  dimnames(m) <- list(actors, actors)
  m
```

```
})
names(Y) <- paste0("t", 1:3)
out <- list_to_array(actors, Y, Xdyad = NULL, Xrow = NULL, Xcol = NULL)
dim(out$Y)
```

`logLik.ame` *Log-likelihood is not directly exposed for ame() / lame() fits*

Description

`ame()` and `lame()` produce a posterior sample, not a maximum- likelihood point. A pointwise log-likelihood is computable from the posterior draws but is not stored on the fit object, so `logLik()` (and the AIC / BIC generics that dispatch through it) error out informatively rather than return a misleading number.

Usage

```
## S3 method for class 'ame'
logLik(object, ...)

## S3 method for class 'lame'
logLik(object, ...)
```

Arguments

`object`	an `ame` or `lame` fit.
`...`	ignored.

Details

For Bayesian model comparison use posterior-predictive checks via `gof` / `gof_plot`, or compute WAIC / LOO yourself from the per-draw log-likelihoods (e.g. via the **loo** package on the BETA / VC chains).

Value

Never returns; raises an error.

logLik.ame_als *Log-likelihood is not defined for a fast AME fit*

Description

The fast estimator minimises a (working-response) least-squares objective, not a family likelihood, so it has no log-likelihood and `AIC` / `BIC` are undefined. Calling `logLik()` raises an informative error rather than returning a misleading number.

Usage

```
## S3 method for class 'ame_als'
logLik(object, ...)
```

Arguments

`object`	an `ame_als` object.
`...`	ignored.

Details

An empirical-Bayes fit (the default for normal, binary and Poisson responses) carries a Laplace approximation, at its posterior mode, to its log marginal likelihood, `fit$eb$log_marginal` (see **Choosing the rank** in `ame_als`), which compares fits of different latent ranks.

Value

Never returns; raises an error.

loo *Generic dispatcher for loo / waic on ame / lame fits*

Description

Lightweight S3 generics so calls of the form `loo(fit)` dispatch through R's S3 system even when the **loo** package is not loaded. When **loo** *is* loaded, its generic resolves first; this fallback only fires for bare-namespace use.

Usage

```
loo(x, ...)

waic(x, ...)
```

Arguments

x	a fitted ame or lame object with $log_lik.
...	passed to the relevant method.

Value

A loo / waic object.

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, R = 1, dcor = FALSE, nscan = 100, burn = 5, odens = 1,
           save_log_lik = TRUE, verbose = FALSE, chains = 1)
if (requireNamespace("loo", quietly = TRUE)) {
  # a fit this small can trip loo's own low-sample Pareto-k diagnostic;
  # that diagnostic warning is expected here and not specific to lame
  print(suppressWarnings(loo(fit)))
}
```

---

loo.ame *Approximate leave-one-out cross-validation for AME / LAME fits*

Description

S3 method for loo that uses the per-iteration pointwise log-likelihood stored on the fit object (fit$log_lik) when the model was fit with save_log_lik = TRUE. Returns the standard loo object with Pareto-k diagnostics.

Usage

```
## S3 method for class 'ame'
loo(x, ...)

## S3 method for class 'lame'
loo(x, ...)

## S3 method for class 'ame_als'
loo(x, ...)
```

Arguments

x	A fitted ame or lame object that has $log_lik.
...	Additional arguments forwarded to loo::loo.matrix (e.g. cores, r_eff).

Details

Observed likelihood factors. LOO and WAIC require a current `fit$log_lik_contract` verifying observed-data conditional likelihood factors. Latent augmented scores, approximate rank/cap diagnostics and legacy fits without this contract are rejected. Directed cell marginals do not factorize under nonzero or estimated residual dyad correlation; joint dyad factors would be needed. Symmetric fits store each unordered dyad once. Correct marginal predictive scores remain useful and readable through `read_log_lik()` even when LOO or WAIC is unavailable.

Chunked log-lik portability. When fit with `save_log_lik = "chunked"`, the on-disk chunk files default to `tempdir()`, which is cleared at the end of the R session. If you intend to `saveRDS()` the fit and reload it in a fresh session, supply an explicit persistent `log_lik_path` (e.g. `"./loglik_chunks"`) so the chunks survive the round trip.

Value

A `loo` object.

Examples

```
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 0, dcor = FALSE,
           nscan = 60, burn = 15, odens = 5,
           save_log_lik = TRUE, verbose = FALSE, chains = 1)
if (requireNamespace("loo", quietly = TRUE)) {
  loo_res <- loo::loo(fit)
  print(loo_res)
}
```

---

mhalf *Symmetric square root of a matrix*

Description

Computes the symmetric square root of a positive semidefinite matrix

Usage

```
mhalf(M)
```

Arguments

`M`	a positive semidefinite matrix

Value

a matrix `H` such that `H^2` equals `M`

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
M <- matrix(c(4, 1, 1, 4), 2, 2)
H <- mhalf(M)
H %*% H  # recovers M
```

---

nobs.ame *Number of observed dyads in an AME / LAME fit*

Description

Tempered symmetric binary fits count each observed unordered dyad-period once, matching their stored likelihood. Other fits count finite (non-missing) cells in Y, summed over time slices for longitudinal fits. Useful as a denominator for sample-size reporting and as the basis for AIC/BIC *if* you also have a log-likelihood (which ame() / lame() do not directly expose; see logLik.ame).

Usage

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

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

Arguments

object	an ame or lame fit.
...	ignored.

Value

Integer: number of observed dyads in Y.

nobs.ame_als *Number of observed dyads in an ame_als fit*

Description

Mirrors nobs.ame so the MCMC and ALS fits expose the same accessor.

Usage

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

Arguments

object	an ame_als fit.
...	ignored.

Value

Integer: number of observed dyads.

nodematch *ERGM-style covariate helpers for ame() / lame()*

Description

nodematch(x) returns an n x n matrix with 1 where x[i] == x[j] and 0 otherwise – the AME analogue of ERGM's nodematch term, suitable for Xdyad as a single homophily covariate. absdiff(x) returns the absolute difference |x[i] - x[j]|, the AME analogue of ERGM's absdiff. nodefactor(x) returns the dyadic matrix of x[i] + x[j] for a numeric x (or, for a factor / character x, a named list with one matrix per level giving the dyadic co-membership count (x[i] == level) + (x[j] == level), with entries 0, 1, or 2, the ERGM nodefactor semantics).

Usage

```
nodematch(x, na_diag = TRUE)

absdiff(x, na_diag = TRUE)

nodefactor(x, na_diag = TRUE)
```

Arguments

x	a numeric, factor, or character vector of length n.
na_diag	logical: set the diagonal to NA? Defaults to TRUE.

Details

All helpers return a matrix *or* list of matrices that is ready to wrap into an n x n x p Xdyad array via simplify2array or array().

Value

An n x n numeric matrix (or, for a factor x passed to nodefactor, a named list of such matrices).

Examples

```
n <- 12
grp <- sample(letters[1:3], n, replace = TRUE)
age <- rnorm(n, 40, 10)
# build a single same-group homophily covariate
Xdyad <- array(nodematch(grp), dim = c(n, n, 1),
               dimnames = list(NULL, NULL, "same_group"))
# combine homophily + age difference
Xdyad <- array(c(nodematch(grp), absdiff(age)),
               dim = c(n, n, 2),
               dimnames = list(NULL, NULL, c("same_group", "age_diff")))
```

---

per_actor_slopes *Post-MCMC per-actor time-varying slopes*

Description

Computes a smoothed per-actor time-varying slope coefficient on a slice of the combined dyadic design array (fit$Xlist). For kind = "row", each row-actor i gets a length-T slope path $\beta_{i,t}$ on the across-column mean of design slice covariate_idx, fit by ridge-penalised least squares on the response-scale residual Y - YPM (the observed value minus the posterior predictive mean) after the main MCMC components; kind = "col" uses the across-row mean for each column-actor.

Usage

```
per_actor_slopes(fit, kind = c("row", "col"), covariate_idx = 1L, lambda = 1)
```

Arguments

fit	A fitted lame object.
kind	"row" (default; per-row-actor slopes) or "col" (per-column-actor slopes).
covariate_idx	Integer index into the third (covariate) dimension of the combined dyadic design array fit$Xlist. Slice 1 is the intercept; dyadic covariates follow. Default 1L.
lambda	Non-negative smoothing parameter (first-difference penalty across periods). Default 1.

Value

A list with `slopes` (an $n_{actors} \times T$ matrix), `kind`, `covariate_idx`, `lambda`, and `label`. Class `"per_actor_slopes"`.

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X, family = "binary", R = 0,
            nscan = 100, burn = 25, odens = 5, verbose = FALSE, chains = 1)
# post-MCMC per-row-actor slopes on the first dyadic covariate
pas <- per_actor_slopes(fit, kind = "row", lambda = 1)
dim(pas$slopes)
```

`plot.ame` *Simple diagnostic plot for AME model fit*

Description

Provides a quick visual summary of an AME model fit, focusing on key convergence diagnostics. For more detailed diagnostics, use the specific plotting functions: trace_plot(), gof_plot(), ab_plot(), and uv_plot().

Usage

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

Arguments

`x`	an object of class "ame" from fitting an AME model
`...`	additional arguments passed to trace_plot()

Details

By default, this function simply calls trace_plot() to show MCMC trace plots and posterior distributions for key parameters. This provides a quick check of model convergence and mixing.

For more detailed visualizations, use the specialized functions:

trace_plot() MCMC diagnostics and posterior distributions

gof_plot() Goodness-of-fit assessment

ab_plot() Additive sender/receiver effects

uv_plot() Multiplicative latent factors

Value

A ggplot2 object from trace_plot() (invisibly)

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

ame, trace_plot, gof_plot, ab_plot, uv_plot

Examples

```
# Fit an AME model
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2, gof = TRUE,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE, chains = 1)

# Quick diagnostic plot (shows trace plots)
plot(fit)
```

---

plot.ame_als *Plot the convergence of a fast AME fit*

Description

Plots the block coordinate descent deviance/SSE history – a convergence diagnostic for an ame_als or lame_als fit. For substantive plots see uv_plot, ab_plot and gof_plot.

Usage

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

Arguments

x	an ame_als object.
...	further arguments passed to the underlying plot.

Value

x, invisibly.

plot.lame *Plot diagnostics for a LAME model fit*

Description

Creates diagnostic plots for a LAME (Longitudinal Additive and Multiplicative Effects) model, including MCMC diagnostics, parameter evolution over time, and longitudinal goodness-of-fit checks. This is the default plot method for LAME objects.

Usage

```
## S3 method for class 'lame'
plot(
  x,
  which = c(1, 2, 3, 4),
  time.points = NULL,
  ask = FALSE,
  pages = c("single", "multiple"),
  ...
)
```

Arguments

x	an object of class "lame" from fitting a LAME model
which	numeric or character vector specifying which plots to produce: 1 or "trace" = MCMC trace plots, 2 or "density" = posterior density plots, 3 or "gof" = longitudinal goodness-of-fit plots, 4 or "effects" = additive and multiplicative effects, 5 or "network" = network snapshots at selected times. Default is c(1,2,3,4) to show main diagnostic plots.
time.points	numeric vector of time points for network snapshots (only used if "network" in which). Default is c(1, middle, last).
ask	logical; if TRUE, user is prompted before each plot page
pages	character string specifying how to arrange plots: "single" = one combined page (default), "multiple" = separate pages for each plot type
...	additional arguments (currently not used)

Details

The function produces a multi-panel plot containing:

MCMC trace plots Shows mixing and convergence of key parameters

Posterior distributions Density plots of regression coefficients and variance components

Longitudinal GOF Time series of observed network statistics with posterior predictive intervals

Effects over time Evolution of additive effects across time periods (if applicable)

Network snapshots Visualization of network at selected time points

The plot adapts to the longitudinal structure:

- Shows temporal trends in network statistics
- Highlights composition changes if actors enter/exit
- Displays credible intervals for time-varying statistics

Value

NULL (invisibly). Plots are displayed as side effects.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

lame, trace_plot, gof_plot, ab_plot, uv_plot

Examples

```
# Create simple longitudinal network data
set.seed(6886)
n <- 10
nms <- paste0("n", 1:n)
Y_list <- list(
  matrix(rnorm(n * n), n, n, dimnames = list(nms, nms)),
  matrix(rnorm(n * n), n, n, dimnames = list(nms, nms))
)
diag(Y_list[[1]]) <- diag(Y_list[[2]]) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(Y_list, family = "normal",
            nscan = 50, burn = 10, odens = 1, verbose = FALSE, plot = FALSE, chains = 1)

# default combined plot
plot(fit)
```

---

posterior_options *Options for saving posterior samples during MCMC*

Description

Options for saving posterior samples during MCMC

Usage

```
posterior_options(save_UV = FALSE, save_ab = FALSE, thin_UV = 10, thin_ab = 10)
```

Arguments

save_UV	Logical; whether to save samples of U and V matrices (default FALSE)
save_ab	Logical; whether to save samples of additive effects a and b (default FALSE)
thin_UV	Integer; thinning interval for U/V samples (default 10)
thin_ab	Integer; thinning interval for a/b samples (default 10)

Value

List of posterior saving options, to be passed to the `posterior_opts` argument of `ame` or `lame`: `ame(Y, ..., posterior_opts = posterior_options(save_UV = TRUE))` or `lame(Y, ..., posterior_opts = posterior_options(save_UV = TRUE))`. With `save_UV = TRUE`, static fits store `U_samples` / `V_samples` (plus `G_samples` for bipartite fits) and `dynamic_uv` fits store the per-period trajectory in `U_draws` / `V_draws`; see `latent_positions` and `simulate_posterior` for summarising them. With `save_ab = TRUE`, static effects use `a_samples` / `b_samples` (the same draws in both for a symmetric fit, which has one effect per actor); longitudinal dynamic effects use actor-by-period-by-draw arrays `a_draws` / `b_draws`. `thin_ab` is measured in retained coefficient draws; `thin_ab = 1` keeps them aligned.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
opts <- posterior_options(save_UV = TRUE, thin_UV = 5)
opts
```

---

posterior_quantiles *Extract posterior quantiles for model components*

Description

Extract posterior quantiles for model components

Usage

```
posterior_quantiles(
  fit,
  component = c("beta", "UV", "ab"),
  probs = c(0.025, 0.5, 0.975)
)
```

Arguments

fit	Fitted ame model object
component	Character; which component: "beta", "UV", "ab". The additive effects "ab" come back as a list of the row (a) and column (b) quantiles; a symmetric fit has one effect per actor, so b equals a.
probs	Numeric vector of probabilities for quantiles

Value

Matrix or array of posterior quantiles

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
X <- array(rnorm(n * n), dim = c(n, n, 1))
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, Xdyad = X, R = 1, nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
posterior_quantiles(fit, component = "beta")
```

---

predict.ame *Predict method for AME models*

Description

Generate predictions from fitted AME models, including point estimates and predictive distributions.

Usage

```
## S3 method for class 'ame'
predict(
  object,
  newdata = NULL,
  type = c("response", "link", "distribution"),
  n_samples = 100,
  include_uncertainty = TRUE,
  ...
)
```

Arguments

object	Fitted AME model object
newdata	Optional dyadic covariates for prediction: a single 3-D array (n x n x p_dyad) whose slices match the dyadic covariates the model was fit with (same order; do not include the intercept or nodal-covariate slices). Nodal covariates cannot be changed at prediction time – their fitted contribution is reused. When omitted, the fitted design is used. Explicit ordinal response predictions average numeric category expectations over matched predictor and cutpoint draws. Newdata predictions require saved draws for enabled additive and latent effects; disabled effects and zero latent rank require no such draws.
type	Character; type of prediction: • "response": predicted values on response scale (default) • "link": predicted values on link scale • "distribution": full posterior predictive distribution
n_samples	For type="distribution", number of posterior samples
include_uncertainty	Logical; include joint parameter uncertainty (default TRUE). Requires aligned additive and latent draws saved through posterior_options(save_ab = TRUE, save_UV = TRUE). With FALSE, only observation noise is simulated around the fitted mean predictor.
...	Additional arguments (not used)

Details

For MCMC Poisson models the response is the mean count of a cell, $\exp(\eta+\sigma^2/2)$, at the posterior mean η of its linear predictor and the posterior mean σ^2 of the residual variance (the model is $Y \sim \text{Poisson}(\exp(z))$, $z \sim N(\eta, \sigma^2)$). The stored YPM holds it, and newdata predictions use the same form at the new linear predictor. type = "distribution" draws counts from the posterior predictive distribution instead.

Value

Depending on type:

- "response"/"link": Matrix of predictions
- "distribution": Array of posterior predictive samples

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Fit model
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
```

```
            nscan = 100, burn = 10, odens = 1, verbose = FALSE, chains = 1)

# Point predictions
Y_pred <- predict(fit)

# Predictions on link scale
Y_link <- predict(fit, type = "link")
```

`predict.ame_als` *Predictions from a fast AME fit*

Description

Returns predictions from an `ame_als` or `lame_als` fit: the fitted linear predictor on the link scale (`type = "link"`) or the response scale (`type = "response"`, the default).

Usage

```
## S3 method for class 'ame_als'
predict(
  object,
  newdata = NULL,
  type = c("response", "link"),
  plug_in = NULL,
  ...
)
```

Arguments

`object`	an `ame_als` object.
`newdata`	optional dyadic covariate array matching `object$X` (`[n_row, n_col, p, T]`); a matrix or 3D array is coerced.
`type`	`"response"` (default) or `"link"`.
`plug_in`	logical; for an empirical-Bayes fit, return the plug-in values (`TRUE`: the link at the fitted predictor, the posterior mean) or the predictive ones that integrate the link over the uncertainty of the predictor (`FALSE`). The default is `TRUE` for a Poisson fit and `FALSE` for a binary fit.
`...`	ignored.

Details

With `newdata = NULL` the training-data fitted values are returned. Supplying `newdata` – a dyadic covariate array with the *same* actors and dimensions as the fitted design – substitutes the dyadic covariate contribution while holding the intercept, additive effects and multiplicative term fixed; it predicts counterfactual dyadic covariates for the same network, not out-of-sample actors.

For a binary empirical-Bayes fit (the default) the response-scale predictions are predictive probabilities: the response integrated over the posterior of the linear predictor, $\Phi(m/\sqrt{1+v})$ for a probit link, rather than applied at its mean (see **Predictions** in `ame_als`). For a Poisson fit they are the plug-in mean counts $e^{m+v_e/2}$, as the samplers report them: the posterior predictive mean of a count is not finite in this model (the inverse-gamma tail of v_e, the inverse-Wishart tail of the scale of the factors for a cell the data do not inform), and the plug-in mean is finite everywhere. `plug_in = FALSE` returns, for a count, the mean that integrates the predictor, $e^{m+(v+v_e)/2}$, and `plug_in = TRUE`, for a binary fit, the link applied to the fitted predictor, which is what `fit$fitted` and `simulate()` hold. `type = "link"` returns the posterior mean m of the predictor. A normal fit's response is its predictor plus Gaussian noise: its predictions are the posterior mean of the predictor, with or without `plug_in`. Predictions for a design given in `newdata` add the coefficients' contribution to the same posterior.

Value

A matrix (cross-sectional) or list of matrices (longitudinal).

`predict.lame` *Predict method for LAME models*

Description

Generate predictions from a fitted longitudinal AME model. Returns a list of matrices (one per time point) on the requested scale.

Usage

```
## S3 method for class 'lame'
predict(
  object,
  newdata = NULL,
  type = c("response", "link"),
  h = 0L,
  by_draw = FALSE,
  interval = c("none", "credible", "prediction"),
  probs = c(0.025, 0.975),
  newexposure = NULL,
  n_draws = NULL,
  seed = NULL,
  ...
)
```

Arguments

`object`	Fitted LAME model object.

`newdata` Optional dyadic covariate arrays. With `h = 0`, a list of `T` arrays (`[n_row, n_col, p]` each, same actors as the fit) for counterfactual in-sample predictions; `NULL` returns the training-data predictions. With `h > 0`, a list of `h` arrays for the future periods; `NULL` reuses the last fitted period's covariates, which scores the forecast against stale covariates whenever they vary over time.

`type` Character; `"response"` (default) or `"link"`.

`h` Integer >= 0: forecast horizon. When `h = 0` (default), returns in-sample predictions as before. When `h > 0`, propagates the AR(1) (or RW1) state-space model forward by `h` periods and returns a list of `h` matrices (one per future period). Requires at least one dynamic component on the fit. Warns when posterior ρ_β is near 1. Coefficient innovations retain the fitted design-based scale. Dynamic coefficient forecasts require equally spaced AR(1) or RW1 periods; irregular-period, RW2, Matern and actor-specific coefficient laws are refused until their future transition is implemented.

`by_draw` When `TRUE` and `h > 0`, returns an `n x n x h x n_draws` array of per-draw forecasts instead of per-period means.

`interval` One of `"none"` (default), `"credible"` or `"prediction"`. When `h > 0`, the per-period output is a list of length-3 lists with `$lower`, `$median`, `$upper` matrices computed at the `probs` quantiles across posterior draws. `"credible"` is the interval of the conditional mean $E[Y_{T+h} \mid \theta_{T+h}]$ given each propagated parameter state: it carries the posterior uncertainty in that state and adds no observation noise, so it is narrower than the spread of realised outcomes and undercovers them (on simulated 30-actor normal panels its nominal 95 percent interval covered 51 to 61 percent of held-out cells). `"prediction"` adds the outcome noise on top of that mean: a draw from $N(\eta, ve)$ per posterior draw and cell for `"normal"`, a count from $Poisson(e^z)$ with $z \sim N(\eta, ve)$ for `"poisson"`. For explicit ordinal models, credible intervals summarize numeric category expectations using matched saved cutpoints; prediction intervals summarize category draws including unit Gaussian observation noise. For `"binary"`, `"cbin"`, `"frn"` and data-induced ordinal models, this option retains the existing credible interval convention and reports that choice. Ignored for in-sample (`h = 0`) predictions. Outcome forecasts preserve symmetric and within-dyad observation dependence. Their unipartite diagonals are missing; bipartite diagonal cells are retained. They are refused for `dynamic_rho = TRUE`, which supplies no law for future reciprocity; conditional-mean forecasts remain available.

`probs` Length-2 vector of lower / upper quantiles for the interval when `interval` is `"credible"` or `"prediction"`. Default `c(0.025, 0.975)`.

`newexposure` Optional length-`h` non-negative numeric vector of future-period exposures (Poisson only). When omitted and the fit has `period_exposure` stored, defaults to the last observed exposure; when both are absent, defaults to 1.

`n_draws` Number of posterior draws to use when `h > 0`. Default `NULL` uses all stored draws. Ignored for in-sample (`h = 0`) predictions.

`seed` Optional RNG seed for the `h > 0` forecast draws, making forecasts reproducible. Ignored when `h = 0`.

`...` Additional arguments (not used).

Details

For h > 0 only $U_t V_t'$ is identified, so each posterior draw of the coordinates carries its own rotation and sign and the product of the posterior-mean coordinates falls well short of the posterior mean of the product (UVPM, or ULUPM on symmetric fits). Under the element-wise AR(1), $E[u_{T+h} v_{T+h}'] = \rho_{uv}^{2h} E[u_T v_T']$ off the diagonal, so the forecast carries UVPM at period T forward by ρ_{uv}^{2h} and each draw's propagated coordinates enter only as the deviation around that mean. Static positions are carried forward unchanged. Dynamic additive effects are propagated as their actor level plus an AR(1) deviation.

For MCMC Poisson models the response is the mean count of a cell in a period, $e_t \exp(\eta + \sigma^2/2)$, with e_t the exposure of the period (period_exposure; one without), η the posterior mean of the linear predictor and σ^2 the posterior mean of the residual variance. The stored YPM holds it in sample, newdata predictions use the same form at the new linear predictor, and a forecast point (h > 0) uses the means of the forecast linear predictor and of the variance over the forecast draws, and the exposure newexposure. With by_draw = TRUE the draws' own conditional means, $e \exp(\eta_s + \sigma_s^2/2)$, are returned and interval = "credible" summarizes them; with interval = "prediction" the summaries are of simulated counts.

Value

With interval = "none" and by_draw = FALSE, a list of prediction matrices (one per period, or per horizon when h > 0). With by_draw = TRUE, an [n_row, n_col, h, n_draws] array. With interval = "credible" or "prediction", a list of h lists holding $lower, $median and $upper matrices.

---

predict.lame_robust *Predict and simulate robust longitudinal networks*

Description

Posterior means and joint draws from a lame_robust() fit. Prediction uses the conditional mean; simulation also includes observation noise with the persistent directed-pair precision from the same posterior iteration.

Usage

```
## S3 method for class 'lame_robust'
predict(
  object,
  h = 0L,
  newdata = NULL,
  type = c("response", "link"),
  by_draw = FALSE,
  nsim = NULL,
  seed = NULL,
  ...
)
```

```
## S3 method for class 'lame_robust'
simulate(object, nsim = 1L, seed = NULL, h = 0L, newdata = NULL, ...)
```

Arguments

object	A lame_robust fit.
h	Nonnegative integer forecast horizon. Zero returns observed periods.
newdata	For a positive horizon with covariates, a list of one dyadic design array per future period. For one period, a matrix or array is also accepted. Actor and covariate names are aligned to the fit. All future covariates must be supplied and finite off the diagonal. Omit otherwise.
type	Either "response" or "link"; both denote the continuous mean.
by_draw	Logical; return joint posterior mean-network draws. Forecast draws include future latent innovations. The default posterior mean integrates those innovations analytically before averaging over draws.
nsim	Number of posterior draws. NULL uses every retained iteration. Simulation defaults to one. Smaller requests sample without replacement; larger requests sample with replacement. Selected indices are sorted.
seed	Optional seed preserving the caller's RNG state.
...	No additional arguments are supported.

Details

Future intercepts, actor deviations, and factors follow their sampled AR(1) laws. Actor innovations respect the sum-to-zero constraints. Finite actor_df draws a new precision for each actor role and forecast period. Persistent actor levels and pair precisions remain the same within each simulated trajectory. Simulation at h = 0 replicates observations conditional on the saved mean paths and persistent pair scales. Finite observation_df draws fresh temporary residual precisions for every replicated observation, including at h = 0; fitted observation precisions are not reused. Simulation at a positive horizon draws both future states and observation noise. Future latent uncertainty is included when by_draw = TRUE; its Monte Carlo average need not equal the analytically integrated default point forecast exactly.

Value

Prediction returns a sender-by-receiver-by-period array, or a sender-by-receiver-by-period-by-draw array when by_draw = TRUE. Simulation always returns the four-dimensional form. Diagonals are NA. Draw arrays carry a draw_index attribute identifying the joint posterior iteration. Singleton dimensions are preserved.

prediction_draws_long *Long-format draws of the linear predictor for marginaleffects-style use*

Description

Returns a long-format data frame with one row per (draw, i, j, period) combination, giving the per-draw linear predictor (or response-scale prediction) at each dyad and period. Only the regression coefficients vary across .draw: the additive (a, b) and multiplicative (U, V) effects are held at their posterior means, so the spread across draws reflects coefficient uncertainty only, not the full posterior of the linear predictor. Intended for **marginaleffects**- / **tidybayes**-style downstream summarisation: column names follow the .draw / .chain / .iteration / .value convention so that tidybayes::spread_draws() and marginaleffects::posterior_draws() auto-dispatch on the returned data frame. Actor and period names from fit$Y's dimnames are carried forward into the actor_i, actor_j, period_label columns.

Usage

```
prediction_draws_long(
  object,
  newdata = NULL,
  type = c("link", "response"),
  n_draws = 100L,
  seed = NULL
)
```

Arguments

object	A fitted lame object.
newdata	Optional list of T dyadic covariate arrays for counterfactual evaluation. Default NULL uses in-sample covariates.
type	One of "link" (default) or "response". For a Poisson fit the response of a draw is its mean count, $e_t \exp(\eta + \sigma^2/2)$, with the draw's own residual variance and the exposure of the period.
n_draws	Number of posterior draws to use. Default 100.
seed	Optional RNG seed.

Value

A long-format data frame with columns .chain, .iteration, .draw, period, period_label, i, j, actor_i, actor_j, .value. Returned as a tibble when the **tibble** package is available; otherwise a plain data frame. The .iteration column identifies the retained coefficient iteration; .draw numbers the requested draws, including repeated selections.

Examples

```
set.seed(1)
n <- 12; Tt <- 3
Y <- lapply(seq_len(Tt), function(t) {
  m <- matrix(rnorm(n * n), n, n); diag(m) <- NA; m
})
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(Y, R = 1, nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
draws <- prediction_draws_long(fit, n_draws = 10, seed = 1)
head(draws)
```

---

print.als_dynamic_beta

Print method for penalised ALS time-varying beta

Description

Print method for penalised ALS time-varying beta

Usage

```
## S3 method for class 'als_dynamic_beta'
print(x, digits = 4, ...)
```

Arguments

x	An als_dynamic_beta result.
digits	Number of significant digits.
...	Ignored.

Value

x, invisibly. Called for its side effect of printing a summary of the penalised ALS time-varying coefficient estimates.

print.ame *Print method for AME model objects*

Description

Displays a formatted summary of a fitted AME (Additive and Multiplicative Effects) model. This method provides a concise overview of model structure, parameter estimates, and goodness-of-fit statistics without generating new data.

Usage

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

Arguments

x	an object of class "ame" from fitting an AME model
...	additional arguments (not currently used)

Details

The print method displays:

- Model type (unipartite/bipartite, symmetric/asymmetric)
- Network dimensions and observation count
- Family and link function used
- Number of MCMC iterations
- Parameter counts for regression coefficients and latent factors
- Basic convergence diagnostics if available

Unlike simulate, this method only formats existing results for display and does not perform any new computations or data generation.

Value

Invisibly returns the input object (for method chaining)

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

summary.ame for detailed summaries, simulate.ame for generating new networks, predict.ame for predictions

Examples

```
# Fit model
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE, chains = 1)

# Display summary
print(fit)
```

`print.ame.sim` *Print methods for AME and LAME simulation objects*

Description

Print methods for AME and LAME simulation objects

Usage

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

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

Arguments

`x`	simulation object of class "ame.sim" or "lame.sim"
`...`	additional arguments (not used)

Value

the simulation object invisibly

Author(s)

Shahryar Minhas

print.ame_als *Print an ame_als object*

Description

Print an ame_als object

Usage

```
## S3 method for class 'ame_als'
print(x, digits = 4, ...)
```

Arguments

x	an ame_als object.
digits	number of digits to display.
...	ignored.

Value

x, invisibly.

print.boot_ame *Print bootstrap results for a fast AME fit*

Description

Print bootstrap results for a fast AME fit

Usage

```
## S3 method for class 'boot_ame'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	a boot_ame object.
digits	number of digits to display.
...	ignored.

Value

x, invisibly.

print.gof_temporal *Print method for gof_temporal output*

Description

Print method for gof_temporal output

Usage

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

Arguments

x	A gof_temporal result.
...	Ignored.

Value

x, invisibly. Called for its side effect of printing a summary of the temporal-trend posterior-predictive check.

print.lame *Print method for LAME objects*

Description

Provides a concise print output for fitted LAME models. When the fit has two or more dynamic effects active simultaneously (dynamic_uv + dynamic_ab + dynamic_beta in any combination), a compact joint table summarising the AR(1) hyperparameters of each dynamic block is printed instead of a separate paragraph per component. Disable the compact mode by passing compact = FALSE. A fit that stores no posterior draws (max_seconds elapsed before the first stored draw) prints its dimensions and says so.

Usage

```
## S3 method for class 'lame'
print(x, compact = TRUE, digits = 3, ...)
```

Arguments

x	an object of class "lame"
compact	logical: when TRUE (default), use a compact joint table for fits with 2+ dynamic components; set to FALSE to force the per-component long-form display.
digits	Number of digits to display in the compact table. Default 3.
...	additional arguments (not used)

Value

the lame object invisibly

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

---

print.lame_multi *Print method for lame_multi*

Description

Print method for lame_multi

Usage

```
## S3 method for class 'lame_multi'
print(x, digits = 3, ...)
```

Arguments

x	A lame_multi object.
digits	Number of significant digits.
...	Ignored.

Value

x, invisibly. Called for its side effect of printing a summary of the pooled multi-panel fit.

---

print.lfo_lame *Print method for lfo() results*

Description

Print method for lfo() results

Usage

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

Arguments

x	A lfo_lame object.
...	Ignored.

Value

x, invisibly. Called for its side effect of printing a summary of the leave-future-out cross-validation results.

---

print.per_actor_slopes *Print method for per_actor_slopes*

Description

Print method for per_actor_slopes

Usage

```
## S3 method for class 'per_actor_slopes'
print(x, digits = 3, ...)
```

Arguments

x	A per_actor_slopes result.
digits	Number of significant digits.
...	Ignored.

Value

x, invisibly. Called for its side effect of printing a summary of the per-actor time-varying slopes.

---

print.summary.ame *Print method for summary.ame objects*

Description

Prints a formatted summary of an AME model fit

Usage

```
## S3 method for class 'summary.ame'
print(x, digits = 3, ...)
```

Arguments

x	a summary.ame object
digits	number of digits to display (default: 3)
...	additional arguments (not used)

Value

the summary.ame object invisibly

---

print.summary.ame_als *Print a fast AME summary*

Description

Print a fast AME summary

Usage

```
## S3 method for class 'summary.ame_als'
print(x, digits = 4, ...)
```

Arguments

x	a summary.ame_als object.
digits	number of digits to display.
...	ignored.

Value

x, invisibly.

---

print.summary.lame *Print method for summary.lame objects*

Description

Prints a formatted summary of a LAME model fit

Usage

```
## S3 method for class 'summary.lame'
print(x, digits = 3, ...)
```

Arguments

x	a summary.lame object
digits	number of digits to display (default: 3)
...	additional arguments (not used)

Value

the summary.lame object invisibly

`prior_summary` *Print the priors used by an AME / LAME / ame_als fit*

Description

Bayesian-hygiene helper modelled on `rstanarm::prior_summary()`. Prints recorded prior settings and the coefficient-prior specification saved by the fitting build. Older fits without that specification are identified explicitly; their prior is not inferred from the current build. For `ame_als` fits it just states that the estimator is a frequentist point estimator and there are no priors.

Usage

```
prior_summary(object, ...)

## Default S3 method:
prior_summary(object, ...)

## S3 method for class 'ame'
prior_summary(object, ...)

## S3 method for class 'lame'
prior_summary(object, ...)

## S3 method for class 'ame_als'
prior_summary(object, ...)
```

Arguments

`object`	a fitted `ame`, `lame`, or `ame_als` object.
`...`	ignored.

Value

`object`, invisibly.

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
prior_summary(fit)
```

procrustes_align *Procrustes alignment of latent positions across time*

Description

Aligns dynamic latent positions across time periods to remove arbitrary rotational indeterminacy. This is essential for interpreting temporal trajectories of latent positions, since the latent space is only identified up to rotation at each time point.

Uses Procrustes rotation: at each time step, finds the orthogonal rotation matrix that best aligns the current sender coordinates to the preceding period, then applies it sequentially through the series.

Usage

```
procrustes_align(
  object = NULL,
  U = NULL,
  V = NULL,
  G = NULL,
  return_fit = FALSE,
  per_draw = FALSE,
  ...
)
```

Arguments

object	A fitted ame or lame object, or NULL if raw arrays are provided via U/V/G.
U	Optional 3D array [n, R, T] of sender latent positions. If object is provided and U is NULL, extracted from object$U.
V	Optional 3D array [n, R, T] of receiver latent positions. For unipartite asymmetric models, U and V share one rotation per period so U V' is unchanged. For bipartite models, aligned jointly with U through the G interaction matrix.
G	Optional [R_row, R_col] interaction matrix for bipartite models; it is returned unchanged, and each period's U G V' is preserved.
return_fit	Logical. If TRUE and object is provided, returns a modified copy of the fit object with aligned latent positions. Default FALSE.
per_draw	Logical. When TRUE, run Procrustes alignment per posterior draw rather than on the posterior-mean trajectory. This uses object$U_draws / object$V_draws on a dynamic fit, or object$U_samples on a static one (populated by posterior_opts = list(save_UV_draws = TRUE)) when present; otherwise falls back to mean-trajectory alignment and emits an informational note. On a bipartite fit each draw is aligned with its own interaction-matrix draw from object$G_samples, so every draw's U G V' is unchanged; without those draws per-draw alignment is an error. Per-draw alignment is the methodologically correct treatment – rotation indeterminacy is a per-draw property, not a per-mean one – but is more memory-hungry. Default FALSE.
...	Additional arguments (currently unused).

Details

For directed unipartite networks, U and V share one Procrustes rotation per period, found from the stacked positions, so the product U V' is unchanged. For symmetric networks the product is ULU' with a diagonal L, which only a sign change of a column of U leaves unchanged; each period takes the column signs that best match the period before it.

For bipartite networks, one G serves every period, so a period's product `U %*% G %*% t(V)` is unchanged only by transformations `R_U`, `R_V` with `t(R_U) %*% G %*% R_V = G`. In the singular-vector bases of G these are sign changes of a singular direction, applied to U and V together; each period takes the sign changes that best match the period before it, and G is returned unchanged.

If U is a 2D matrix (static model with a single time point), it is returned unchanged with an informational message.

Alignment is within one fit: each period of a single dynamic trajectory is rotated onto the period before it. Two different fits are never aligned to each other automatically. Because raw arrays are accepted, you can do that explicitly: stack the two fits' `U` matrices into an `[n, R, 2]` array (same actors in the same order, same `R`) and call `procrustes_align(U = that_array)`; the second slice comes back rotated onto the first.

Value

If `return_fit = FALSE` (default): a list with components `U` (aligned sender positions), `V` (aligned receiver positions, if applicable), and `G` (the interaction matrix, for bipartite).

If `return_fit = TRUE`: a copy of `object` with aligned latent positions replacing the originals.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

`latent_positions` for extracting aligned positions as a tidy data frame, `uv_plot` for visualizing latent positions

Examples

```
data(YX_bin_list)
# YX_bin_list$Y stores latent-scale values; threshold to 0/1 first
Y_bin <- lapply(YX_bin_list$Y, function(y) 1 * (y > 0))
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(Y_bin, Xdyad = YX_bin_list$X, R = 2,
            family = "binary", dynamic_uv = TRUE,
            burn = 5, nscan = 5, odens = 1,
            verbose = FALSE, chains = 1)
aligned <- procrustes_align(fit)
str(aligned$U)  # aligned 3D array [n, R, T]
```

rbeta_ab_bip_gibbs_cpp *Full bipartite Gibbs update for beta, a, b*

Description

Single C++ function replacing the bipartite beta/a/b update block in lame.R

Usage

```
rbeta_ab_bip_gibbs_cpp(
  Z,
  Xlist,
  UV_eff,
  a_current,
  b_current,
  s2,
  g_prior,
  va,
  vb,
  rvar,
  cvar,
  center = FALSE
)
```

Arguments

Z	3D array (nA x nB x T)
Xlist	List of T arrays (nA x nB x p)
UV_eff	nA x nB matrix (U*G*V' or U*V')
a_current	Current row effects (length nA)
b_current	Current column effects (length nB)
s2	Dyadic variance
g_prior	G-prior parameter
va	Row effect variance (diagonal element of Sab)
vb	Column effect variance (diagonal element of Sab)
rvar	Whether to update row effects
cvar	Whether to update column effects
center	Condition each actor margin to sum to zero.

Value

List with beta, a, b

read_log_lik *Read the per-iteration log-lik matrix back from on-disk chunks*

Description

For fits run with save_log_lik = "chunked", the per-iteration pointwise log-likelihood lives in per-column-chunk binary files instead of fit$log_lik. This helper reconstitutes the [n_stored, n_obs] matrix in memory by issuing one readBin per chunk. loo.lame() / waic.lame() call this transparently.

Usage

```
read_log_lik(x)
```

Arguments

x	A fitted ame/lame object with $log_lik_chunks (i.e. fit with save_log_lik = "chunked").

Value

The full [n_stored, n_obs] double matrix.

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1,
           save_log_lik = TRUE, verbose = FALSE, chains = 1)
ll <- read_log_lik(fit)
dim(ll)
```

reconstruct_EZ *Reconstruct EZ and UVPM matrices from AME model output*

Description

Helper functions to recover EZ (linear predictor) and UVPM (posterior mean of the multiplicative product) matrices. When the fit stores the quantity (UVPM on asymmetric fits, ULUPM on symmetric fits), the stored posterior mean is returned directly; otherwise it is reconstructed from the saved factors.

Note: EZ returns the linear predictor (η), not the response:

- For Gaussian: $EZ = \eta = \mu$ (identity link)

- For Poisson: $EZ = \eta = \log(\lambda)$ (can be negative)
- For Binary: $EZ = \eta$ = probit inverse of p (can be any real value) Use YPM for predictions on the response scale.

Usage

```
reconstruct_EZ(fit, X = NULL)

reconstruct_UVPM(fit)
```

Arguments

`fit`	Fitted AME model object
`X`	Covariate array (optional, will use `fit$X` if available)

Details

Fits that carry the stored posterior mean (`UVPM` or, for symmetric fits, `ULUPM`) have it returned as-is; older fit objects without one are rebuilt from the posterior factor means.

Value

Reconstructed matrix

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
EZ <- reconstruct_EZ(fit)
UVPM <- reconstruct_UVPM(fit)
dim(EZ)
```

`residuals.ame` *Extract residuals from AME model*

Description

Computes residuals as the difference between observed values and fitted values. For `type = "response"`, returns `Y - fitted(object)`. For `type = "pearson"`, returns response residuals scaled by the standard deviation the family assigns to a response with that fitted value μ (`fitted(object)`, which for a Poisson fit already carries the exposure):

family	variance
normal	σ_e^2, the posterior mean residual variance
binary, cbin	$\mu(1-\mu)$ (a cbin cell is a nomination with that marginal probability)
poisson	$\mu + \mu^2(e^{\sigma_e^2} - 1)$, the variance of a count when $y \sim \text{Poisson}(e^z)$ and $z \sim N(\eta, \sigma_e^2)$
ordinal	the variance of the category distribution at the posterior-mean linear predictor; with explicit cutpoints the poster
frn	not defined: a rank within its row has a variance that depends on the other cells of the row, and the call is an erro

A cell whose variance is zero has no standardised residual and is `NA`.

Usage

```
## S3 method for class 'ame'
residuals(object, type = c("response", "pearson"), ...)
```

Arguments

`object`	Fitted AME model object (class "ame").
`type`	Character; `"response"` (default) for raw residuals or `"pearson"` for standardized residuals.
`...`	Additional arguments (not used).

Value

An n x n matrix (unipartite) or nA x nB matrix (bipartite) of residuals. Entries where the original data was `NA` remain `NA`.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

`fitted.ame`, `predict.ame`

residuals.ame_als *Residuals from a fast AME fit*

Description

Returns residuals from an ame_als or lame_als fit.

Usage

```
## S3 method for class 'ame_als'
residuals(object, type = c("response", "working"), plug_in = NULL, ...)
```

Arguments

object	an ame_als object.
type	"response" (default) or "working".
plug_in	logical; for an empirical-Bayes fit, take the response residuals from the plug-in fitted values (TRUE) or from the predictive means (FALSE); the default is the one of fitted.ame_als: TRUE for a Poisson fit, FALSE for a binary fit.
...	ignored.

Details

type = "response" (default) returns observed Y minus the response-scale fitted values, so residuals() reconciles with fitted(): for a binary empirical-Bayes fit the predictive probabilities, or with plug_in = TRUE the plug-in ones, and for a Poisson fit the plug-in mean counts, or with plug_in = FALSE the predictive means that integrate the predictor. type = "working" returns the Gaussian working-scale residuals the estimator actually minimised (identical to "response" for the normal family).

Value

A matrix (cross-sectional) or list of matrices (longitudinal).

residuals.lame *Extract residuals from LAME model*

Description

Response residuals are Y - fitted(object) in each period. Pearson residuals scale them by the standard deviation the family assigns to a response with that fitted value, as in residuals.ame: the residual variance for the normal family, $\mu(1-\mu)$ for binary and cbin, $\mu + \mu^2(e^{\sigma_e^2} - 1)$ for Poisson (the fitted mean μ carries the exposure of the period), and the variance of the category distribution at the posterior-mean linear predictor for ordinal fits. The frn family has no cell-wise variance and is an error of class lame_pearson_residuals. A cell whose variance is zero is NA.

Usage

```
## S3 method for class 'lame'
residuals(object, type = c("response", "pearson"), ...)
```

Arguments

object	Fitted LAME model
type	Type of residuals ("response" or "pearson")
...	Additional arguments

Value

List of residual matrices (one per time point)

---

rhat_dynamic_beta *Multivariate R-hat for dynamic_beta coefficient paths*

Description

Given one or more multi-chain lame fits whose $BETA is [n_iter, p, T], computes the multivariate R-hat per coefficient k treating the length-T path as one multivariate observation per iteration.

Usage

```
rhat_dynamic_beta(fit_list, coefs = NULL)
```

Arguments

fit_list	A list of fitted lame objects from lame_parallel(..., n_chains = K, combine_method = "list"), or any ame_chain_list produced by re-running lame() with different seeds.
coefs	Optional character vector of coefficient names to subset (matches dimnames(fit$BETA)[[2]]).

Details

For chain c, let $\beta_t^{(c)} \in \mathbb{R}^T$ be the path. With m chains and n iterations per chain, define

$$W_k = \frac{1}{m}\sum_c S_c^{(k)}, \quad B_k = \frac{n}{m-1}\sum_c (\bar{\beta}_c^{(k)} - \bar{\beta}^{(k)})(\bar{\beta}_c^{(k)} - \bar{\beta}^{(k)})'$$

where $S_c^{(k)}$ is the within-chain sample covariance of path k in chain c. Then $V_k = ((n-1)/n)W_k + ((m+1)/(mn))B_k$ and $\hat{R}_k^{mvt} = \sqrt{\lambda_{\max}(W_k^{-1}V_k)}$. For nearly degenerate covariances we add a tiny ridge to W_k.

Value

Data frame with one row per coefficient: `coef`, `rhat_mvt`, `rhat_max_univariate`, `n_chains`, `n_iter_per_chain`, `n_periods`. `rhat_mvt` is the Brooks-Gelman multivariate statistic; `rhat_max_univariate` is the max over periods of the split rank-normalised R-hat of each (k, t) series, the statistic of `fit$rhat` and of the between-chain gate of `lame`.

Examples

```
data(YX_bin_list)
# note: pass seed = to lame() -- an external set.seed() does not vary
# the sampler, so it would produce identical chains
# each fit is one chain (chains = 1); rhat_dynamic_beta() compares them
fit_list <- lapply(c(1L, 2L), function(s) {
 lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X, family = "binary", R = 0,
       dynamic_beta = "dyad", seed = s,
       nscan = 60, burn = 15, odens = 5, verbose = FALSE, chains = 1)
})
rhat_dynamic_beta(fit_list)
```

`rmvnorm` *Simulation from a multivariate normal distribution*

Description

Simulates a matrix where the rows are i.i.d. samples from a multivariate normal distribution. Positive semidefinite covariances are supported, including degenerate distributions with zero-variance directions.

Usage

```
rmvnorm(n, mu, Sigma, Sigma.chol = NULL)
```

Arguments

`n`	nonnegative integer sample size
`mu`	finite numeric multivariate mean vector
`Sigma`	finite symmetric positive semidefinite covariance matrix
`Sigma.chol`	optional finite square factor whose crossproduct is `Sigma`; when supplied, this factor determines the covariance

Value

a matrix with `n` rows

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
set.seed(1)
mu <- c(0, 1)
Sigma <- matrix(c(1, 0.3, 0.3, 1), 2, 2)
draws <- rmvnorm(100, mu, Sigma)
colMeans(draws)
```

---

rUV_dynamic_bip_fc_cpp

Bipartite dynamic UV Gibbs update

Description

Replaces the nested R loops for bipartite dynamic_uv in lame.R. Uses direct 2x2 inverse formula for common R=2 case.

Usage

```
rUV_dynamic_bip_fc_cpp(
  U_state,
  V_state,
  E,
  G,
  rho_uv,
  sigma_uv,
  s2,
  max_abs = -1
)
```

Arguments

U_state	3D array (nA x RA x T)
V_state	3D array (nB x RB x T)
E	3D array of residuals (nA x nB x T)
G	Interaction matrix (RA x RB)
rho_uv	AR(1) persistence parameter
sigma_uv	Innovation standard deviation
s2	Dyadic variance
max_abs	Optional positive coordinate bound; nonpositive or infinite values disable it. Each actor conditional respects the bound.

Value

List with updated U_cube, V_cube

`rUV_dynamic_fc_cpp` *Update dynamic latent positions using AR(1) process*

Description

The per-period observation model is the pair model the rest of the sampler uses: the two cells of a dyad, (e_ij, e_ji), have covariance s2 * matrix(c(1, rho, rho, 1), 2, 2). Conditioning one actor's coordinates on every other actor's leaves an exact Gaussian full conditional in which the dyadic correlation enters three ways: the likelihood precision is scaled by 1 / (1 - rho^2), the linear term carries -rho times the reciprocal residual, and the self-cell (i, i) contributes a rank-one correction. Passing `rho = 0` reproduces the independent-dyad kernel exactly.

Usage

```
rUV_dynamic_fc_cpp(
  U_current,
  V_current,
  ET,
  rho_uv,
  sigma_uv,
  s2,
  shrink,
  symmetric,
  rho = 0,
  max_abs = -1
)
```

Arguments

`U_current`	Current 3D array of U positions (n x R x T)
`V_current`	Current 3D array of V positions (n x R x T)
`ET`	3D array of residuals (n x n x T)
`rho_uv`	AR(1) autoregressive parameter for latent positions
`sigma_uv`	Innovation standard deviation for latent positions
`s2`	Dyadic variance
`shrink`	Whether to apply shrinkage
`symmetric`	Whether network is symmetric
`rho`	Dyadic correlation of the (i, j) and (j, i) residual pair
`max_abs`	Optional positive coordinate bound; nonpositive or infinite values disable it. Each actor conditional respects the bound.

Value

List with updated U and V arrays

`rUV_dynamic_snap_fc_cpp` *Update dynamic latent positions with snap-shift model selection*

Description

Like rUV_dynamic_fc_cpp but, for t > 0, chooses per actor between an AR(1) drift prior and a diffuse N(0, kappa^2 I) snap prior via a Gaussian log-marginal-likelihood model selection, drawing a Bernoulli snap indicator delta and sampling the latent position from the selected posterior.

Usage

```
rUV_dynamic_snap_fc_cpp(
  U_current,
  V_current,
  ET,
  rho_uv,
  sigma_uv,
  s2,
  kappa,
  pi_snap,
  delta_u_current,
  delta_v_current,
  shrink,
  symmetric,
  rho = 0,
  max_abs = -1
)
```

Arguments

`U_current`	Current 3D array of U positions (n x R x T)
`V_current`	Current 3D array of V positions (n x R x T)
`ET`	3D array of residuals (n x n x T)
`rho_uv`	AR(1) autoregressive parameter for the drift prior
`sigma_uv`	Innovation standard deviation for the drift prior
`s2`	Dyadic variance
`kappa`	Diffuse snap-prior standard deviation (kappa^2 » sigma_uv^2)
`pi_snap`	Prior snap probability
`delta_u_current`	Current sender-side snap indicators from the previous sweep

delta_v_current	Current receiver-side snap indicators from the previous sweep
shrink	Whether to apply shrinkage
symmetric	Whether network is symmetric
rho	Dyadic correlation of the (i, j) and (j, i) residual pair
max_abs	Optional positive coordinate bound; nonpositive or infinite values disable it. Each actor conditional respects the bound.

Value

List with updated U, V arrays and delta_u, delta_v snap indicators

---

rUV_dynamic_t_fc_cpp *Update dynamic latent positions with heavy-tailed (Student-t) AR(1) innovations*

Description

Like rUV_dynamic_fc_cpp but each AR(1) innovation is Student-t rather than Gaussian, via a scale-mixture: the innovation for `u[t, i]` has variance `sigma^2 / lambda[t, i]` with `lambda[t, i]` distributed as Gamma(nu/2, nu/2). Provides a continuous heavy-tailed alternative to the discrete snap-shift model.

Usage

```
rUV_dynamic_t_fc_cpp(
  U_current,
  V_current,
  ET,
  rho_uv,
  sigma_uv,
  s2,
  nu,
  lambda_u,
  lambda_v,
  shrink,
  symmetric,
  rho = 0,
  max_abs = -1
)
```

Arguments

U_current	Current 3D array of U positions (n x R x T)
V_current	Current 3D array of V positions (n x R x T)
ET	3D array of residuals (n x n x T)

rho_uv	AR(1) autoregressive parameter
sigma_uv	Innovation scale
s2	Dyadic variance
nu	Student-t degrees of freedom
lambda_u	Current local scales for U (n x T)
lambda_v	Current local scales for V (n x T)
shrink	Whether to apply shrinkage
symmetric	Whether network is symmetric
rho	Dyadic correlation of the (i, j) and (j, i) residual pair
max_abs	Optional positive coordinate bound; nonpositive or infinite values disable it. Each actor conditional respects the bound.

Value

List with updated U, V arrays and lambda_u, lambda_v local scales

---

rUV_sym_fc *Gibbs sampling of U and V*

Description

A Gibbs sampler for updating the multiplicative effect matrices U and V in the symmetric case. In this case `U%*%t(V)` is symmetric, so this is parameterized as `V=U%*%L` where `L` is the diagonal matrix of eigenvalues of `U%*%t(V)`.

Usage

```
rUV_sym_fc(E, U, V, s2 = 1, shrink=TRUE, l_var = 1)
```

Arguments

E	square residual relational matrix
U	current value of U
V	current value of V
s2	dyadic variance
shrink	adaptively shrink the factors with a hierarchical prior
l_var	prior variance of an eigen-weight: 1 for a unit response (the default), the variance of the response for a normal response of free scale

Value

U	a new value of U
V	a new value of V

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
U0<-matrix(rnorm(30,2),30,2) ; V0<-U0%*%diag(c(3,-2))
E<- U0%*%t(V0) + matrix(rnorm(30^2),30,30)
rUV_sym_fc(E, U0, V0)
```

---

rZ_bin_bip_batch_cpp *Batch binary Z sampling across all time periods (bipartite, rho=0)*

Description

Vectorized probit update for bipartite binary networks without dyadic correlation.

Usage

```
rZ_bin_bip_batch_cpp(Z, EZ, Y)
```

Arguments

Z	3D array of latent values (nA x nB x T)
EZ	3D array of expected values (nA x nB x T)
Y	3D array of observed values (nA x nB x T)

Value

Updated Z array

---

rZ_nrm_batch_cpp *Batch normal Z sampling across all time periods*

Description

Replaces the per-time-period R loop with a single C++ call that loops internally, reducing R-to-C++ transition overhead.

Usage

```
rZ_nrm_batch_cpp(Z, EZ, rho, s2, Y)
```

Arguments

Z	3D array of latent values (n x n x T)
EZ	3D array of expected values (n x n x T)
rho	Dyadic correlation parameter
s2	Dyadic variance
Y	3D array of observed values (n x n x T)

Value

List with updated Z and E_nrm (residuals)

---

sampler_describe *Describe the estimator behind a fitted object*

Description

Reports which estimation routine produced a fitted object and what uncertainty information, if any, is available. Methods exist for `ame_als` fits (the fast block coordinate descent estimator) and `boot_ame` bootstrap objects.

Usage

```
sampler_describe(object, verbose = TRUE)

## S3 method for class 'ame_als'
sampler_describe(object, verbose = TRUE)

## S3 method for class 'boot_ame'
sampler_describe(object, verbose = TRUE)

## S3 method for class 'ame'
sampler_describe(object, verbose = TRUE)

## S3 method for class 'lame'
sampler_describe(object, verbose = TRUE)
```

Arguments

object	a fitted object (`ame_als` or `boot_ame`).
verbose	logical; if `TRUE` (default) print the description.

Value

The input `object`, invisibly.

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
fit <- ame_als(Y, R = 1, family = "normal", verbose = FALSE)
sampler_describe(fit)
```

`sample_beta_dynamic_cpp` *Sample the dynamic-block beta path via FFBS*

Description

Forward-filter / backward-sample the AR(1) state-space model for the dynamic-block beta coefficients. Returns the joint draw of beta_dyn at every time period.

Usage

```
sample_beta_dynamic_cpp(
  Xdyn_list,
  Xstat_list,
  Z_list,
  offset_list,
  beta_static,
  rho_by_coef,
  sigma_by_coef,
  Lambda,
  beta0_mean,
  beta0_cov,
  s2,
  dyad_rho,
  bipartite,
  symmetric,
  use_dyad_rho,
  Omega_in = NULL,
  h_in = NULL
)
```

Arguments

`Xdyn_list`	T-length list of (n*n) x p_dyn long-format design matrices for the dynamic block (column-major reshape per period).
`Xstat_list`	T-length list of (n*n) x p_static long-format design matrices for the static block.
`Z_list`	T-length list of (n x n) latent matrices.
`offset_list`	T-length list of (n x n) offset matrices (a_i + b_j + U_i'V_j contributions; everything that's not in X*beta).

beta_static	Length p_static current static beta vector.
rho_by_coef	Length p_dyn vector of AR(1) rho values for each dynamic coefficient. (Per-block but expanded per-column for vectorised indexing.)
sigma_by_coef	Length p_dyn vector of AR(1) innovation standard deviations for each dynamic coefficient.
Lambda	Block-diagonal innovation scale matrix (p_dyn x p_dyn). Combined with sigma^2 to give Q = sigma^2 * Lambda.
beta0_mean	Length p_dyn prior mean for the first state beta_1 (the prior is placed directly on beta_1, no predict step at t = 1).
beta0_cov	p_dyn x p_dyn prior covariance for beta_1. Must be a fixed matrix that does not depend on the current (rho, sigma) draw so the transition-only hyperparameter conditionals remain exact.
s2	Dyadic variance.
dyad_rho	Dyadic correlation (ignored when use_dyad_rho=FALSE).
bipartite	Whether the network is bipartite.
symmetric	Whether the network is symmetric.
use_dyad_rho	Whether to use the dyad-corr branch (TRUE only for unipartite, asymmetric, with a non-zero rho).
Omega_in, h_in	Optional per-period sufficient statistics, as returned by beta_dynamic_stats_cpp for the same arguments (a p_dyn x p_dyn x T array and a T x p_dyn matrix); the filter then reads them instead of accumulating them again. The default (NULL) accumulates.

Value

List with: path – a (T x p_dyn) matrix of beta draws (one row per period); chol_fail – integer count of Cholesky failures.

`sample_beta_static_cpp` *Sample the static-block beta conditional on the dynamic path*

Description

Conjugate Gaussian update for the static coefficient block, treating the dynamic path as known. Uses a flat-ish ridge-style prior `prior_prec` on the static beta (typically diag(1/g) to match the existing g-prior path).

Usage

```
sample_beta_static_cpp(
  Xdyn_list,
  Xstat_list,
  Z_list,
  offset_list,
  beta_dyn_path,
  prior_mean,
  prior_prec,
  s2,
  dyad_rho,
  bipartite,
  symmetric,
  use_dyad_rho
)
```

Arguments

Xdyn_list	T-length list of long-format dynamic design matrices.
Xstat_list	T-length list of long-format static design matrices.
Z_list	T-length list of latent (n x n) matrices.
offset_list	T-length list of (n x n) offsets.
beta_dyn_path	(T x p_dyn) dynamic path matrix.
prior_mean	Length p_static prior mean.
prior_prec	p_static x p_static prior precision.
s2	Dyadic variance.
dyad_rho	Dyadic correlation.
bipartite	TRUE/FALSE.
symmetric	TRUE/FALSE.
use_dyad_rho	TRUE/FALSE.

Value

List with: beta – length p_static; chol_fail – integer.

---

sample_dynamic_ab_cpp *Sample dynamic additive effects with AR(1) evolution*

Description

Gibbs update of the per-period additive effects under the AR(1) state model with stationary initial condition. For each actor and period the full conditional combines the AR(1) bridge prior (stationary init at t = 1) with the dyadic residual likelihood: resid_ij = a_i + b_j + e_ij.

Usage

```
sample_dynamic_ab_cpp(
  a_current,
  b_current,
  Z_array,
  EZ_array,
  rho_ab,
  sigma_ab,
  s2,
  symmetric,
  rho_dyad = 0,
  center = FALSE,
  rvar = TRUE,
  cvar = TRUE
)
```

Arguments

a_current	Current 2D array of row effects (n x T)
b_current	Current 2D array of column effects (n x T)
Z_array	3D array of latent positions (n x n x T)
EZ_array	3D array of expected values without additive effects (n x n x T)
rho_ab	AR(1) parameter for additive effects
sigma_ab	Innovation standard deviation
s2	Dyadic residual variance
symmetric	Whether the network is symmetric
rho_dyad	Within-dyad correlation of the residual pair (e_ij, e_ji); 0 (the default) treats the dyads as independent. Ignored for symmetric networks and for non-square slices. Must lie strictly between -1 and 1 for a directed square panel.
center	Whether each period's effects are drawn conditional on summing to zero across the actors of their margin, the identification used when the model has a static intercept. The draw is then the conditional of the block's joint normal given the constraint, not a projection of an unconstrained draw.
rvar	Whether the sender margin is active.
cvar	Whether the receiver margin is active.

Details

With rho_dyad = 0 the dyads are independent, e_ij ~ N(0, s2), and the conditional for a_i uses only row i of the residual. With rho_dyad != 0 the two directed residuals of a dyad are jointly normal, (e_ij, e_ji) ~ N(0, s2 * ((1, rho), (rho, 1))), whose inverse covariance is ((q, p), (p, q)) with q = 1 / (s2 (1 - rho^2)) and p = -rho / (s2 (1 - rho^2)). The effect being drawn enters e_ij directly and enters the pair's cross term through e_ji, so each fully observed dyad contributes q to the precision and q (r_ij - b_j) + p (r_ji - a_j - b_i) to the numerator of the a-step (symmetrically for the b-step). A dyad with only one direction observed falls back to the marginal isotropic weight 1 / s2.

The reciprocal effect (b_j for the a-step, the freshly updated a_i for the b-step) is subtracted from each residual, missing residuals are skipped, the diagonal is skipped, and for symmetric networks each dyad contributes exactly once (resid(i,j) = a_i + a_j + e_ij) so the pair term does not apply and `rho_dyad` is ignored.

Value

List with updated a and b arrays

`sample_rho_beta_cpp` *Sample the AR(1) rho for each dynamic block*

Description

Truncated-Normal full conditional, one rho per block.

Usage

```
sample_rho_beta_cpp(
  beta_path,
  group_id,
  n_groups,
  Lambda_inv,
  sigma_by_coef,
  rho_current,
  rho_prior_mean,
  rho_prior_sd,
  rho_lower = 0,
  rho_upper = 0.999
)
```

Arguments

`beta_path`	(T x p_dyn) matrix of beta draws.
`group_id`	Length p_dyn integer vector (1-based) of block IDs.
`n_groups`	Number of distinct block IDs.
`Lambda_inv`	(p_dyn x p_dyn) inverse of the (full) Lambda scale.
`sigma_by_coef`	Length p_dyn vector of per-coef sigma (block-shared).
`rho_current`	Length n_groups vector of current rho values.
`rho_prior_mean`	Length n_groups vector of prior means.
`rho_prior_sd`	Length n_groups vector of prior SDs.
`rho_lower`	Lower truncation bound (typically 0).
`rho_upper`	Upper truncation bound (typically 0.999).

Value

Length n_groups vector of new rho values.

sample_sigma_beta_cpp *Sample the AR(1) innovation sigma for each dynamic block*

Description

Inverse-Gamma full conditional, one sigma per block.

Usage

```
sample_sigma_beta_cpp(
  beta_path,
  group_id,
  n_groups,
  Lambda_inv,
  rho_by_group,
  prior_shape,
  prior_scale
)
```

Arguments

beta_path	(T x p_dyn) matrix.
group_id	Length p_dyn integer vector (1-based) of block IDs.
n_groups	Number of distinct block IDs.
Lambda_inv	(p_dyn x p_dyn) inverse of the (full) Lambda scale.
rho_by_group	Length n_groups vector of current rho values.
prior_shape	Length n_groups vector of IG shape parameters.
prior_scale	Length n_groups vector of IG scale parameters.

Value

Length n_groups vector of new sigma values.

sampsonmonks *Sampson's monastery data*

Description

Several dyadic variables measured on 18 members of a monastery.

Format

A socioarray whose dimensions represent nominators, nominees and relations. Each monk was asked to rank up to three other monks on a variety of positive and negative relations. A rank of three indicates the "highest" ranking for a particular relational variable. The relations `like_m1` and `like_m2` are evaluations of liking at one and two timepoints previous to when the other relations (including `like`) were measured. In the copy of this array that circulated with **amen**, the three liking layers had their monks in a different order from the array's row and column names; they are re-aligned here so every layer follows the same actor labels.

Source

Originally available at `http://moreno.ss.uci.edu/data.html#sampson` (site no longer accessible).

---

sheep *Sheep dominance data*

Description

Number of dominance encounters between 28 female bighorn sheep. Cell (i,j) records the number of times sheep i dominated sheep j. From Hass (1991).

Format

A list consisting of the following:

- `dom`: a directed socioarray recording the number of dominance encounters.
- `age`: the age of each sheep in years.

Source

Originally available at `http://moreno.ss.uci.edu/data.html#sheep` (site no longer accessible).

---

simulate.ame *Simulate networks from a fitted AME model*

Description

Generates networks from aligned joint posterior draws. A fit that did not save them can use the conditional approximation by asking for it with `type = "approximate"`; see Details.

Usage

```
## S3 method for class 'ame'
simulate(
  object,
  nsim = 100,
  seed = NULL,
  newdata = NULL,
  burn_in = 0,
  thin = 1,
  return_latent = FALSE,
  type = c("posterior", "approximate", "auto"),
  ...
)
```

Arguments

object	fitted model object of class "ame"
nsim	number of networks to simulate (default: 100)
seed	Optional random seed. An explicit seed is local to this call: the caller's RNG state is restored on return or error. With NULL, the caller's current RNG stream advances normally.
newdata	optional list containing new covariate data: **Xdyad** dyadic covariates (n x n x p array or nA x nB x p for bipartite) **Xrow** row/sender covariates (n x p matrix or nA x p for bipartite) **Xcol** column/receiver covariates (n x p matrix or nB x p for bipartite) If NULL, uses covariates from original model fit
burn_in	number of initial MCMC samples to discard (default: 0, assumes burn-in already removed)
thin	thinning interval for MCMC samples (default: 1, use every sample)
return_latent	logical: return latent Z matrices in addition to Y? (default: FALSE)
type	simulation route: `"posterior"` (default) replicates the network from aligned saved posterior draws; `"approximate"` is the conditional approximation. `"auto"` is accepted as an alias of `"posterior"`. See Details.
...	additional arguments (not currently used)

Details

With `type = "posterior"`, every replicate uses the same saved MCMC iteration for coefficients, actor effects, positions, coupling and observation variance. Saved blocks may be thinned differently when their iteration indices are available; only shared iterations are sampled. Missing joint storage is an error (class `lame_simulate_draws`) that names `type = "approximate"`. Store effects and positions with `posterior_options(save_ab = TRUE, save_UV = TRUE, thin_ab = 1, thin_UV = 1)` when fitting.

`type = "approximate"` is the conditional approximation: it substitutes posterior summaries for the latent trajectories and does not propagate their joint posterior uncertainty. It runs only when

requested. The former default `type = "auto"` chose between the two routes; it is now an alias of `"posterior"`. The returned `method` identifies the route.

Ordinal fits with explicit cutpoints use the corresponding saved cutpoint draw. Data-induced ordinal fits match the observed category frequencies, so these replications assess dependence conditional on those frequencies.

Value

A list with components:

Y list of nsim simulated networks in the same format as the original data

method joint posterior or conditional approximation

draw_index sampled coefficient-row indices for joint replication

Z if return_latent=TRUE, list of nsim latent Z matrices

family the family of the model (binary, normal, etc.)

mode network mode (unipartite or bipartite)

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Fit a model
data(YX_bin)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_bin$Y, Xdyad = YX_bin$X, burn = 50, nscan = 250, odens = 5,
           family = "binary", verbose = FALSE,
          posterior_opts = posterior_options(save_ab = TRUE, save_UV = TRUE), chains = 1)

# Simulate 10 networks from posterior
sims <- simulate(fit, nsim = 10)
```

---

simulate.ame_als *Simulate networks from a fitted ame_als model*

Description

Draws `nsim` replicates of `Y` from the fitted ALS mean structure: `mu`, `beta`, `a`, `b`, `U`, `V`, and the family-appropriate noise distribution. For directed unipartite normal, binary and Poisson fits, reciprocal dyads use the fitted `rho`. Binary IRLS fits use their stated marginal link and a Gaussian copula with the fitted dyadic correlation. Square bipartite networks retain their diagonal cells. The returned object has class `"ame.sim"` so it is compatible with `plot_ppc_*`-style consumers (where applicable).

Usage

```
## S3 method for class 'ame_als'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	an ame_als fit.
nsim	integer; number of replicates to simulate (default 1).
seed	optional RNG seed.
...	ignored.

Details

ALS is a point estimator. simulate.ame_als holds mu, beta, a, b, U, V at the point estimate and resamples the noise. For uncertainty over the parameters themselves, use ame_als_bootstrap (whose replicates each carry their own resampled Y) and combine those simulations.

Value

An object of class "ame.sim" with element Y – a list of nsim simulated outcome arrays (one matrix for a cross-section, one list-of-matrices per slice for a longitudinal fit).

---

simulate.lame *Simulate longitudinal networks from a fitted LAME model*

Description

Generates network replicates at fitted periods from aligned posterior trajectories. A fit that did not save them, and the legacy trajectory scenarios (start_from, extended n_time), use the conditional approximation by asking for it with type = "approximate"; see Details.

Usage

```
## S3 method for class 'lame'
simulate(
  object,
  nsim = 100,
  seed = NULL,
  newdata = NULL,
  n_time = NULL,
  burn_in = 0,
  thin = 1,
  return_latent = FALSE,
  start_from = "posterior",
  type = c("posterior", "approximate", "auto"),
  ...
)
```

Arguments

object	fitted model object of class "lame"
nsim	number of network trajectories to simulate (default: 100)
seed	Optional random seed. An explicit seed is local to this call: the caller's RNG state is restored on return or error. With NULL, the caller's current RNG stream advances normally.
newdata	optional list containing new covariate data: **Xdyad** list of T dyadic covariate arrays (n x n x p or nA x nB x p) **Xrow** list of T row/sender covariate matrices (n x p or nA x p) **Xcol** list of T column/receiver covariate matrices (n x p or nB x p) If NULL, uses covariates from original model fit
n_time	number of time periods to simulate. If NULL, uses same as original data
burn_in	number of initial MCMC samples to discard (default: 0)
thin	thinning interval for MCMC samples (default: 1, use every sample)
return_latent	logical: return latent Z matrices in addition to Y? (default: FALSE)
start_from	character: how to initialize the simulation **"posterior"** use saved trajectories for joint replication; posterior summaries for the approximation (default) **"random"** random initialization **"data"** use first time point from original data
type	simulation route: "posterior" (default) replicates the network from aligned saved posterior trajectories; "approximate" is the conditional approximation. "auto" is accepted as an alias of "posterior". See Details.
...	additional arguments (not currently used)

Details

With type = "posterior", every replicate uses the same saved MCMC iteration for coefficients, actor effects, positions, coupling and observation variance and period-specific reciprocity. Saved blocks may be thinned differently when their iteration indices are available; only shared iterations are sampled. Missing joint storage is an error (class lame_simulate_draws) that names type = "approximate". Store effects and positions with posterior_options(save_ab = TRUE, save_UV = TRUE, thin_ab = 1, thin_UV = 1) when fitting.

type = "approximate" is the conditional approximation: it substitutes posterior summaries for the latent trajectories and does not propagate their joint posterior uncertainty. It runs only when requested. The former default type = "auto" chose between the two routes; it is now an alias of "posterior". The returned method identifies the route.

Joint longitudinal replication uses saved trajectories at fitted periods; it does not evolve the final state through the historical covariates. Use predict(fit, h = ...) for future periods. The legacy start_from and extended n_time options apply to the conditional approximation only; with type = "posterior" they are an error (class lame_simulate_periods). The approximation extends n_time past the fitted periods only for static coefficients; a dynamic_beta fit has coefficients for its fitted periods alone, so a longer n_time is the same error. In the approximation, newdata$Xdyad

must be a list with one array per period; any other shape is an error (class `lame_simulate_xlist`). Ordinal fits with explicit cutpoints use the corresponding saved cutpoint draw. Data-induced ordinal fits match the observed category frequencies, so these replications assess dependence conditional on those frequencies.

Value

A list with components:

Y list of nsim simulated longitudinal network trajectories, each element is a list of T networks

method joint posterior or conditional approximation

draw_index sampled coefficient-row indices for joint replication

Z if return_latent=TRUE, list of nsim latent Z trajectories

family the family of the model (binary, normal, etc.)

mode network mode (unipartite or bipartite)

n_time number of time periods

Author(s)

Shahryar Minhas

Examples

```
# Create simple longitudinal network data
set.seed(1)
n <- 10
nms <- paste0("n", 1:n)
Y_list <- list(
  matrix(rnorm(n * n), n, n, dimnames = list(nms, nms)),
  matrix(rnorm(n * n), n, n, dimnames = list(nms, nms))
)
diag(Y_list[[1]]) <- diag(Y_list[[2]]) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(Y_list, family = "normal",
            nscan = 50, burn = 10, odens = 1, verbose = FALSE, plot = FALSE,
          posterior_opts = posterior_options(save_ab = TRUE, save_UV = TRUE), chains = 1)

# Simulate 10 network trajectories from posterior
sims <- simulate(fit, nsim = 10)
```

`simulate_posterior` *Simulate posterior distributions from fitted AME model*

Description

Simulate posterior distributions from fitted AME model

Usage

```
simulate_posterior(
  fit,
  component = c("UV", "ab", "beta", "Y"),
  n_samples = NULL,
  seed = NULL
)
```

Arguments

`fit`	Fitted ame model object
`component`	Character; which component to simulate: "UV", "ab", "beta", "Y"
`n_samples`	Number of posterior samples to return. Defaults to `NULL`, which uses every saved draw when the fit stores them (see `posterior_options`) and 100 otherwise. A smaller value draws a random subset, which widens Monte Carlo error in the tails of any interval computed from the result.
`seed`	Random seed for reproducibility

Details

This function can simulate posterior distributions even when they weren't saved during MCMC, by using the posterior means and variance components.

For more accurate posteriors, use posterior_options() during model fitting to save the actual MCMC samples. Longitudinal additive trajectories require saved draws; their joint posterior is not reconstructed from summaries.

Value

Array or matrix of posterior samples

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Fit a model with multiplicative effects
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE, chains = 1)

# Get posterior samples of regression coefficients
beta_post <- simulate_posterior(fit, "beta", n_samples = 50)
```

`snap_category_summary` *Summarize snap indices by actor category*

Description

Summarize snap indices by actor category

Usage

```
snap_category_summary(fit, groups, years = NULL, side = c("u", "v"))
```

Arguments

`fit`	a `lame` fit with snap-shift MCMC output, run with retained snap draws for posterior uncertainty.
`groups`	a named list mapping group names to actor names/indices, or a named vector whose values are group labels and whose names are actors.
`years`	optional year/period names or indices. Default uses every period.
`side`	`"u"` for sender/row-side snap indicators or `"v"` for receiver/column-side indicators.

Value

A data frame with one row per group-year.

Examples

```
set.seed(1)
n <- 12; Tt <- 3
Y <- lapply(seq_len(Tt), function(t) {
  m <- matrix(rnorm(n * n), n, n); diag(m) <- NA; m
})
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(Y, R = 1, dynamic_uv = TRUE, dynamic_uv_kind = "snap",
            keep_snap_draws = "draws",
            nscan = 20, burn = 5, odens = 1, verbose = FALSE, plot = FALSE, chains = 1)
snap_category_summary(fit, groups = list(g1 = 1:6, g2 = 7:12), side = "u")
```

snap_index_draws *Extract posterior draws of snap indices*

Description

Computes draw-level averages of MCMC snap indicators over selected actors and years. The result is useful for system-level rupture indices and focal-actor rupture indices. Fits should be run with `keep_snap_draws = "draws"` for posterior uncertainty. If only `snap_prob` is present, the function returns a single score row and warns that intervals cannot be read as posterior intervals.

Usage

```
snap_index_draws(fit, actors = NULL, years = NULL, side = c("u", "v"))
```

Arguments

`fit`	a `lame` fit with snap-shift MCMC output, run with retained snap draws for posterior uncertainty.
`actors`	optional actor names or indices. Default uses every actor on the selected side.
`years`	optional year/period names or indices. Default uses every period.
`side`	`"u"` for sender/row-side snap indicators or `"v"` for receiver/column-side indicators.

Value

A data frame with columns `draw`, `year`, `side`, `n_actors`, and `snap_index`.

Examples

```
set.seed(1)
n <- 12; Tt <- 3
Y <- lapply(seq_len(Tt), function(t) {
  m <- matrix(rnorm(n * n), n, n); diag(m) <- NA; m
})
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(Y, R = 1, dynamic_uv = TRUE, dynamic_uv_kind = "snap",
            keep_snap_draws = "draws",
            nscan = 20, burn = 5, odens = 1, verbose = FALSE, plot = FALSE, chains = 1)
draws <- snap_index_draws(fit, side = "u")
head(draws)
```

snap_index_summary *Summarize posterior snap indices*

Description

Summarize posterior snap indices

Usage

```
snap_index_summary(
  fit,
  actors = NULL,
  years = NULL,
  side = c("u", "v"),
  probs = c(0.025, 0.1, 0.5, 0.9, 0.975)
)
```

Arguments

fit	a lame fit with snap-shift MCMC output, run with retained snap draws for posterior uncertainty.
actors	optional actor names or indices. Default uses every actor on the selected side.
years	optional year/period names or indices. Default uses every period.
side	"u" for sender/row-side snap indicators or "v" for receiver/column-side indicators.
probs	numeric quantiles to report.

Value

A data frame with one row per selected year.

Examples

```
set.seed(1)
n <- 12; Tt <- 3
Y <- lapply(seq_len(Tt), function(t) {
  m <- matrix(rnorm(n * n), n, n); diag(m) <- NA; m
})
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(Y, R = 1, dynamic_uv = TRUE, dynamic_uv_kind = "snap",
            keep_snap_draws = "draws",
            nscan = 20, burn = 5, odens = 1, verbose = FALSE, plot = FALSE, chains = 1)
snap_index_summary(fit, side = "u")
```

snap_rank_summary *Summarize posterior rank uncertainty for snap years*

Description

Ranks year-level snap indices within each retained draw, with rank 1 assigned to the highest snap index.

Usage

```
snap_rank_summary(fit, actors = NULL, years = NULL, side = c("u", "v"))
```

Arguments

fit	a lame fit with snap-shift MCMC output, run with retained snap draws for posterior uncertainty.
actors	optional actor names or indices. Default uses every actor on the selected side.
years	optional year/period names or indices. Default uses every period.
side	"u" for sender/row-side snap indicators or "v" for receiver/column-side indicators.

Value

A data frame with year-level index means and rank probabilities.

Examples

```
set.seed(1)
n <- 12; Tt <- 3
Y <- lapply(seq_len(Tt), function(t) {
  m <- matrix(rnorm(n * n), n, n); diag(m) <- NA; m
})
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(Y, R = 1, dynamic_uv = TRUE, dynamic_uv_kind = "snap",
            keep_snap_draws = "draws",
            nscan = 20, burn = 5, odens = 1, verbose = FALSE, plot = FALSE, chains = 1)
snap_rank_summary(fit, side = "u")
```

`summary.ame` *Summary of an AME object*

Description

Summarizes a fitted AME (Additive and Multiplicative Effects) model, including parameter estimates, standard errors, credible intervals, and model diagnostics.

Usage

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

Arguments

`object`	an object of class "ame", typically the result of fitting an AME model using the `ame` function
`...`	additional parameters (currently not used)

Details

The summary includes:

Regression coefficients Posterior means, posterior standard deviations, z-values, normal-tail p-values, and 95% credible intervals for dyadic, sender, and receiver covariates. The z-value is the posterior mean divided by the posterior SD, and the p-value is the two-sided normal tail probability of that z-value: a posterior summary, not a frequentist test. The credible intervals are posterior quantiles of the draws, and the full posterior is in the BETA matrix.

Variance components Estimates and standard errors for:

va Variance of additive sender/row effects (asymmetric networks)

cab Covariance between sender and receiver effects

vb Variance of additive receiver/column effects (asymmetric networks)

rho Dyadic correlation (reciprocity in directed networks)

ve Residual variance

For symmetric networks, only va and ve are estimated.

Rhat and ESS A fit of two or more chains adds the columns `Rhat` and `ESS` to both tables. `Rhat` is the split rank-normalised R-hat of Vehtari et al. (2021) that the between-chain gate compares with 1.05, on the draws the fit holds (the last block of every chain); for a monitored quantity it is the value in `fit$rhat`. `ESS` is the effective sample size of the pooled draws. A one-chain fit has neither column: R-hat compares chains.

Value

A list of class "summary.ame" containing:

call	The original function call
beta	Matrix of regression coefficient estimates and statistics
variance	Matrix of variance component estimates; every row is reported (a single period carries no event-rate rule), with a note on the intercept where a saturated cap withholds it
ab_var_unidentified	TRUE when variance rows are NA
ab_coef_reason	"saturated" when a cbin or frn fit whose rows are all at the cap, or half of them with cells hidden, withholds the intercept and row-covariate coefficients (NA otherwise); a row at its cap leaves the level of the rows unbounded above, and in simulated one-period designs the intercept interval covered 0.64 of the panels with every row at the cap and 0.84 of those with cells hidden and a share from 0.5, against 0.98 or more elsewhere
prior_supplied	TRUE when the fit's prior carried Sab0, eta0 or etaab; the print method names the prior
odmax_saturated_share	share of observed rows at their nomination cap, which sets the sampler's prior pooling and the intercept rule (NULL off cbin / frn)

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

ame, print.summary.ame

---

summary.ame.sim *Summary method for AME simulations*

Description

Summary method for AME simulations

Usage

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

Arguments

object	simulation object of class "ame.sim"
...	additional arguments (not used)

Value

summary statistics invisibly

---

summary.ame_als *Summarize an ame_als object*

Description

Summarize an ame_als object

Usage

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

Arguments

object	an ame_als object.
...	ignored.

Value

A list of class "summary.ame_als", printed as a table.

---

summary.boot_ame *Summarize bootstrap results for a fast AME fit*

Description

Summarize bootstrap results for a fast AME fit

Usage

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

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

Arguments

object	a boot_ame object.
...	ignored.
x	a summary.boot_ame object.

Value

An object of class "summary.boot_ame", printed as a set of tables.

summary.lame *Summary of a LAME object*

Description

Summarizes a fitted LAME (Longitudinal Additive and Multiplicative Effects) model, including parameter estimates, standard errors, credible intervals, and model diagnostics.

Usage

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

Arguments

object	an object of class "lame", typically the result of fitting a longitudinal AME model using the lame function
...	additional parameters (currently not used)

Details

The summary includes:

Regression coefficients Posterior means, posterior standard deviations, z-values, normal-tail p-values, and 95% credible intervals for dyadic, sender, and receiver covariates. The z-value is the posterior mean divided by the posterior SD, and the p-value is the two-sided normal tail probability of that z-value: a posterior summary, not a frequentist test. The credible intervals are posterior quantiles of the draws, and the full posterior is in the BETA matrix.

Variance components Estimates and standard errors for:

- **va** Variance of additive sender/row effects
- **cab** Covariance between sender and receiver effects
- **vb** Variance of additive receiver/column effects
- **rho** Dyadic correlation (reciprocity)
- **ve** Residual variance

Dynamic coefficients per period Only printed when the fit was produced with dynamic_beta on at least one coefficient. The table has one row per coefficient with columns:

- **Mean** average of the per-period posterior means across t
- **Min, Max** smallest and largest per-period posterior mean
- **Drift** Max - Min: the absolute range of the per-period posterior means, in coefficient units
- **Drift_pct** 100 * Drift / |Mean|, the drift as a percentage of the average level. Reported as NA when the time-average is near zero (within 5\ range), because a trajectory that crosses zero makes the percentage explode; read Drift in that case
- **Dynamic** "Y" if the coefficient was flagged as dynamic, "N" if it was held static

The block also prints the per-block AR(1) hyperparameters (rho_beta = ...). For per-period credible intervals use confint.lame.

For dynamic fits the summary reports the posterior means of the persistence parameters rho_uv (latent positions) and rho_ab (additive effects). These are learned from period-to-period transitions, so on short panels, and on sparse bipartite panels where each period's positions are themselves uncertain, the data inform them weakly. The summary measures how weakly. For each reported persistence it divides the posterior standard deviation by the standard deviation of the prior the fit used (the normal prior of prior$rho_uv_mean and prior$rho_uv_sd, or of prior$rho_ab_mean and prior$rho_ab_sd, truncated to the support of the sampler, $|\rho| < 0.99$; see prior_summary()) and returns the ratio in persistence_prior_check. A ratio near one says that the posterior is as wide as the prior, so the data add little to that persistence. When the ratio is above 0.6 the printed summary states it, for example rho_uv: posterior sd 0.073 is 0.92 of the prior sd 0.079: the data barely inform it. The cutoff, a posterior sd above 0.6 of the prior sd, is the one rho_beta_prior_check applies to the coefficient persistences.

Value

A list of class "summary.lame" containing:

call — The original function call

beta — Matrix of regression coefficient estimates and statistics

variance — Matrix of variance component estimates. On a directed binary panel of two periods or more with fewer than 1.2 events per actor-period the va and vb rows are NA: in simulated panels their 95 percent intervals covered the generating variances in 0.28 to 0.91 of the panels below that rate. The draws stay in fit$VC. Under a supplied effects prior the rows are reported under it.

ab_var_unidentified — TRUE when those rows are NA

ab_var_unidentified_reason — "sparse" when the event rate put them there (NA when they are reported)

ab_data_reason — the same reason read off the data alone: it differs from ab_var_unidentified_reason when the effects prior was supplied, in which case the rows are reported under that prior

ab_coef_reason — "sparse", "saturated" or "cap_shrunk" when the intercept and row-covariate coefficients are NA (an event rate below 0.6, or 1.6 with a multiplicative term; a one-period cbin or frn design whose rows are at the cap; a cbin or frn design of two periods or more with a cap share of 0.5 or more and a posterior-mean va from 0.5 to 1.5), NA otherwise

prior_supplied — TRUE when the fit's prior carried Sab0, eta0 or etaab; the print method then names the prior the variances are reported under

events_per_actor_period — events per actor per period (NULL on families without an event rate), the rate the sparse-panel rules read

events_per_actor — events per actor over the panel (NULL on families without an event rate and on fits saved before it was recorded)

odmax_saturated_share — share of observed rows at their nomination cap, which sets the sampler's prior pooling and the one-period intercept rule (NULL off cbin / frn)

`persistence_prior_check`
: for dynamic fits, a data frame with one row per reported persistence (`rho_uv`, `rho_ab`): the posterior standard deviation `post_sd`, the standard deviation of the truncated prior it started from `prior_sd`, their ratio `sd_ratio`, and `weak`, `TRUE` when the ratio is above 0.6 (`NULL` when no persistence is reported)

`n.periods`
: Number of time periods in the longitudinal data

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

`lame`, `print.summary.lame`

`summary.lame.sim` *Summary method for LAME simulations*

Description

Summary method for LAME simulations

Usage

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

Arguments

`object`
: simulation object of class "lame.sim"

`...`
: additional arguments (not used)

Value

summary statistics

summary.lame_robust *Summarize a robust longitudinal network model*

Description

Summaries use retained posterior draws. Coefficient summaries include the static covariate coefficients and every time intercept; summary() also reports estimated variance components and AR coefficients. Unused factor hyperparameters are excluded when the rank is zero.

Usage

```
## S3 method for class 'lame_robust'
coef(object, ...)

## S3 method for class 'lame_robust'
vcov(object, ...)

## S3 method for class 'lame_robust'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'lame_robust'
fitted(object, ...)

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

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

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

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

Arguments

object, x	A lame_robust fit, or its summary for the summary print method.
...	Additional arguments are accepted only by fitted(), which passes them to predict.lame_robust().
parm	Coefficient names or positive column indices; omit for all.
level	Central credible-interval probability, strictly between zero and one.

Value

`coef()` returns posterior coefficient means. `vcov()` returns their posterior covariance. `confint()` returns central posterior quantiles. `fitted()` returns fitted mean networks; `nobs()` counts observed off-diagonal responses. `summary()` returns a `summary.lame_robust` object containing a posterior summary table and resolved priors. Print methods return their input invisibly.

See Also

`lame_robust()`, `compute_mcmc_diagnostics()`

---

tidy *S3 generic for* `tidy`

Description

Light-weight fallback generic so that calls of the form `tidy(fit)` dispatch through R's S3 system even when the **broom** or **generics** packages are not loaded. When either is loaded, its generic resolves first; this generic only fires for bare-namespace use.

Usage

```
tidy(x, ...)
```

Arguments

`x`	An object to tidy.
`...`	Passed to the relevant method.

Value

A data frame; method-specific schema.

Examples

```
set.seed(1)
n <- 12
Y <- matrix(rnorm(n * n), n, n); diag(Y) <- NA
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(Y, R = 1, nscan = 20, burn = 5, odens = 1, verbose = FALSE, chains = 1)
tidy(fit)
```

`tidy.ame` *Tidy method for fitted* `ame` / `lame` *objects*

Description

Returns a data frame with one row per estimated coefficient, compatible with the **broom** idiom. For `dynamic_beta` fits (3-D `BETA`), returns one row per coefficient *per period* with a `period` column. Standard errors are posterior standard deviations; `statistic` is `estimate / std.error`.

Usage

```
## S3 method for class 'ame'
tidy(x, conf.int = TRUE, conf.level = 0.95, ...)

## S3 method for class 'lame'
tidy(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	A fitted `ame` / `lame` object.
`conf.int`	Logical; include 95\ (`conf.low`, `conf.high`). Default `TRUE`.
`conf.level`	Confidence level for the interval. Default `0.95`.
`...`	Ignored.

Details

Note on `p.value`**.** This column is included for **broom** compatibility; it is a posterior summary, not a classical test: the two-sided Normal tail probability of the posterior mean over the marginal posterior standard deviation, matching `summary(fit)`. The `conf.low` / `conf.high` columns are the credible interval. The posterior probability of the sign is available directly from `x$BETA`, for example `mean(sign(BETA) == sign(mean(BETA)))`.

Loaded as an S3 method against `generics::tidy` when the **generics** package is available; works as `tidy(fit)` either way once **broom** is loaded.

Value

Data frame with columns `term`, `estimate`, `std.error`, `statistic`, `p.value`, `conf.low`, `conf.high`, and (for dynamic_beta fits) `period`.

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X, family = "binary", R = 0,
            nscan = 100, burn = 20, odens = 5, verbose = FALSE, chains = 1)
tidy(fit)
```

`tidy.ame_als` *Tidy method for fitted* `ame_als` / `lame_als` *objects*

Description

Returns a data frame with one row per regression coefficient, compatible with the **broom** idiom, so that ALS fits compose with **modelsummary** / **kableExtra** pipelines next to MCMC fits. Standard errors come from the bootstrap object attached to `x$bootstrap` when present (preferred, fully propagated), and otherwise from `vcov.ame_als`: the Laplace covariance of an empirical-Bayes fit (the default for normal, binary and Poisson fits) or the conditional sandwich covariance of a fixed-effects fit. `statistic` is `estimate / std.error`; `p.value` is the Normal-approximation two-sided tail $2(1-\Phi(|z|))$ from the bootstrap or sandwich standard error. It is a Wald-style summary for the point estimator, not a posterior probability.

Usage

```
## S3 method for class 'ame_als'
tidy(x, conf.int = TRUE, conf.level = 0.95, ...)

## S3 method for class 'lame_als'
tidy(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

`x`	A fitted `ame_als` / `lame_als` object.
`conf.int`	Logical; include `conf.low` / `conf.high` columns. Default `TRUE`.
`conf.level`	Confidence level. Default `0.95`.
`...`	Passed to `vcov.ame_als` (e.g. `cluster = "dyad"`).

Details

One row is returned for every regression coefficient of `coef(fit)`: the intercept, the dyadic covariates and the node covariates. The node-covariate coefficients of a fixed-effects fit have no sandwich covariance, so they keep their estimate with `NA` uncertainty. Additive (`a`, `b`) and multiplicative (`U`, `V`) parameters are not included; use `ame_als_bootstrap` and inspect the bootstrap object directly if you need them.

Value

Data frame with columns `term`, `estimate`, `std.error`, `statistic`, `p.value`, `conf.low`, `conf.high`, plus a `se_source` column recording `"bootstrap"`, `"laplace"` (an empirical-Bayes normal, binary or Poisson fit, whose covariance covers every coefficient, node-covariate coefficients included) or `"sandwich"`.

Examples

```
data(YX_nrm)
fit <- ame_als(Y = YX_nrm$Y, Xdyad = YX_nrm$X[, , 2:3],
               family = "normal", R = 1, verbose = FALSE)
tidy(fit)
```

`tidy.boot_ame` *Tidy method for a standalone bootstrap object (*`boot_ame`*)*

Description

`ame_als_bootstrap()` returns an object of class `"boot_ame"` (not `"ame_als"`); this tidy method exposes the bootstrap estimates as a broom-style data frame so the standalone object composes with **modelsummary** the same way an embedded `ame_als(..., bootstrap = N)` fit does.

Usage

```
## S3 method for class 'boot_ame'
tidy(x, conf.level = 0.95, ...)
```

Arguments

`x`	A `boot_ame` object.
`conf.level`	Confidence level. Default `0.95`.
`...`	Ignored.

Value

Data frame with columns `term`, `estimate`, `std.error`, `statistic`, `p.value`, `conf.low`, `conf.high`, `se_source`.

`trace_plot` *MCMC trace plots and density plots for AME/LAME model parameters*

Description

Creates diagnostic plots for Markov Chain Monte Carlo (MCMC) samples from AME or LAME models. Displays trace plots to assess convergence and mixing, alongside density plots to visualize posterior distributions.

Usage

```
trace_plot(
  fit,
  params = c("all", "beta", "variance", "dynamic"),
  include = NULL,
  exclude = NULL,
  ncol = 3,
  nrow = NULL,
  burn.in = 0,
  thin = 1,
  title = NULL
)
```

Arguments

fit	An object of class "ame" or "lame" containing MCMC samples
params	Character vector specifying which parameters to plot: "beta" for regression coefficients, "variance" for variance components, "dynamic" for the dynamic hyperparameters, or "all" (default) for all three
include	Character vector of specific parameter names to include. Raw names ("va", "ve", "rho", "rho_uv", "sigma_ab", ...) and their display labels ("Error variance", "Position persistence (rho_uv)", ...) are both accepted; display labels match regardless of case. For dynamic_beta fits, per-period traces are named like "x_dyad[t1]"; supplying the base name (e.g. "x_dyad") selects all of its periods
exclude	Character vector of specific parameter names to exclude. Base names match all per-period traces, as with include
ncol	Number of columns for plot layout (default 3)
nrow	Number of rows for plot layout (default NULL, determined automatically)
burn.in	Number of initial iterations of each chain to exclude as burn-in (default 0, assumes burn-in already removed)
thin	Thinning interval for display, within each chain (default 1, no thinning)
title	Optional title for the plot

Details

This function produces two types of diagnostic plots:

Trace plots Show the evolution of parameter values across MCMC iterations. Good mixing is indicated by rapid exploration of the parameter space with no trends or stuck periods.

Density plots Show the posterior distribution of parameters. Multiple modes may indicate identification issues or convergence problems.

The plots help diagnose:

- Convergence: Has the chain reached the stationary distribution?

- Mixing: Is the chain exploring the parameter space efficiently?
- Autocorrelation: Are successive samples highly correlated?

Parameters displayed include:

- Regression coefficients (beta)
- Variance components (va, vb, cab, rho, ve)
- Dynamic hyperparameters, when the fit has them: the persistence and innovation of the actor effects (`rho_ab`, `sigma_ab`) and of the latent positions (`rho_uv`, `sigma_uv`), the reset rate of snap positions (`pi_snap`), the per-group persistence and innovation of dynamic coefficients (`rho_beta`, `sigma_beta`) and the per-period dyadic correlation of `dynamic_rho` fits

A quantity the model fixes draws no panel: the error variance of the binary, ordinal, cbin and frn families (fixed at 1), the dyadic correlation and sender-receiver covariance of bipartite fits, the dyadic correlation of a fit with `dcor = FALSE` and the receiver terms of a symmetric fit. Such a quantity is left out only when its draws are constant, so a stuck chain on an estimated quantity always shows. A quantity that `summary()` reports as not estimated (for example the effect variances of a sparse panel) is left out as well, with a message naming it.

For a fit with several chains, `burn.in` and `thin` apply within each chain.

Value

A ggplot2 object that can be further customized

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Fit an AME model
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X,
           nscan = 50, burn = 10, odens = 1, verbose = FALSE, chains = 1)

# Basic trace plots for all parameters
trace_plot(fit)

# Only regression coefficients
trace_plot(fit, params = "beta")

# Only variance components
trace_plot(fit, params = "variance")

# The persistence and innovation of dynamic actor effects
Y <- lapply(1:3, function(t) YX_nrm$Y[1:20, 1:20])
fit_dyn <- lame(Y, R = 0, dynamic_ab = TRUE, nscan = 100, burn = 20, odens = 2,
                verbose = FALSE, chains = 1)
trace_plot(fit_dyn, params = "dynamic")
```

`update.ame` *Update an AME / LAME fit*

Description

S3 method for `update` that re-fits the model with modified arguments. Reuses the original call recorded in `fit$call`. If the fit was produced with `freeze_call = TRUE`, the data snapshot on `fit$data_snapshot` is used in place of looking up names in the caller's environment.

Usage

```
## S3 method for class 'ame'
update(object, ..., evaluate = TRUE)

## S3 method for class 'lame'
update(object, ..., evaluate = TRUE)
```

Arguments

`object`	A fitted `ame` or `lame` object.
`...`	Named arguments to overwrite in the original call (e.g. `nscan = 5000`, `dynamic_beta = TRUE`, `R = 2`).
`evaluate`	Logical: if `TRUE` (default), evaluate the updated call and return the new fit; if `FALSE`, return the unevaluated call.

Value

A new fitted object, or (when `evaluate = FALSE`) the modified call.

Examples

```
data(YX_bin_list)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- lame(lapply(YX_bin_list$Y, function(y) 1 * (y > 0)), YX_bin_list$X, family = "binary",
            burn = 5, nscan = 20, odens = 1, verbose = FALSE, chains = 1)
# toggle dynamic_beta on without rewriting the whole call
fit_dyn <- update(fit, dynamic_beta = "dyad")
dim(fit_dyn$BETA)  # 3-D now
```

`update.ame_als` *Update an* `ame_als` / `lame_als` *fit*

Description

S3 method for `update` that re-fits an ALS fit with modified arguments. Two routes:

- if `...` contains only the warm-start refit knobs (`Y_new`, `X_new`, `Z_new`, `max_iter`, `tol`, `verbose`, `bias_correct`), it delegates to `ame_als_refit` for a fast warm-started refit (a dynamic fit is refit through its own fitter and comes back dynamic);
- otherwise (e.g. changing `R`, `family`, `lambda`, `bootstrap`) it re-evaluates the original `object$call` with the overrides, matching the `update.ame` semantics on the MCMC side.

The split exists because `ame_als_refit()` is a genuinely warm-started single-fit refit (faster, narrower argument set), while changing `R` / `family` requires a cold-start refit through `ame_als`.

Usage

```
## S3 method for class 'ame_als'
update(object, ..., evaluate = TRUE)

## S3 method for class 'lame_als'
update(object, ..., evaluate = TRUE)
```

Arguments

`object`	A fitted `ame_als` / `lame_als` object.
`...`	Named arguments. See `ame_als_refit` for the warm-start argument list and `ame_als` for the cold-start argument list.
`evaluate`	Logical: if `TRUE` (default), evaluate the updated call and return the new fit; if `FALSE`, return the unevaluated call (cold-start path only).

Value

A new fitted `ame_als` object, or (when `evaluate = FALSE` on the cold-start path) the modified call.

Examples

```
Y <- matrix(rnorm(400), 20, 20); diag(Y) <- NA
fit <- ame_als(Y, R = 1, family = "normal", verbose = FALSE)
# warm-start refit with the same arguments (fast)
fit_w <- update(fit, max_iter = 50)
# cold-start refit with a different R (full restart)
fit_R2 <- update(fit, R = 2)
```

uv_plot *Visualize multiplicative effects (latent factors) from AME models*

Description

Creates a two-dimensional visualization of the multiplicative effects (U and V) from an AME or LAME model. These latent factors capture network structure beyond what is explained by covariates and additive effects, including clustering, transitivity, and other higher-order dependencies.

Usage

```
uv_plot(
  fit = NULL,
  Y = NULL,
  U = NULL,
  V = NULL,
  row.names = NULL,
  col.names = NULL,
  layout = c("circle", "biplot"),
  vscale = 0.8,
  show.edges = FALSE,
  edge.alpha = 0.3,
  node.size = "magnitude",
  label.nodes = TRUE,
  label.size = 3,
  show.usernames = NULL,
  sender.color = "darkred",
  receiver.color = "darkblue",
  colors = NULL,
  title = NULL,
  time_point = NULL,
  plot_type = c("snapshot", "trajectory", "faceted"),
  show_arrows = TRUE,
  highlight = NULL
)
```

Arguments

fit	An object of class "ame" or "lame" containing multiplicative effects, or a network matrix Y if U and V are provided separately
Y	Network matrix (only needed if fit is not provided)
U	Matrix of sender latent factors (extracted from fit if not provided)
V	Matrix of receiver latent factors (extracted from fit if not provided)
row.names	Names for row nodes (defaults to rownames of Y or U)
col.names	Names for column nodes (defaults to colnames of Y or V)

layout	Character string specifying layout: "circle" (default) or "biplot"
vscale	Scaling factor for V positions relative to U (default 0.8)
show.edges	Logical; if TRUE, show network edges (default FALSE)
edge.alpha	Transparency for edges (default 0.3)
node.size	Node size: a numeric value, "degree" to scale by degree (needs Y), or "magnitude" (the default) to scale by the length of each latent vector
label.nodes	Logical; if TRUE, show node labels (default TRUE)
label.size	Size of node labels (default 3)
show.usernames	Which nodes to label: an integer (that many nodes, ranked by latent magnitude when available, else by degree), TRUE for all, a character vector of actor names, or NULL for the default
sender.color	Color for sender/row nodes (default "darkred")
receiver.color	Color for receiver/column nodes (default "darkblue")
colors	Optional vector of colors for nodes (e.g., for communities)
title	Optional title for the plot
time_point	For dynamic UV, which time point to plot (default: last). Can be numeric index or "average" for time-averaged positions
plot_type	For dynamic UV: "snapshot" (single time), "trajectory" (evolution), "faceted" (grid of time points). For static UV, this is ignored.
show_arrows	For trajectory plots, whether to show directional arrows
highlight	Optional character vector of actor names to highlight on a plot_type = "trajectory" plot. Highlighted actors are coloured with the ColorBrewer Dark2 palette; all other actors are rendered in grey at lower alpha. Ignored for static or snapshot plots.

Details

The multiplicative effects in AME models provide a low-rank representation of network structure through latent factors:

U matrix Sender-specific latent positions (row factors)

V matrix Receiver-specific latent positions (column factors)

UV' product Captures dyad-specific effects beyond additive terms

The visualization can show:

Circular layout Default layout placing nodes on a circle with latent positions shown as deviations

Biplot layout Shows U and V positions directly in latent space

Network overlay Optional display of actual network ties

Interpretation (this is an eigenmodel, not a latent-distance model):

- The latent contribution to a tie i -> j is the inner product u_i' v_j: sender and receiver vectors that align (small angle, large magnitude) raise the tie propensity, opposed vectors lower it. Euclidean distance between plotted points is not the operative quantity.
- Actors with similar sender (or receiver) vectors have similar connection patterns; clustering of vectors indicates community structure.

Value

A ggplot2 object that can be further customized

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
# Fit an AME model with multiplicative effects
data(YX_nrm)
# chains = 1 keeps the example short; the default runs two chains and compares them
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE, chains = 1)

# Basic visualization
uv_plot(fit)

# Use biplot layout
uv_plot(fit, layout = "biplot")
```

---

vcov.ame *Posterior covariance of AME model coefficients*

Description

Returns the posterior covariance matrix of regression coefficients.

Usage

```
## S3 method for class 'ame'
vcov(object, ...)

## S3 method for class 'lame'
vcov(object, ...)
```

Arguments

object	fitted AME model (class "ame")
...	additional arguments (ignored)

Value

p x p covariance matrix of posterior BETA draws

`vcov.ame_als` *Sandwich covariance for the regression coefficients of a fast AME fit*

Description

Returns a heteroskedasticity-robust (optionally dyad-clustered) sandwich covariance matrix for the intercept and *dyadic*-covariate coefficients of an `ame_als` fit. This is a fast analytic alternative to the bootstrap for those coefficients.

Usage

```
## S3 method for class 'ame_als'
vcov(object, cluster = c("dyad", "none"), ...)
```

Arguments

`object`	an `ame_als` fit.
`cluster`	`"dyad"` (default) for a dyad-clustered robust meat, or `"none"` for an HC0 (heteroskedasticity-only) meat. Ignored when the fit carries a bootstrap, since the bootstrap covariance is returned.
`...`	ignored.

Details

The estimate is the conditional sandwich $B^{-}MB^{-}$ with bread $B = D'WD$ (D the observed intercept + dyadic-covariate design, W the fit's observation weights) and meat M the heteroskedasticity-robust (`cluster = "none"`, an HC0 meat) or dyad-clustered (`cluster = "dyad"`, the default) outer product of the weighted score contributions $w_\ell e_\ell d_\ell$. For a normal or transform fit the weights are unit, so this reduces to the ordinary $D'D$ sandwich; for an IRLS fit it uses the final IRLS weights, matching the estimating equation the fit actually solved. Dyad clustering pools the score across (i, j), (j, i) and time, so it reflects dyadic dependence (reciprocity, repeated observation).

It is **conditional**: the additive effects `a`, `b` and the multiplicative term are held fixed, so it omits their estimation uncertainty and is anti-conservative. Node-covariate, additive and multiplicative standard errors are not returned – use `ame_als_bootstrap` for those and for fully-propagated inference.

Empirical-Bayes fits. A binary, Poisson or normal fit by the empirical-Bayes estimator (`fit$bias_correct$method == "eb"`, the default for these families; see `ame_als`) returns the Laplace covariance instead of the sandwich: the inverse of the penalised Fisher information of all the parameters (the coefficients, the actor effects, the node-covariate coefficients and the latent factors) at the final weights (for a normal fit the exact information of the Gaussian cell, whose Laplace approximation is exact for the coefficients and the effects), with the actor effects' covariance `fit$eb$sab` and the factor scales held at their estimates. It covers every coefficient, node-covariate coefficients included, and propagates the uncertainty of the actor effects and the factors. The covariance conditional on the factors is `fit$eb$vcov_cond`. A bootstrap attached to the fit takes precedence, as for every other fit.

On a fit carrying the split-sample jackknife (`bias_correct`, from `bias_correct = "jackknife"`; see `ame_als`) the sandwich is the one of the uncorrected estimator's regression block: its bread and

weights are those of the full-data IRLS fit, and its meat uses the residuals of the reported fit. It does not include the variance the correction itself adds (the corrected estimator combines three fits); ame_als_bootstrap refits every replicate with the correction and does.

Value

A covariance matrix with matching row/column names. When the fit carries a $bootstrap, this is the bootstrap covariance over all estimated coefficients, matching coef(). For an empirical-Bayes fit it is the Laplace covariance of all the coefficients. Otherwise it is the conditional sandwich, covering c(intercept, dyadic coefficients) only.

See Also

ame_als_bootstrap for bootstrap uncertainty covering all parameters.

---

vcov.boot_ame *Bootstrap covariance of the regression coefficients*

Description

Returns the sample covariance matrix of the bootstrap replicate intercept + regression coefficients of a boot_ame object – the covariance underlying the reported standard errors and confidence intervals.

Usage

```
## S3 method for class 'boot_ame'
vcov(object, ...)
```

Arguments

object	a boot_ame object.
...	ignored.

Value

A covariance matrix over the intercept and regression coefficients; the rows and columns of a parameter the bootstrap returns no standard error for (withheld) are NA.

`vignette_data` *TIES sanctions data for vignettes*

Description

Longitudinal directed binary network of international economic sanctions among 35 countries across four years (1993, 1994, 1995, 2000), derived from the Threat and Imposition of Sanctions (TIES) dataset (Morgan et al. 2014). Entry $y_{ij,t} = 1$ means country i imposed sanctions on country j in year t. Network density is approximately 2\

Usage

```
data(vignette_data)
```

Format

Four objects:

Y List of 4 binary adjacency matrices (35 x 35), one per year.

Xdyad List of 4 arrays (35 x 35 x 2) of dyadic covariates: `distance` (geographic) and `shared_igos` (shared IGO memberships).

Xrow List of 4 matrices (35 x 2) of sender covariates: `log_gdp` and `log_pop`.

Xcol List of 4 matrices (35 x 2) of receiver covariates: `log_gdp` and `log_pop`.

Details

When loaded via `data(vignette_data)`, the following objects are placed in the calling environment: `Y`, `Xdyad`, `Xrow`, `Xcol`.

References

Morgan, T. Clifton, Bapat, N., & Kobayashi, Y. (2014). Threat and Imposition of Economic Sanctions 1945–2005. *Conflict Management and Peace Science*, 31(5), 541–558.

Examples

```
data(vignette_data)
cat("Countries:", nrow(Y[[1]]), "\n")
cat("Time periods:", length(Y), "\n")
```

waic.ame *WAIC for AME / LAME fits*

Description

S3 method for `waic` that uses the stored `fit$log_lik`.

Usage

```
## S3 method for class 'ame'
waic(x, ...)

## S3 method for class 'lame'
waic(x, ...)

## S3 method for class 'ame_als'
waic(x, ...)
```

Arguments

`x`	A fitted `ame` or `lame` object with `$log_lik`.
`...`	Additional arguments forwarded to `loo::waic.matrix`.

Value

A `waic` object.

Xbeta *Linear combinations of submatrices of an array*

Description

Computes a matrix of expected values based on an array X of predictors and a vector beta of regression coefficients.

Usage

```
Xbeta(X, beta)
```

Arguments

`X`	an n by n by p array
`beta`	a p by 1 vector

Value

An n by n matrix

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

Examples

```
set.seed(1)
n <- 12
X <- array(rnorm(n * n * 2), dim = c(n, n, 2))
beta <- c(0.5, -0.3)
EZ <- Xbeta(X, beta)
dim(EZ)
```

Xcol *Column covariates*

Description

Column covariates loaded by `data(vignette_data)`. See `vignette_data` for full details.

Format

A list of matrices of column covariates

See Also

`vignette_data`

Xdyad *Dyadic covariates*

Description

Dyadic covariates loaded by `data(vignette_data)`. See `vignette_data` for full details.

Format

A list of arrays of dyadic covariates

See Also

`vignette_data`

Xrow *Row covariates*

Description

Row covariates loaded by `data(vignette_data)`. See `vignette_data` for full details.

Format

A list of matrices of row covariates

See Also

`vignette_data`

Y *Relational matrix*

Description

Relational matrix loaded by `data(vignette_data)`. See `vignette_data` for full details.

Format

A list of square binary adjacency matrices

See Also

`vignette_data`

YX_bin *binary relational data and covariates*

Description

a synthetic dataset that includes binary relational data as well as information on eight covariates

Usage

```
data(YX_bin)
```

Format

The format is: List of 2 $ Y: num [1:100, 1:100] NA 0 0 0 0 0 0 0 0 1 ... $ X: num [1:100, 1:100, 1:8] 1 1 1 1 1 1 1 1 1 1 ... ..- attr(*, "dimnames")=List of 3 .. ..$: NULL .. ..$: NULL .. ..$: chr [1:8] "intercept" "rgpa" "rsmoke" "cgpa" ...

Examples

```
data(YX_bin)
gof_stats(YX_bin$Y)
```

`YX_bin_list` *Synthetic longitudinal binary relational data, list-form (latent-scale)*

Description

The list-form sibling of `YX_bin_long`: same synthetic 4-period 50-actor panel reshaped from arrays to lists. The shipped `Y` entries hold the *latent probit-scale predictor* (range roughly -20 to 17), not 0/1 ties. To recover the binary tie indicator the dataset name implies, threshold at zero, e.g. `Y_bin <- lapply(YX_bin_list$Y, function(Yt) 1 * (Yt > 0))`. If passed unthresholded to `lame(..., family = "binary")` the fit will warn and silently apply the same `Y > 0` threshold.

Format

A list with two elements:

Y List of 4 numeric matrices, each `[50, 50]`, with `NA` on the diagonal. Latent probit-scale predictor; threshold at 0 for the binary indicator.

X List of 4 numeric arrays, each `[50, 50, 3]` of dyadic covariates.

`YX_bin_long` *synthetic longitudinal binary relational data (latent-scale)*

Description

a synthetic 50-actor, 4-period dataset used by the vignettes. The shipped `Y` array holds the *latent probit-scale predictor* (range roughly -24 to 20), not 0/1 ties. To recover the binary tie indicator the dataset name implies, threshold at zero: `Y_bin <- (YX_bin_long$Y > 0) * 1`. If passed unthresholded to `lame(..., family = "binary")` the fit will warn and silently apply the same `Y > 0` threshold.

Usage

```
data(YX_bin_long)
```

Format

A list with two elements:

Y Numeric array `[50, 50, 4]`. Latent probit-scale predictor; threshold at 0 to get the binary tie indicator.

X Numeric array `[50, 50, 3, 4]` of dyadic covariates.

Examples

```
data(YX_bin_long)
# threshold latent z to 0/1 before computing binary GOF stats
Yt <- 1 * (YX_bin_long$Y[, , 1] > 0)
diag(Yt) <- NA
gof_stats(Yt)
```

YX_cbin *Censored binary nomination data and covariates*

Description

a synthetic dataset that includes relational data where the number of nominations per row is censored at 10, along with information on eight covariates

Usage

```
data(YX_cbin)
```

Format

The format is: List of 2 $ Y: num [1:100, 1:100] NA 0 0 0 1 0 0 0 0 3 ... $ X: num [1:100, 1:100, 1:8] 1 1 1 1 1 1 1 1 1 1 ... ..- attr(*, "dimnames")=List of 3 .. ..$: NULL .. ..$: NULL .. ..$: chr [1:8] "intercept" "rgpa" "rsmoke" "cgpa" ...

Examples

```
data(YX_cbin)
gof_stats(YX_cbin$Y)
```

YX_frn *Fixed rank nomination data and covariates*

Description

a synthetic dataset that includes fixed rank nomination data as well as information on eight covariates

Usage

```
data(YX_frn)
```

Format

The format is: List of 2 $ Y: num [1:100, 1:100] NA 0 0 0 1 0 0 0 0 3 ... $ X: num [1:100, 1:100, 1:8] 1 1 1 1 1 1 1 1 1 1 ... ..- attr(*, "dimnames")=List of 3 .. ..$: NULL .. ..$: NULL .. ..$: chr [1:8] "intercept" "rgpa" "rsmoke" "cgpa" ...

Examples

```
data(YX_frn)
gof_stats(YX_frn$Y)
```

YX_nrm *normal relational data and covariates*

Description

a synthetic dataset that includes continuous (normal) relational data as well as information on eight covariates

Usage

```
data(YX_nrm)
```

Format

The format is: List of 2 $ Y: num [1:100, 1:100] NA -4.05 -0.181 -3.053 -1.579 ... $ X: num [1:100, 1:100, 1:8] 1 1 1 1 1 1 1 1 1 1 ... ..- attr(*, "dimnames")=List of 3 .. ..$: NULL .. ..$: NULL .. ..$: chr [1:8] "intercept" "rgpa" "rsmoke" "cgpa" ...

Examples

```
data(YX_nrm)
gof_stats(YX_nrm$Y)
```

YX_ord *ordinal relational data and covariates*

Description

a synthetic dataset that includes ordinal relational data as well as information on seven covariates

Usage

```
data(YX_ord)
```

Format

The format is: List of 2 $ Y: num [1:100, 1:100] NA 0 3 0 3 1 0 1 1 0 ... $ X: num [1:100, 1:100, 1:7] 1 1 1 1 1 1 1 1 1 1 ... ..- attr(*, "dimnames")=List of 3 .. ..$: NULL .. ..$: NULL .. ..$: chr [1:7] "rgpa" "rsmoke" "cgpa" "csmoke" ...

Examples

```
data(YX_ord)
gof_stats(YX_ord$Y)
```

Index

∗ **datasets**
addhealthc3, 8
addhealthc9, 9
vignette_data, 254
Xcol, 256
Xdyad, 256
Xrow, 257
Y, 257
YX_bin, 257
YX_bin_list, 258
YX_bin_long, 258
YX_cbin, 259
YX_frn, 259
YX_nrm, 260
YX_ord, 260

ab_plot, 5, *7*, *76*, *180*, *182*
ab_plot.ame_als, 7, *54*
absdiff, *28*
absdiff *(*nodematch*)*, 177
addhealthc3, 8
addhealthc9, 9
als_dynamic_beta, 9, *153*, *159*
als_start_vals, 11, *128*, *132*, *148*
ame, *11*, 12, *30–37*, *39*, *40*, *42*, *50*, *53–55*, *60*, *67*, *70*, *80*, *99*, *114*, *119–121*, *129*, *140*, *144*, *150*, *153*, *155–159*, *162*, *180*, *183*, *235*
ame_als, *11*, *13*, *20*, *26*, *27*, 30, *54–56*, *58*, *59*, *61–63*, *71*, *82*, *115*, *133*, *154*, *159*, *173*, *180*, *186*, *187*, *207*, *248*, *252*
ame_als_bootstrap, *33*, *49–51*, *53*, *55*, 55, *61*, *63*, *76*, *83*, *84*, *100*, *132*, *135*, *157–159*, *226*, *243*, *252*, *253*
ame_als_refit, *49*, *55*, *61*, 61, *248*
ame_memory_settings, 63
ame_memory_usage, 64
ame_options, 65
ame_parallel, 66, *161*
array_to_list, 67
as_draws, 68
as_draws.ame, 69
as_draws.ame_als *(*as_draws.ame*)*, 69
as_draws.lame *(*as_draws.ame*)*, 69
as_lame_y, *26*, 70
autoplot.ame *(*autoplot.lame*)*, 71
autoplot.ame_als, 71
autoplot.lame, *71*, 71
autoplot.lame_als *(*autoplot.ame_als*)*, 71

boot_ame *(*ame_als_bootstrap*)*, 55
boot_ame-no-fitted, 73

check_format, 73
coef.als_dynamic_beta, 75
coef.ame, *29*, 75
coef.ame_als, 76
coef.boot_ame, 77
coef.lame, *153*
coef.lame *(*coef.ame*)*, 75
coef.lame_robust *(*summary.lame_robust*)*, 240
coldwar, 77
combine_ame_chains, *28*, 78, *138*
compact_ame, 79
compute_mcmc_diagnostics, *29*, 80, *138*
compute_mcmc_diagnostics(), *166*, *241*
comtrade, 81
confint, *20*, *135*
confint.ame, *76*, 81
confint.ame_als, *33*, *71*, 82, *158*
confint.boot_ame, *61*, *77*, *83*, 83
confint.lame, *237*
confint.lame *(*confint.ame*)*, 81
confint.lame_robust *(*summary.lame_robust*)*, 240

detect_change_point, 84
dutchcollege, 86
dynamic_beta_prior_summary, 87, *117*

el2sm, 88
evaluate_heldout, 89

fitted, *73*
fitted.ame, 90, *206*
fitted.ame_als, 91, *207*
fitted.boot_ame *(*boot_ame-no-fitted*)*, 73
fitted.lame, 92
fitted.lame_robust *(*summary.lame_robust*)*, 240
forecast_pit, 93
formula.ame, 95
formula.lame *(*formula.ame*)*, 95

glance, 95
glance.ame, 96
glance.ame_als, 97
glance.lame *(*glance.ame*)*, 96
glance.lame_als *(*glance.ame_als*)*, 97
gof, *23*, *29*, 98, *153*, *172*
gof_plot, *23*, *29*, *99*, 100, *103*, *105*, *153*, *172*, *180*, *182*
gof_plot.ame_als, *54*, *100*, 103
gof_stats, 104
gof_stats_bipartite, *104*, 106, *153*
gof_stats_unipartite, 107
gof_temporal, 108

init_dynamic_ab_cpp, 110
IR90s, 111

lame, *10, 11*, *15–22*, *26–29*, *31–37*, *39*, *40*, *42*, *50*, *53, 54*, *56*, *60*, *62*, *67*, *70*, *80*, *99*, 111, *154–159*, *161*, *167*, *182, 183*, *209*, *230–233*, *239*
lame(), *166*
lame_als, *10, 11*, *54–56*, *58, 59*, *61*, *71*, *132, 133*, *153*, 154, *180*, *186*, *207*
lame_multi, *67*, 159
lame_parallel, *160*, 161
lame_resume, *128*, *131*, *135*, *139*, *153*, 162
lame_robust, 163
lame_robust(), *189*, *241*
lame_snap_als, 166
latent_positions, *21*, *29*, *41*, *54*, *76*, *115*, *149*, *153*, 167, *183*, *202*
lazegalaw, 169
lfo, 170
list_to_array, 171

logLik.ame, 172, *176*
logLik.ame_als, 173
logLik.lame *(*logLik.ame*)*, 172
loo, 173, *174*
loo.ame, 174
loo.ame_als *(*loo.ame*)*, 174
loo.lame *(*loo.ame*)*, 174

mhalf, 175

nobs.ame, 176, *177*
nobs.ame_als, 177
nobs.lame *(*nobs.ame*)*, 176
nobs.lame_robust *(*summary.lame_robust*)*, 240
nodefactor, *28*
nodefactor *(*nodematch*)*, 177
nodematch, *28*, 177

per_actor_slopes, 178
plot.ame, 179
plot.ame_als, 180
plot.lame, 181
posterior_options, *19*, *128*, *169*, 182, *229*
posterior_quantiles, 183
predict.ame, *76*, *91*, 184, *193*, *206*
predict.ame_als, *92*, 186
predict.lame, 187
predict.lame_robust, 189
predict.lame_robust(), *166*, *240*
prediction_draws_long, 191
print.als_dynamic_beta, 192
print.ame, 193
print.ame.sim, 194
print.ame_als, 195
print.boot_ame, 195
print.gof_temporal, 196
print.lame, 196
print.lame.sim *(*print.ame.sim*)*, 194
print.lame_multi, 197
print.lame_robust *(*summary.lame_robust*)*, 240
print.lfo_lame, 197
print.per_actor_slopes, 198
print.summary.ame, 198, *235*
print.summary.ame_als, 199
print.summary.boot_ame *(*summary.boot_ame*)*, 236
print.summary.lame, 199, *239*

print.summary.lame_robust
(summary.lame_robust), 240
prior_summary, 200
procrustes_align, *21*, *29*, *153*, *169*, 201

rbeta_ab_bip_gibbs_cpp, 203
read_log_lik, 204
reconstruct_EZ, 204
reconstruct_UVPM (reconstruct_EZ), 204
residuals, *73*
residuals.ame, *91*, 206, *207*
residuals.ame_als, 207
residuals.boot_ame
(boot_ame-no-fitted), 73
residuals.lame, 207
rhat_dynamic_beta, 208
rmvnorm, 209
rUV_dynamic_bip_fc_cpp, 210
rUV_dynamic_fc_cpp, 211
rUV_dynamic_snap_fc_cpp, 212
rUV_dynamic_t_fc_cpp, 213
rUV_sym_fc, 214
rZ_bin_bip_batch_cpp, 215
rZ_nrm_batch_cpp, 215

sample_beta_dynamic_cpp, 217
sample_beta_static_cpp, 218
sample_dynamic_ab_cpp, 219
sample_rho_beta_cpp, 221
sample_sigma_beta_cpp, 222
sampler_describe, 216
sampsonmonks, 222
sheep, 223
simulate.ame, *193*, 223
simulate.ame_als, *54*, *103*, 225
simulate.lame, 226
simulate.lame_robust
(predict.lame_robust), 189
simulate.lame_robust(), *166*
simulate_posterior, *183*, 229
snap_category_summary, 230
snap_index_draws, 231
snap_index_summary, 232
snap_rank_summary, 233
summary.ame, *29*, *76*, *193*, 234
summary.ame.sim, 235
summary.ame_als, 236
summary.boot_ame, 236
summary.lame, *153*, 237
summary.lame.sim, 239
summary.lame_robust, 240

tidy, 241
tidy.ame, 242
tidy.ame_als, 243
tidy.boot_ame, 244
tidy.lame (tidy.ame), 242
tidy.lame_als (tidy.ame_als), 243
trace_plot, *54*, *138*, *180*, *182*, 244

update, *247*, *248*
update.ame, 247, *248*
update.ame_als, 248
update.lame (update.ame), 247
update.lame_als (update.ame_als), 248
uv_plot, *41*, *54*, *169*, *180*, *182*, *202*, 249

vcov.ame, *76*, 251
vcov.ame_als, *53*, *55*, *71*, *82*, *83*, *243*, 252
vcov.boot_ame, *77*, 253
vcov.lame (vcov.ame), 251
vcov.lame_robust (summary.lame_robust),
240
vignette_data, 254, *256*, *257*

waic, *255*
waic (loo), 173
waic.ame, 255
waic.ame_als (waic.ame), 255
waic.lame (waic.ame), 255

Xbeta, 255
Xcol, 256
Xdyad, 256
Xrow, 257

Y, 257
YX_bin, 257
YX_bin_list, 258
YX_bin_long, *258*, 258
YX_cbin, 259
YX_frn, 259
YX_nrm, 260
YX_ord, 260