Package: lame (via r-universe)

October 14, 2026

Title Longitudinal Additive and Multiplicative Effects Models for Networks

Version 1.3.4

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

RoxygenNote 7.3.3

LinkingTo Rcpp, RcppArmadillo

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

Suggests knitr, rmarkdown, testthat (>= 3.0.0), igraph, network, posterior, loo, digest, tibble, broom, generics, pROC, precrec, statmod, dplyr, modelsummary, callr, amen

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

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

Date/Publication 2026-08-04 15:10:03 UTC

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

RemoteRef HEAD

RemoteSha bcd9d7952b7072d192c20f3bcaa07b126c9776d3

Contents

ab_plot . . . 6
ab_plot.ame_als . . . 7
addhealthc3 . . . 8
addhealthc9 . . . 9
als_dynamic_beta . . . 9
als_start_vals . . . 11
ame . . . 11
ame_als . . . 22
ame_als_bootstrap . . . 27
ame_als_refit . . . 30
ame_memory_settings . . . 32
ame_memory_usage . . . 32
ame_options . . . 33
ame_parallel . . . 34
array_to_list . . . 35
as_draws . . . 36
as_draws.ame . . . 37
as_lame_y . . . 38
autoplot.ame_als . . . 39
autoplot.lame . . . 39
boot_ame-no-fitted . . . 41
check_format . . . 41
coef.als_dynamic_beta . . . 43
coef.ame . . . 43
coef.ame_als . . . 44
coef.boot_ame . . . 45
coldwar . . . 45
combine_ame_chains . . . 46

compact_ame . . . 46
compute_mcmc_diagnostics . . . 47
compute_XtX_Xty_bip_cpp . . . 48
comtrade . . . 48
confint.ame . . . 49
confint.ame_als . . . 49
confint.boot_ame . . . 50
design_array_listwisedel . . . 51
detect_change_point . . . 52
dutchcollege . . . 53
dynamic_beta_prior_summary . . . 53
el2sm . . . 55
evaluate_heldout . . . 56
fitted.ame . . . 57
fitted.ame_als . . . 58
fitted.lame . . . 58
forecast_pit . . . 59
formula.ame . . . 60
get_design_rep . . . 61
get_EZ_dynamic_beta_cpp . . . 62
get_fit_object . . . 63
get_start_vals . . . 65
glance . . . 66
glance.ame . . . 66
glance.ame_als . . . 67
gof . . . 68
gof_plot . . . 70
gof_plot.ame_als . . . 73
gof_stats . . . 74
gof_stats_bipartite . . . 76
gof_stats_unipartite . . . 77
gof_temporal . . . 78
init_dynamic_ab_cpp . . . 79
init_dynamic_positions . . . 80
IR90s . . . 81
lame . . . 81
lame_als . . . 95
lame_multi . . . 98
lame_parallel . . . 99
lame_resume . . . 100
lame_snap_als . . . 101
latent_positions . . . 105
lazegalaw . . . 107
lfo . . . 107
list_to_array . . . 108
logLik.ame . . . 109
logLik.ame_als . . . 109
loo . . . 110

loo.ame . . . 110
mhalf . . . 112
nobs.ame . . . 112
nobs.ame_als . . . 113
nodematch . . . 113
per_actor_slopes . . . 114
plot.ame . . . 115
plot.ame_als . . . 116
plot.lame . . . 117
posterior_options . . . 118
posterior_quantiles . . . 119
predict.ame . . . 120
predict.ame_als . . . 121
predict.lame . . . 122
prediction_draws_long . . . 123
print.als_dynamic_beta . . . 124
print.ame . . . 124
print.ame.sim . . . 126
print.ame_als . . . 126
print.boot_ame . . . 127
print.gof_temporal . . . 127
print.lame . . . 128
print.lame_multi . . . 128
print.lfo_lame . . . 129
print.per_actor_slopes . . . 129
print.summary.ame . . . 130
print.summary.ame_als . . . 130
print.summary.lame . . . 131
prior_summary . . . 131
procrustes_align . . . 132
rbeta_ab_bip_gibbs_cpp . . . 134
read_log_lik . . . 135
reconstruct_EZ . . . 135
residuals.ame . . . 136
residuals.ame_als . . . 137
residuals.lame . . . 138
rhat_dynamic_beta . . . 138
rmvnorm . . . 139
rSab_fc . . . 140
rUV_dynamic_bip_fc_cpp . . . 141
rUV_dynamic_fc . . . 142
rUV_dynamic_fc_cpp . . . 142
rUV_dynamic_snap_fc . . . 143
rUV_dynamic_snap_fc_cpp . . . 144
rUV_dynamic_t_fc . . . 146
rUV_dynamic_t_fc_cpp . . . 147
rUV_sym_fc . . . 148
rZ_bin_bip_batch_cpp . . . 149

rZ_nrm_batch_cpp . . . 149
rZ_nrm_fc . . . 150
rZ_pois_fc . . . 150
sample_beta_dynamic_cpp . . . 151
sample_beta_static_cpp . . . 152
sample_dynamic_ab_cpp . . . 153
sample_rho_ab_cpp . . . 154
sample_rho_beta_cpp . . . 155
sample_rho_uv . . . 156
sample_sigma_ab_cpp . . . 157
sample_sigma_beta_cpp . . . 158
sample_sigma_uv . . . 158
sampler_describe . . . 159
sampsonmonks . . . 160
sheep . . . 160
simulate.ame . . . 161
simulate.ame_als . . . 164
simulate.lame . . . 164
simulate_posterior . . . 168
simY_pois . . . 169
snap_category_summary . . . 170
snap_index_draws . . . 171
snap_index_summary . . . 171
snap_rank_summary . . . 172
summary.ame . . . 173
summary.ame.sim . . . 174
summary.ame_als . . . 174
summary.boot_ame . . . 175
summary.lame . . . 175
summary.lame.sim . . . 177
tidy . . . 177
tidy.ame . . . 178
tidy.ame_als . . . 179
tidy.boot_ame . . . 180
trace_plot . . . 180
update.ame . . . 182
update.ame_als . . . 183
uv_plot . . . 184
vcov.ame . . . 186
vcov.ame_als . . . 187
vcov.boot_ame . . . 188
vignette_data . . . 188
waic.ame . . . 189
Xbeta . . . 190
Xbeta_bip_cpp . . . 190
Xcol . . . 191
Xdyad . . . 191
Xrow . . . 191

Y 192
YX_bin 192
YX_bin_list 193
YX_bin_long 193
YX_cbin 194
YX_frn 194
YX_nrm 195
YX_ord 195

Index **197**

`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"),
  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) or "receiver"
`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)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE)

# 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.

---

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 sociomatrix (Y) and 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: 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(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), `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`.

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 = 1,
  use_sparse_matrices = FALSE,
  method = c("mcmc", "als"),
  bootstrap = 0L,
  bootstrap_type = c("parametric", "block"),
  bootstrap_block_length = 1L,
  bootstrap_seed = NULL,
  save_log_lik = FALSE,
  ordinal_cutpoints = c("data_induced", "explicit"),
  print,
  ...,
  model.name = 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.

`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.

`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

`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 defaults to `diag(2)`. 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 unipartite continuous-family fit, round(4 *

vdfmlt) for a unipartite discrete-family fit, and 4 + 3 * (nA + nB) / 200 for a bipartite fit – except bipartite `"binary"`, which also uses round(4 * vdfmlt) – 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 Prior scale for the dyadic residual variance v_e (default: 1), entering the inverse-gamma draw as a pseudo-observation scale. Larger values pull v_e upward. The regression-coefficient prior is controlled separately by `g`, not by `s20`.

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 defaults to `diag(2 * R)` times a scale estimated from the data.

kappa0 Prior degrees of freedom for the multiplicative-effects covariance Σ_{uv} (default when R > 0: 2 * R + 2, times the same probit-moment multiplier as `eta0` for a unipartite discrete-family fit). Higher values impose stronger shrinkage of the latent factors toward the prior scale `kappa0 * Suv0`.

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** for the Zellner g-prior on regression coefficients (`beta ~ N(0, g * sigma^2 * solve(XtX))` where `XtX` is the design cross-product). If not specified, defaults are: for `normal` family, `g = n * var(Y)`; for other families, `g = n` (number of non-missing dyads). Per-coefficient (vector) `g` is not currently supported by the unipartite path – pass a scalar. *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. thin_UV / thin_ab thin the stored draws. Default NULL.

n_chains — integer: number of MCMC chains to run (default: 1)

cores — integer: number of cores for parallel chains (default: 1)

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, prior, ...) are warned about and ignored; the call forwards to ame_als.

bootstrap — integer (only used when method = "als"): number of bootstrap replicates. 0 (default) skips the bootstrap; N > 0 runs N replicates and attaches the result so that confint returns bootstrap intervals.

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.

save_log_lik — logical: when TRUE, attach a per-iteration pointwise log-likelihood matrix fit$log_lik (an n_iter x n_obs array), required for loo(fit) and waic(fit). For families normal, binary, cbin, poisson, and ordinal this is the exact observed-data log

density of Y (`fit$log_lik_method = "observed_exact"`); only frn falls back to the augmented-Z normal approximation on the latent scale (with a one-time warning). Default `FALSE` (no log_lik storage; default fit is byte-identical to previous releases).

`ordinal_cutpoints`
: character: cutpoint convention for `family = "ordinal"`. `"data_induced"` (default) uses the data-induced cutpoints; `"explicit"` samples explicit cutpoints via a Cowles (1996) Metropolis-Hastings update. 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.

`model.name`
: optional string for model selection output

Details

This command provides posterior inference for parameters in AME models of cross-sectional relational data, assuming one of eight 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 an arc-sine prior on $(-1, 1)$ (density proportional to $(1 - \rho^2)^{-1/2}$), 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; `R > floor(n/3)` is rarely identifiable and will issue a warning. 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 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: arc-sine prior on $(-1, 1)$, density $p(\rho) \propto (1 - \rho^2)^{-1/2}$, 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 always uses `Sab0 = diag(2)`. 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. In Social Relations Model simulations the data-scaled default recovers small true variances with modest bias where a fixed `diag(2)` prior can nearly double them; the data-scaled prior is mildly more conservative in the opposite regime, when the true variance is large relative to the residual scale.

This default differs from the `amen` package, whose `ame(family = "nrm")` leaves `Sab0 = diag(2)` and `eta0 = 4`. To reproduce `amen`'s additive-effects posterior, pass `prior = list(Sab0 = diag(2), eta0 = 4)`. The choice affects only `va`, `vb` and (weakly) `cab`; `ve` and `rho` are unchanged.

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 in this model.

"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 AME model for count data: $Y \sim \text{Poisson}(\exp(z))$ with $z \sim N(\eta, \sigma^2)$, a lognormal-mixed Poisson. The conditional mean given the latent z is $\exp(z)$; the marginal mean is $\exp(\eta + \sigma^2/2)$, not $\exp(\eta)$.

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). 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)
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	Posterior mean of Y on response scale (for predictions and imputing missing values)

Metadata:

family	Model family (normal, binary, etc.)
mode	Network mode (unipartite or bipartite)
symmetric	Logical indicating if network is symmetric
R	Dimension of multiplicative effects

Optional Posterior Samples (if requested via posterior_options):

U_samples	Samples of U (n xR xiterations array)
V_samples	Samples of V (m xR xiterations array)
a_samples	Samples of row effects (n xiterations matrix)
b_samples	Samples of column effects (m xiterations matrix)

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`. `family = "ordinal"` with `symmetric = TRUE` is supported via the dedicated sampler in `R/rZ_ord_sym_fc.R`, which uses the symmetric-doubled precision and mirrors upper-triangle draws to the lower triangle so $Z = t(Z)$ holds at every sweep.

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 validator `any(is.finite(Y - t(Y)))` evaluates `FALSE` and the call proceeds: the sampler then treats the populated triangle as the symmetric data and mirrors it. This is usually intended, but if the upper / lower triangles were meant to differ, the model is silently fitting half the data. Audit `anyNA(Y[upper.tri(Y)]) != anyNA(Y[lower.tri(Y)])` before calling if you are unsure.

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.

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`.

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).

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_bin)
fit <- ame(YX_bin$Y, Xdyad = YX_bin$X, burn = 10, nscan = 100, odens = 1,
           family = "binary", verbose = FALSE)
summary(fit)
# Note: you should run the Markov chain much longer in practice
```

`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.

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"),
  bootstrap = 0L,
  bootstrap_type = c("parametric", "block"),
  bootstrap_block_length = 1L,
  bootstrap_seed = NULL,
  verbose = TRUE,
  seed = 6886
)
```

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`.
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.
max_iter	maximum number of block coordinate descent iterations (default 200).
tol	convergence tolerance on the relative change in residual sum of squares (default `1e-6`).
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, `"irls"` (default for `binary`/`poisson`) runs iteratively reweighted least squares, giving a fast approximate 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 lowest-SSE fit, warning when the starts reach materially different optima. Reproducible given `seed`; the global RNG stream is left unchanged.

`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. 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`.

`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`). The block coordinate descent is deterministic, so the point estimate is reproducible regardless; the argument is retained for API consistency with `ame`.

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 (with a `ginv` pseudoinverse fallback for rank-deficient designs); Gauss-Seidel sweeps to convergence for the additive effects (a, b); 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 may need more iterations to converge, so raise `max_iter` if the fit reports non-convergence.

For `family = "normal"` this 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`. 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 and the slope inflation is bounded and stable in R; a residual finite-sample (incidental-parameters) upward bias remains, on the order of +10-15\ posterior mean). This residual is a property of the estimator class, not a code defect: because a nonlinear-likelihood point estimator profiles the $2Rn$ latent factors at their joint optimum rather than integrating them, it cannot be consistent for the regression coefficient (Neyman-Scott). The bias is dominated by *rank over-specification* – it is largest when R is set higher than the data support (the factors then absorb noise) and is only ~5\ `ame()` / `lame()` path integrates the factors out and is verified calibrated by simulation-based calibration, so binary `R > 0` ALS is best used as a fast exploratory / rank-selection score, with `ame(..., method = "mcmc")` for final coefficient inference (`bootstrap = N` gives ALS-scale intervals but reproduces, rather than removes, this bias). 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.

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. 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): 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.

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 `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` with five descriptive entries:

`va`, `vb` empirical variances of the sender and receiver additive effects.

`cab` covariance of the sender and receiver effects (`NA` for symmetric or bipartite models).

`rho` dyadic residual reciprocity — the correlation between the residuals of (i, j) and (j, i) — not the sender/receiver correlation `cor(a, b)`.

`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`.

These are descriptive summaries of the point estimates, not random-effect variance components. 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 point estimator for normal, binary, and poisson panels, including named panels where actors enter or exit: `dynamic_ab`, selected intercept/dyadic/node `dynamic_beta`, and AR(1) or Student-t `dynamic_uv` for directed, symmetric, and bipartite panels. The snap-only `dynamic_uv = TRUE, dynamic_uv_kind = "snap"` case routes to `lame_snap_als` for supported normal unipartite and bipartite panels. 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. bipartite `dynamic_g` is available on the dynamic als path for normal, binary, and poisson panels. changing actor composition requires row and column names on every slice so actors can be aligned; smoothing penalties are broken across actor-entry gaps. 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.** ALS has no priors. `prior = list(...)` and `g = ...` are MCMC-only; the dispatcher `ame(..., method = "als")` warns and ignores them.
- **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; `n_chains` is dropped.

- **Convergence diagnostics.** No Rhat / ESS / trace_plot.

Features that work the same on ALS fits: coef, vcov (sandwich on regression block), confint (auto-routes bootstrap intervals when present, sandwich Wald 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 (requires T > 1); with few slices its intervals are necessarily coarse.

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. Replicates that error or return non-finite values are dropped and counted (n_valid / n_total). Standard errors are replicate column standard deviations; confidence intervals use the percentile method.

Usage

```
ame_als_bootstrap(
  object,
  R = 200,
  type = c("parametric", "block"),
  block_length = 1,
  seed = NULL,
  verbose = TRUE
)

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

Arguments

object	an ame_als object from ame_als or lame_als.
R	integer number of bootstrap replicates (default 200).
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).

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 is well-determined only when the multiplicative singular values are well separated; with near-equal singular values the per-column `U`/`V` standard errors reflect an unstable rotation, and the subspace — or, for symmetric models, the eigenvalues `L` — is the summary of record. For a symmetric model the eigenvalues `L` of the multiplicative term are aggregated and reported as the primary multiplicative-uncertainty summary.

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); with only a few time slices it is necessarily coarse, its intervals correspondingly imprecise and somewhat anti-conservative, so `parametric` is the default. A variance component fit with `R > 0` can still carry a small residual parametric-bootstrap bias (the low-rank refit re-absorbs simulated noise); `summary()` flags any point estimate that falls outside its interval. A binary IRLS fit can carry a finite-sample (incidental-parameters) bias in the point estimator itself, which the bootstrap reproduces rather than removes. 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 `n_valid`, `n_total`, `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.

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

```
Y <- replicate(6, { 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
)
```

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).
tol	convergence tolerance (default 1e-5).
verbose	logical; print progress (default FALSE).

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.

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

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

---

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

Run AME model with multiple parallel chains

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 parallel chains to run (default = 4)
cores	Number of CPU cores to use for parallel processing. 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

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

---

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).

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(..., n_chains = N)) 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.
...	ignored.

Details

The reshape uses fit$chain_indicator (set when n_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)
fit <- lame(YX_bin_list$Y, YX_bin_list$X, family = "binary", R = 0,
            dynamic_beta = "dyad",
            nscan = 60, burn = 15, odens = 5, verbose = FALSE)
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

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 : https://polisci.la.psu.edu/people/xuc11/

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

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

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

---

compute_mcmc_diagnostics

Compute MCMC convergence diagnostics for multiple chains

Description

Computes Gelman-Rubin 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 fitted ame/lame objects, one per chain (each must carry a BETA matrix).

Value

A list with rhat, ess, param_names (over BETA and VC), n_chains and n_samples.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

compute_XtX_Xty_bip_cpp *Compute X'X and X'y for bipartite covariate regression*

Description

Replaces the O(T x p^2 x n^2) nested R loop for bipartite XtX and Xty computation with a single C++ call.

Usage

```
compute_XtX_Xty_bip_cpp(Xlist, resid, p)
```

Arguments

`Xlist`	List of T arrays, each nA x nB x p
`resid`	3D array of residuals nA x nB x T
`p`	Number of covariates

Value

List with XtX (p x p) and Xty (p vector)

comtrade *Comtrade data*

Description

Eleven years of import and export data between 229 countries. The data use the SITC Rev. 1 commodity classification, aggregated at the first level (AG1).

Format

A list consisting of a socioarray `Trade` and a vector `dollars2010` of inflation rates. The socioarray gives yearly trade volume (exports and imports) in dollars for 10 different commodity classes for eleven years between 229 countries. This gives a five-way array. The first index is the reporting country, so `Trade[i,j,t,k,1]` is what `i` reports for exports to `j`, but in general this is not the same as `Trade[j,i,t,k,2]`, what `j` reports as importing from `i`.

Source

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. `NULL` (default) returns intervals for all available parameters.
`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`).

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

These intervals 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 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.

---

design_array_listwisedel

Computes the design socioarray of covariate values

Description

Computes the design socioarray of covariate values for an AME fit

Usage

```
design_array_listwisedel(Xrow=NULL,Xcol=NULL,Xdyad=NULL,intercept=TRUE,n)
```

Arguments

Xrow	an n x pr matrix of row covariates
Xcol	an n x pc matrix of column covariates
Xdyad	an n x n x pd array of dyadic covariates
intercept	logical
n	number of rows/columns

Value

an n x n x (pr+pc+pd+intercept) 3-way array

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

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

Description

For each dynamic coefficient in a `dynamic_beta` fit, compares the posterior distribution of the maximum scaled first-difference $M = \max_t |\beta_t - \beta_{t-1}|/\sigma_\beta$ to a Monte-Carlo approximation of the same statistic under the AR(1) prior. The returned `bf` column is the tail-ratio score $\Pr(M^{post} > m^*)/0.05$, where m^* is the 95\ quantile of M^{prior}. It is kept under the historical column name for compatibility, but it is not a marginal-likelihood Bayes factor. Use it to surface posterior temporal jumps that the AR(1) prior cannot comfortably accommodate.

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 prior simulations for the null distribution. Default `2000`.
`threshold_bf`	Cutoff for the "warn" column. Default `10`; lower to 5 if the false-positive rate on AR(1) data is too high.
`seed`	Optional seed for reproducibility.

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`).

Examples

```
data(YX_bin_list)
fit <- lame(YX_bin_list$Y, YX_bin_list$X, family = "binary", R = 0,
            dynamic_beta = "dyad",
            nscan = 60, burn = 15, odens = 5, verbose = FALSE)
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")
```

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"`.

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).

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
fit <- ame(Y_train, R = 0, family = "binary",
           burn = 15, nscan = 60, odens = 5, verbose = FALSE, plot = FALSE)
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 posterior mean of the network on the response scale (YPM). For binary models, these are predicted probabilities between 0 and 1. For normal models, these are predicted continuous values. For Poisson models, these are predicted counts.

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, ...)
```

Arguments

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

Value

A matrix or list of matrices of fitted values.

fitted.lame *Extract fitted values from LAME model*

Description

Extract fitted values from LAME model

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)
```

Arguments

fit	A fitted lame 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 fit$YPM[[t]] elements.
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.

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 is computed analytically from the per-draw forecast variance. For discrete families ("binary", "poisson", "ordinal") the randomised PIT of Czado, Gneiting & Held (2009) is used. For rank families ("cbin", "frn") PIT is not implemented; the function returns NULL with an informational note.

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 (KS p-value), cover_95 (fraction of PIT values in [0.025, 0.975]).

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 <- YX_bin_list$Y[1:3]
Y_test <- YX_bin_list$Y[4]
X_train <- YX_bin_list$X[1:3]
fit <- lame(Y_train, Xdyad = X_train,
            family = "binary", R = 0,
            dynamic_beta = "dyad",
            nscan = 60, burn = 15, odens = 5, verbose = FALSE)
pit <- forecast_pit(fit, y_future = Y_test)
pit$ks_p
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.

get_design_rep *Create design array for replicate data*

Description

Create design array for replicate data

Usage

```
get_design_rep(Y, Xdyad, Xrow, Xcol, actorSet, intercept, n, N, pr, pc, pd)
```

Arguments

Y	dependent variable in array format
Xdyad	dyadic covariates in array format
Xrow	sender covariates in array format
Xcol	receiver covariates in array format
actorSet	vector of actors
intercept	logical indicating whether to include intercept
n	number of actors
N	number of replicates
pr	number of receiver covariates
pc	number of sender covariates
pd	number of dyadic covariates

Value

returns list of design array values necessary for ame_repL

Author(s)

Shahryar Minhas

`get_EZ_dynamic_beta_cpp` *Compute EZ when beta is time-varying*

Description

Rebuild the (n x n x T) or (nA x nB x T) EZ cube using a per-period beta vector. Mirrors get_EZ_cpp / get_EZ_bip_cpp but accepts a (T x p) beta matrix where each row is the beta for that period.

Usage

```
get_EZ_dynamic_beta_cpp(
  Xlist,
  beta_full_path,
  a_mat,
  b_mat,
  U_cube,
  V_cube,
  G,
  bipartite,
  symmetric
)
```

Arguments

`Xlist`	T-length list of design arrays (each n x n x p or nA x nB x p).
`beta_full_path`	(T x p) matrix of per-period beta. For coefficients that are not in the dynamic block, the rows are identical (the static beta replicated across all periods).
`a_mat`	(n_a x T) row effects (or repmat-ed static effects).
`b_mat`	(n_b x T) column effects.
`U_cube`	(n_u x R x T) row latent positions (or replicated-static).
`V_cube`	(n_v x R x T) column latent positions.
`G`	(R x R or RA x RB) interaction matrix (bipartite); identity for unipartite.
`bipartite`	TRUE/FALSE.
`symmetric`	TRUE/FALSE.

Value

n_a x n_b x T cube of EZ values.

get_fit_object *Get fitted object from MCMC results*

Description

Get fitted object from MCMC results

Usage

```
get_fit_object(
  APS,
  BPS,
  UVPS,
  YPS,
  BETA,
  VC,
  GOF,
  Xlist,
  actorByYr,
  colActorByYr = NULL,
  start_vals,
  symmetric,
  tryErrorChecks,
  model.name = NULL,
  U = NULL,
  V = NULL,
  dynamic_uv = FALSE,
  dynamic_ab = FALSE,
  bip = FALSE,
  rho_ab = NULL,
  rho_uv = NULL,
  family = NULL,
  odmax = NULL,
  nA = NULL,
  nB = NULL,
  n_time = NULL,
  Y_obs = NULL,
  G = NULL,
  dynamic_beta = FALSE,
  beta_dynamic_mask = NULL,
  beta_dynamic_groups = NULL,
  rho_beta = NULL,
  sigma_beta = NULL,
  RHO_BETA = NULL,
  SIGMA_BETA = NULL,
  dynamic_rho = FALSE,
  RHO = NULL,
```

```
  rho_path = NULL
)
```

Arguments

APS	summed additive sender random effects (or matrix for dynamic)
BPS	summed additive receiver random effects (or matrix for dynamic)
UVPS	summed multiplicative random effects
YPS	summed Y posterior predictive values
BETA	Matrix of draws for regression coefficient estimates
VC	Matrix of draws for variance estimates
GOF	Matrix of draws for goodness of fit calculations
Xlist	List based version of design array
actorByYr	List of actors by time point. In bipartite mode this is the per-year list of row actors.
colActorByYr	Bipartite only. List of column actors by time point; defaults to NULL (unipartite).
start_vals	start_vals for future model run
symmetric	logical indicating whether model is symmetric
tryErrorChecks	list with counts of MCMC errors
model.name	Name of the model (optional)
U	Latent sender positions (optional, for dynamic UV)
V	Latent receiver positions (optional, for dynamic UV)
dynamic_uv	logical indicating whether UV effects are dynamic
dynamic_ab	logical indicating whether additive effects are dynamic
bip	logical indicating whether the network is bipartite
rho_ab	temporal correlation parameter for additive effects (optional)
rho_uv	temporal correlation parameter for multiplicative effects (optional)
family	character string specifying the model family (e.g., "binary", "normal", "poisson")
odmax	vector of maximum ranks for ordinal or fixed rank nomination families
nA	number of actors in first mode (for bipartite networks)
nB	number of actors in second mode (for bipartite networks)
n_time	number of time periods (for longitudinal models)
Y_obs	original observed network (stored for residuals computation)
G	bipartite interaction matrix mapping row to column latent spaces
dynamic_beta	logical or scalar; whether the BETA storage is 3-D (dynamic_beta path). Default FALSE.
beta_dynamic_mask	logical vector marking which coefficients are dynamic.

beta_dynamic_groups	character vector of per-coefficient block labels ("intercept", "dyad", "row", "col"); "" for static coefficients.
rho_beta	named numeric vector of per-block AR(1) rho values (one per dynamic block).
sigma_beta	named numeric vector of per-block AR(1) innovation standard deviations.
RHO_BETA	matrix of per-iteration rho_beta draws (rows = MCMC draw, cols = dynamic block).
SIGMA_BETA	matrix of per-iteration sigma_beta draws.
dynamic_rho	logical indicating whether residual dyadic reciprocity varies by period.
RHO	matrix of per-iteration, per-period dyadic reciprocity draws.
rho_path	numeric vector of period-specific dyadic reciprocity values.

Value

Fitted AME object

Author(s)

Shahryar Minhas

---

get_start_vals *Get fitted object from MCMC results*

Description

Get fitted object from MCMC results

Usage

```
get_start_vals(start_vals, Y, family, xP, rvar, cvar, R, odmax = NULL)
```

Arguments

start_vals	List object that is null or contains starting values
Y	dependent variable in array format
family	character vector (e.g. 'bin', 'nrm') specifying family type
xP	number of exogenous covariates
rvar	logical indicating whether to include sender random effects
cvar	logical indicating whether to include receiver random effects
R	Number of dimensions for multiplicative effects
odmax	vector of maximum ranks for cbin/frn families (optional)

Value

List of starting values for MCMC

Author(s)

Shahryar Minhas

`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.

`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)
fit <- lame(YX_bin_list$Y, YX_bin_list$X, family = "binary", R = 0,
            nscan = 100, burn = 20, odens = 5, verbose = FALSE)
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_bin_list)
Y1 <- 1 * (YX_bin_list$Y[[1]] > 0); diag(Y1) <- NA
fit <- ame_als(Y = Y1, Xdyad = YX_bin_list$X[[1]],
               family = "binary", 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 re-running 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) - same format as for ame()
nsim	Number of posterior predictive simulations to generate (default 100). If NULL, uses all available posterior samples.
verbose	Logical; print progress information

Details

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)`

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 matrix of GOF statistics with the same format as if gof=TRUE was used during model estimation. First row contains observed statistics, subsequent rows contain posterior predictive statistics.

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

See Also

ame, lame, gof_plot

Examples

```
# Run model without GOF during fitting
data(YX_nrm)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2, gof = FALSE,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE)

# 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", "four.cycles". 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).

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. Captured through bipartite multiplicative effects with appropriate R_row, R_col.

Interpretation Guide:

Histogram plots (static models):

- Dashed orange vertical line (Okabe-Ito `#D55E00`): observed statistic value (dual-encoded on colour and linetype)
- Grey histogram: distribution from posterior predictive simulations
- Good fit: orange line falls within the bulk of the histogram
- Poor fit: orange 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)
fit_ame <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, gof = TRUE,
               nscan = 100, burn = 10, odens = 1, verbose = FALSE)

# 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) on each replicate, and overlays the observed value on a histogram of replicate values.

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

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.

`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)
```

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.

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")

# 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)
```

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.

Details

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.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)
```

---

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 `p_pp = 2 * min(p_up, 1 - p_up)`, which runs from 0 (observed slope in the extreme tail) to 1 (observed slope dead-centre). A static fit on truly trending data yields `p_pp` near 0; a dynamic fit that captures the trend yields `p_pp` near 1.

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; picks "reciprocity" for unipartite directed fits with $T \geq 3$ and family normal / poisson, where "density" is constant and uninformative; otherwise "density"), "density", "mean" (mean of off-diagonal Y; the right "density" analogue for continuous outcomes), "reciprocity", "transitivity".
n_rep	Number of posterior-predictive replicates to draw (each replicate is a full T-period network from simulate(fit)).
seed	Optional RNG seed.

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)

Examples

```
data(YX_bin_list)
fit <- lame(YX_bin_list$Y, YX_bin_list$X, family = "binary", R = 0,
            dynamic_beta = "dyad",
            nscan = 60, burn = 15, odens = 5, verbose = FALSE)
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

---

init_dynamic_positions

Initialize dynamic latent positions with AR(1) structure

Description

Initialize dynamic latent positions with AR(1) structure

Usage

```
init_dynamic_positions(n, R, Tn, rho_uv, sigma_uv)
```

Arguments

n	Number of actors
R	Latent dimension
Tn	Number of time points
rho_uv	AR(1) parameter
sigma_uv	Innovation standard deviation

Value

3D array of latent positions (n x R x Tn)

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,
  bootstrap = 0L,
  bootstrap_type = c("parametric", "block"),
  bootstrap_block_length = 1L,
  bootstrap_seed = NULL,
  resume_from = NULL,
  print
)
```

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)?
cvar	logical: fit column random effects (asymmetric case)?
dcor	logical: fit a dyadic correlation (asymmetric case)?
nvar	logical: fit nodal random effects (symmetric case)?
R	integer: dimension of the multiplicative effects (can be zero)
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. fitU / fitV store only the posterior-mean

trajectory cubes; draw-level trajectory uncertainty requires refitting with `posterior_opts = list(save_UV_draws = TRUE)`, which attaches the per-iteration cubes as `fit$U_draws` / `fit$V_draws`. Default `FALSE`.

`dynamic_ab` logical: fit dynamic additive effects (sender/receiver effects) that evolve over time using AR(1) processes. When TRUE, the row effects (a) and column effects (b) become time-varying, following $a_{i,t} = \rho_{ab} a_{i,t-1} + \epsilon_{i,t}$. 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.

`dynamic_G` logical (bipartite only). When TRUE the bipartite interaction matrix G_t in $U_t G_t V_t'$ varies by period and is returned as `fit$G_cube`, an $R_{\text{row}} \times R_{\text{col}} \times T$ array. MCMC fits also return posterior summaries for this cube; ALS fits return the penalized point path using the same storage names. The marginal bilinear predictor is identified under the package's canonicalization conventions, while individual entries of G_t can move under equivalent rotations and scalings of U_t and V_t. 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, defaults shape 2 / scale 1). 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`. Note: with a single dynamic coefficient block and small T, the AR(1) parameter ρ_β sees only $T - 1$ transitions and is strongly informed by its default truncated-Normal prior (mean 0.8, sd 0.15) rather than the data; if ρ_β itself is of interest, check prior sensitivity (see `dynamic_beta_prior_summary`). 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. MCMC fits store per-draw paths in `fit$RHO` and the posterior mean path in `fit$rho_path`. ALS dynamic fits report a residual `rho_path` that can be passed to MCMC with `als_start_vals`. 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).

`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 currently supported for unipartite directed and symmetric models only. With `method = "als"`, `"snap"` is also supported for normal bipartite panels when it is the only dynamic block and G is static. With `method = "als"`, `"t"` is supported for bipartite normal, binary, and Poisson panels. ALS Student-t dynamic UV fits attach the final local transition-weight matrices as `fit$lambda_u` and `fit$lambda_v`.

`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?

`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

`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 the `"normal"` and `"poisson"` families, and for unipartite `"binary"`, this defaults to `diag(2)` scaled to the observed

sender/receiver heterogeneity in `Y` rather than to a fixed `diag(2)`; see `ame` for the rationale. 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 `"binary"` 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 prior is fixed in the longitudinal sampler; `Suv0` and `kappa0` apply to `ame` only.

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 Minimum observed sender/receiver cells used before a static additive effect is left unshrunk (default: max(10, T)). Actors with less support are pulled toward the prior mean for that role.

rho_uv_mean For dynamic_uv=TRUE: Prior mean for UV AR(1) parameter (default: 0.9). Values close to 1 indicate high temporal persistence.

rho_uv_sd For dynamic_uv=TRUE: Prior SD for UV AR(1) parameter (default: 0.1). Controls uncertainty about temporal dependence.

sigma_uv_shape For dynamic_uv=TRUE: Shape parameter for inverse-gamma prior on UV innovation variance (default: 2).

sigma_uv_scale For dynamic_uv=TRUE: Scale parameter for inverse-gamma prior on UV innovation variance (default: 1).

uv_max_abs For dynamic_uv=TRUE: loose bound on the raw latent coordinate scale after the product-preserving normalization step (default: 50). This guards the unidentified coordinate scale without changing the fitted $U_t V_t'$ bilinear term.

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 for inverse-gamma prior on additive effects innovation variance (default: 1).

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 for the inverse-Gamma prior on the per-block innovation variance (default: 2).

sigma_beta_scale For dynamic_beta: Scale parameter for the inverse-Gamma prior (default: 1).

sigma_beta_init For dynamic_beta: Initial value of the per-block innovation standard deviation (default: 0.25). Affects mixing, not the stationary distribution.

beta0_mean For dynamic_beta: Mean of the Gaussian prior on the dynamic coefficients at t = 0 (default: 0).

beta0_var For dynamic_beta: Variance of the Gaussian prior on the dynamic coefficients at t = 0 (default: 10). Weakly informative.

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 = 0.1) for very smooth time-varying coefficients with tight innovations.

`g` optional scalar or vector for g-prior on regression coefficients. Default is p^2 where p is the number of regression parameters. 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.

`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. Use `als_start_vals` to start MCMC from an ALS fit.

`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, 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 (the ame spelling) 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 fitU_draws / fitV_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]), except under dynamic_G = TRUE whose per-period cube already gets posterior summaries. save_ab is not available on the longitudinal path and is ignored with a warning; thin_UV / thin_ab are accepted for posterior_options compatibility but 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. With a single dynamic block the value is recorded but there is no other block to pool against.

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 margin. 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. Draws are stored in fit$THETA_ACTOR when retained; summary mode stores fit$theta_actor_mean and fit$theta_actor_sd.

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) or "drop_population". "center" preserves the population coefficient and constrains per-actor deviations to sum to zero per period via pairwise contrast FFBS. "drop_population" is reserved and currently falls back to "center".

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 Matérn 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.

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.

checkpoint_path
: optional file path. When set, the chain periodically writes a snapshot of BETA, VC, the RNG state, and the 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". Selects the argument for selecting which pointwise log-lik is stored on fit$log_lik. "observed_exact" uses the closed-form marginal log-likelihood for normal, binary, cbin, poisson, and ordinal fits. "observed_ghk" uses a randomized-Halton GHK Monte Carlo marginal for rank-family fits ("frn") and the exact observed likelihood when no GHK step is needed. "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 explicit cutpoints via a Cowles (1996) Metropolis-Hastings update. 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 penalized point-estimation path for `dynamic_ab`, selected `dynamic_beta`, smooth AR(1) `dynamic_uv` in directed, symmetric, and bipartite panels, bipartite `dynamic_G`, and Student-t `dynamic_uv` in directed, symmetric, and bipartite panels. The snap-only `dynamic_uv = TRUE, dynamic_uv_kind = "snap"` case routes to `lame_snap_als` for supported normal unipartite and bipartite panels. 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 and actor-entry gaps break the dynamic smoothing penalties. Posterior draws, rank/censored families, and bipartite snap-shift fits with node covariates, dynamic coefficients, or dynamic `G_t` are not handled by the snap ALS shortcut.

`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`. Smooth dynamic ALS fits report fitted-surface and coefficient-path differences across starts; snap ALS fits report snap-score, snap-class, top-period, and fitted-surface differences.

`als_max_iter` positive integer (only used when `method = "als"`): maximum block-coordinate iterations for the ALS fit. Default `1000`.

`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.

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

`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.

`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 stored draws on the continuation. Note: `checkpoint_path` and `max_seconds` cannot currently be overridden on a resume (they are stripped before re-evaluating the saved call); open both at the original `lame()` call if you need iterative checkpointing or a time budget on a chain of resumes. Pass `verbose = FALSE` on resume (the sampling progress bar is gated on `burn != 0` and the continuation forces `burn = 0`). Default `NULL`.

`print` Deprecated. Use `verbose` instead.

Details

This command provides posterior inference for parameters in AME models of longitudinal relational data, assuming one of eight 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.

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 unipartite directed and symmetric models. With `method = "als"`, Student-t dynamic UV is also supported for bipartite normal, binary, and Poisson panels, and snap ALS is supported for normal unipartite and bipartite panels, including named panels where actors enter or exit.

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): The sender (a) and receiver (b) effects evolve as:

$$a_{i,t} = \rho_{ab} a_{i,t-1} + \epsilon_{i,t}$$

$$b_{i,t} = \rho_{ab} b_{i,t-1} + \eta_{i,t}$$

where $\epsilon_{i,t}, \eta_{i,t} \sim N(0, \sigma_{ab}^2)$. This models time-varying individual activity levels (outdegree) and popularity (indegree).

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:

- $\rho_{uv}, \rho_{ab} \sim TruncNormal(mean, sd, 0, 1)$: 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 increase computation by roughly 30 to 50 percent per iteration
- 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
- Recommend burn >= 1000 and nscan >= 20000 for dynamic models

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.

When dynamic_G = TRUE the bipartite interaction matrix G_t varies by period (a separate $R_{\text{row}} \times R_{\text{col}}$ matrix at every time slice) and is returned as fit$G_cube. The MCMC estimator samples G_t with a Carter-Kohn/FFBS update; method = "als" estimates a penalized point path for normal, binary, and Poisson bipartite panels, including named changing-composition panels. The marginal $U_t G_t V_t'$ linear predictor is the identified object; individual G_t entries can shift under equivalent rotations and scalings of U_t and V_t. dynamic_G = TRUE is bipartite only.

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

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 in this model.

"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.

"poisson": An overdispersed Poisson AME model for count data: $Y \sim \text{Poisson}(\exp(z))$ with $z \sim N(\eta, \sigma^2)$, a lognormal-mixed Poisson. The conditional mean given the latent z is $\exp(z)$; the marginal mean is $\exp(\eta + \sigma^2/2)$, not $\exp(\eta)$.

Value

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

APM — posterior mean of additive row effects a

BPM — posterior mean of additive column effects b

U	posterior mean of multiplicative row effects u. For dynamic_uv=TRUE, this is a 3D array (n x R x T)
V	posterior mean of multiplicative column effects v (asymmetric case). For dynamic_uv=TRUE, this is a 3D array (n x R x T)
UVPM	posterior mean of UV
ULUPM	posterior mean of ULU (symmetric case)
L	posterior mean of L (symmetric case)
EZ	estimate of expectation of Z matrix. For `mode = "bipartite"`, `EZ` is a list of per-period matrices whose row/column dimnames are currently `NULL`. Use `names(fit$APM)` / `names(fit$BPM)` (or `dimnames(fit$YPM[[t]])`) to recover the row/column actor ordering, which is the lexicographic sort of the input actor names (so `"r10"` sorts before `"r2"` unless you zero-pad).
YPM	posterior mean of Y (for imputing missing values)
GOF	observed (first row) and posterior predictive (remaining rows) values of four goodness-of-fit statistics. See `gof` for post-hoc computation and `gof_plot` for visualization.
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, `lame_snap_als` for the approximate dynamic snap-shift 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)
fit<-lame(YX_bin_list$Y,YX_bin_list$X,burn=5,nscan=5,odens=1,family="binary")
# 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(YX_bin_list$Y, YX_bin_list$X,
                family = "binary", R = 0,
                nscan = 60, burn = 15, odens = 5,
```

```
                dynamic_beta = "dyad")
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. 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"),
  bootstrap = 0L,
  bootstrap_type = c("parametric", "block"),
  bootstrap_block_length = 1L,
  bootstrap_seed = NULL,
  verbose = TRUE,
  seed = 6886
)
```

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.
max_iter	maximum number of block coordinate descent iterations (default 200).
tol	convergence tolerance on the relative change in residual sum of squares (default 1e-6).
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, "irls" (default for binary/poisson) runs iteratively reweighted least squares, giving a fast approximate 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 lowest-SSE fit, warning when the starts reach materially different optima. Reproducible given `seed`; the global RNG stream is left unchanged.

`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. 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`.

`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`). The block coordinate descent is deterministic, so the point estimate is reproducible regardless; the argument is retained for API consistency with `ame`.

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 adapted here originates with that work (implemented in `sir::sir_alsfit()`).

See Also

`ame_als`, `ame_als_bootstrap`, `lame` for the full MCMC estimator, `lame_snap_als` for the approximate dynamic snap-shift point 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)
fit_multi <- lame_multi(
  Y_list = list(YX_bin_list$Y, YX_bin_list$Y),
  Xdyad_list = list(YX_bin_list$X, YX_bin_list$X),
  family = "binary", R = 0,
  nscan = 100, burn = 25, odens = 5, verbose = FALSE)
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::rhat(as_draws(fit)) 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(YX_bin_list$Y, Xdyad = YX_bin_list$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 implementation is pragmatic: it loads the saved RNG state and re-invokes lame() with the original call arguments. The result is a fresh fit that picks up where the previous one left off in the random-number stream; the underlying MCMC counter restarts at 1.

Usage

```
lame_resume(path, nscan_more = NULL, ..., .envir = NULL)
```

Arguments

path	Checkpoint file path (the one passed as checkpoint_path to lame()).
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 stored draws. Both call shapes are supported. The `lame_resume(path, ...)` form is safer for nested calls: the consolidated form re-evaluates the saved `lame()` call in `parent.frame()`, which is the `lame()` frame whose required formal arguments (`Y`, `Xdyad`, etc.) were not supplied on the resume call. Prefer `lame_resume(path, ...)` when calling from inside other functions or from non-global scopes.

Use `nscan_more = K` to override the `nscan` value to `K` for the continuation. Other arguments can be overridden by passing them to `...`.

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(YX_bin_list$Y, YX_bin_list$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$terminated_early)) {
  fit2 <- lame_resume(ck, nscan_more = 200)
  dim(fit2$BETA)
}
```

---

lame_snap_als *Fast approximate dynamic snap-shift AME estimator*

Description

Fits a fast point-estimator approximation to the dynamic snap-shift latent-factor model for longitudinal normal-valued networks. It targets the same drift-versus-reset transition estimand as `lame` with `dynamic_uv = TRUE` and `dynamic_uv_kind = "snap"`, but returns ALS snap scores rather than MCMC draws.

Usage

```
lame_snap_als(
  Y,
  Xdyad = NULL,
  Xrow = NULL,
  Xcol = NULL,
```

```
  R = 2L,
  R_row = NULL,
  R_col = NULL,
  family = "normal",
  mode = c("unipartite", "bipartite"),
  symmetric = FALSE,
  max_iter = 200L,
  tol = 1e-06,
  snap_kappa = 2,
  snap_pi_prior = c(a = 1, b = 9),
  snap_update = c("soft", "hard", "annealed"),
  snap_damping = 0.7,
  estimate_rho_uv = TRUE,
  estimate_sigma_uv = TRUE,
  rho_uv = NULL,
  sigma_uv = NULL,
  hyper_update = c("robust", "em"),
  drift_quantile = 0.05,
  drift_min_transitions = 50,
  rho_prior_mean = 0.98,
  rho_prior_weight = 25,
  align = c("sequential", "global", "none"),
  threshold = 0.5,
  min_sigma = 1e-04,
  sigma_floor_fraction = 0.75,
  ridge = 1e-08,
  snap_stability_tol = 0.05,
  snap_convergence = c("quantile", "max", "classification"),
  snap_delta_quantile = 0.95,
  snap_class_change_tol = 0.005,
  unstable_top_n = 10L,
  stability = c("none", "quick", "validation"),
  verbose = TRUE,
  seed = 6886
)
```

Arguments

`Y`	a list of relational matrices, or a 3D array. Unipartite fits use `[n, n, T]` panels. Bipartite fits use rectangular `[n_row, n_col, T]` panels. For named lists, slices may have changing actor composition; the estimator pads to the union actor set and treats absent actor-periods as unobserved and snap-ineligible. A longitudinal **netify** object is also accepted and converted with `netify::to_lame(lame = TRUE)`.
`Xdyad`	optional list of dyadic covariate matrices/arrays, or `NULL`.
`Xrow, Xcol`	not supported by this fast snap-shift estimator; pass `NULL`. Use `lame` with `method = "mcmc"` for node-covariate dynamic snap models.

R	positive integer latent rank. For bipartite fits, used as the default for R_row and R_col.
R_row, R_col	positive integer latent ranks for bipartite row and column positions. Defaults to R.
family	currently only "normal".
mode	"unipartite" or "bipartite".
symmetric	logical; if TRUE, fit a symmetric latent-factor approximation and report one snap-probability matrix.
max_iter	maximum block-coordinate iterations.
tol	convergence tolerance on the relative change of the tracked $-2\times$ variational free energy (see Details).
snap_kappa	diffuse snap-prior standard deviation.
snap_pi_prior	length-two vector c(a, b) for the beta prior used to regularize the snap rate.
snap_update	one of "soft", "hard", or "annealed".
snap_damping	scalar in (0, 1]; damping applied to soft snap-score updates. Values below 1 mix the new score with the previous score to reduce oscillation on large panels. Ignored for snap_update = "hard".
estimate_rho_uv, estimate_sigma_uv	logical flags for updating the AR drift persistence and innovation scale.
rho_uv, sigma_uv	optional fixed/initial AR drift parameters.
hyper_update	one of "robust" or "em". The default "robust" estimates drift hyperparameters from the lower tail of transition innovations, which prevents broad ruptures from being absorbed into an overly diffuse drift process. "em" uses the untrimmed soft-classification moment update.
drift_quantile	lower-tail transition quantile used by hyper_update = "robust" for estimating smooth-drift hyperparameters.
drift_min_transitions	minimum effective number of transitions retained by the lower-tail update. This keeps the default from overfitting the smooth-drift scale on small panels.
rho_prior_mean, rho_prior_weight	weak regularization for the persistence estimate. The defaults encode the snap-shift model's intended persistent, low-innovation drift baseline.
align	initialization alignment mode; "global" aligns each period to the pooled static ALS fit, "sequential" aligns each period to the previous initialized period, and "none" leaves per-period factors unaligned.
threshold	hard-classification threshold for snap_class.
min_sigma	lower bound for the drift scale.
sigma_floor_fraction	for robust hyperparameter updates, the iterative drift scale cannot fall below this fraction of the initial sigma_uv. This prevents broad snap assignments from collapsing the smooth-drift variance to min_sigma.
ridge	small ridge added to latent-position normal equations.

snap_stability_tol
: tolerance for declaring soft snap scores stable. This is separate from tol, which tracks the penalized objective.

snap_convergence
: convergence criterion for snap scores. "quantile" uses snap_delta_quantile of the absolute score changes plus the class-change share; "max" uses the worst actor-period score change; "classification" uses only class-change stability. The worst-case max is always stored and warned about when high.

snap_delta_quantile
: quantile of actor-period score changes used by snap_convergence = "quantile".

snap_class_change_tol
: maximum share of eligible actor-periods whose snap_class may change between iterations while still declaring snap convergence.

unstable_top_n
: number of largest actor-period snap-score changes to store in convergence diagnostics.

stability
: optional start-sensitivity preset. "none" runs one fit. "quick" and "validation" rerun the same estimator from additional seeds and attach fit$stability with snap-score, classification, top-period, and fitted-surface comparisons.

verbose
: logical; print progress.

seed
: integer seed for initialization perturbations.

Details

The raw snap-ALS intercept mu is *not* separately identified from the uncentered multiplicative term: a constant level can be absorbed into $\mathrm{mean}(U_t V_t')$ (bipartite: $\mathrm{mean}(U_t G V_t')$), so fit$mu on its own is arbitrary. After convergence the estimator therefore computes a post-hoc identified level: fit$mu_t stores the per-period level $\mu + \mathrm{mean}(U_t V_t')$ (named by period) and fit$mu_identified is the mean of fit$mu_t. Interpret the network level through mu_identified; the latent trajectories and the covariate coefficients are unaffected by this shift and remain reliable outputs of this estimator.

On longer panels, the default align = "sequential" keeps the period-to-period movement in view. align = "global" can help on very short panels, but on longer panels it can pull each period back toward the pooled static fit and make early jumps look too strong. If you have a fixed drift scale, pass rho_uv and sigma_uv with estimate_rho_uv = FALSE and estimate_sigma_uv = FALSE.

The convergence output separates the usual score summary from the worst moving actor-period. By default, convergence uses the 95th percentile of score changes and the share of class changes. final_max_snap_delta and unstable_transitions still show the largest local moves. Read snap_prob as an ALS snap score. It is useful for rankings and heuristic classifications; it is not a Bayesian posterior probability.

For unipartite fits, the tracked objective (fit$objective_trace, fit$convergence$final_objective) is the exact $-2\times$ variational free energy of the snap/drift mixture model, including the Gaussian log-normalizers, the responsibility entropy, and the beta prior on the snap rate. The exact block updates descend it monotonically; small increases can come from damped snap scores or the robust hyperparameter updates. Because it carries log-variance normalizers it can be negative and is not a deviance; fit$param_trace$sse tracks raw data fit directly.

In bipartite mode, the fitted multiplicative term is $U_t G V_t'$ with one static interaction matrix G. The returned `snap_prob` matrix contains row-actor snap scores and `snap_prob_v` contains column-actor snap scores. Bipartite snap ALS does not estimate node covariates, dynamic coefficients, or a dynamic G_t; those combinations need a separate model.

Value

An object of class `"lame_snap_als"` with dynamic latent positions, snap scores, fitted values, residuals, and convergence diagnostics.

Examples

```
set.seed(1)
Y_bip <- lapply(seq_len(3), function(t) {
  m <- matrix(rnorm(6 * 5), 6, 5)
  rownames(m) <- paste0("r", seq_len(6))
  colnames(m) <- paste0("c", seq_len(5))
  m
})
fit_bip <- lame_snap_als(Y_bip, R = 1, mode = "bipartite",
                         max_iter = 3, verbose = FALSE)
dim(fit_bip$snap_prob_v)
```

---

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. Optionally applies Procrustes alignment for dynamic models and includes posterior standard deviations when posterior samples 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 models (dynamic_uv = TRUE), apply Procrustes alignment across time to remove rotational indeterminacy. Default is FALSE for ame objects and TRUE for lame objects.

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 samples are not available. To enable, fit the model with posterior_opts = posterior_options(save_UV = TRUE).

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)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
           burn = 5, nscan = 5, odens = 1, verbose = FALSE)
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`, uses the original posterior means (a much rougher approximation).
`...`	Passed to the refit `lame()` call (typically `nscan`, `burn`, `odens`, `verbose`).

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` (effective number of parameters), `per_period` (data frame with `period`, `elpd`, `n_obs`), and `periods` (the periods evaluated).

Examples

```
data(YX_bin_list)
fit <- lame(YX_bin_list$Y, YX_bin_list$X, family = "binary", R = 0,
            dynamic_beta = "dyad",
            nscan = 60, burn = 15, odens = 5, verbose = FALSE)
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

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.

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.

---

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

What log_lik **measures.** For family in {normal, binary, cbin, poisson, ordinal} the stored pointwise log-likelihood is the exact family-specific Y density on the response scale, so elpd_loo is directly comparable to a loo() output from Stan / brms fit to the same family. For the rank likelihood frn the exact marginal needs GHK Monte Carlo (Halton sequence); on the longitudinal lame() path you can opt in with log_lik_method = "observed_ghk", on the cross-sectional ame() path the fallback is the augmented-Z normal approximation (with a one-time warning). Inspect fit$log_lik_method on any fit to see which branch was used.

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)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 0,
           nscan = 60, burn = 15, odens = 5,
           save_log_lik = TRUE, verbose = FALSE)
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 definite matrix

Usage

```
mhalf(M)
```

Arguments

M	a positive definite matrix

Value

a matrix H such that H^2 equals M

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

nobs.ame *Number of observed dyads in an AME / LAME fit*

Description

Number of 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 residual after the main MCMC linear predictor; 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)
fit <- lame(YX_bin_list$Y, YX_bin_list$X, family = "binary", R = 0,
            nscan = 100, burn = 25, odens = 5, verbose = FALSE)
# 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)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2, gof = TRUE,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE)

# 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
fit <- lame(Y_list, family = "normal",
            nscan = 50, burn = 10, odens = 1, verbose = FALSE, plot = FALSE)

# 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`: `ame(Y, ..., posterior_opts = posterior_options(save_UV = TRUE))`

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

`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"
probs	Numeric vector of probabilities for quantiles

Value

Matrix or array of posterior quantiles

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

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.
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 parameter uncertainty (default TRUE)
...	Additional arguments (not used)

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)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE)

# 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"), ...)
```

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".
...	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.

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"),
  probs = c(0.025, 0.975),
  newexposure = NULL,
  n_draws = NULL,
  seed = NULL,
  ...
)
```

Arguments

object	Fitted LAME model object.
newdata	Optional list of T dyadic covariate arrays ([n_row, n_col, p] each, with the same actors as the fit) to compute counterfactual predictions. When NULL, the training-data predictions are returned.
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.
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) or "credible". When h > 0 and "credible", the per-period output is a list of length-3 lists with $lower, $median, $upper matrices computed at the probs quantiles across posterior draws. Ignored for in-sample (h = 0) predictions.
probs	Length-2 vector of lower / upper quantiles for the credible interval when interval = "credible". 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).

Value

List of prediction matrices (one per time point).

---

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".
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.

---

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)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE)

# 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`.

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 the priors that were actually in effect for a fit, with defaults filled in. 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.

`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$U.
V	Optional 3D array [n, R, T] of receiver latent positions. For unipartite asymmetric models, aligned independently. For bipartite models, aligned jointly with U via the G interaction matrix.
G	Optional [R_row, R_col] interaction matrix for bipartite models, transformed to preserve the invariant U G V'.
return_fit	Logical. If TRUE and object is provided, returns a modified copy of the fit object with aligned latent positions. Default FALSE.
per_draw	Logical. When TRUE, run Procrustes alignment per posterior draw rather than on the posterior-mean trajectory. This uses object$U_full / object$V_full (the per-draw posterior cubes – populated when posterior_opts includes storing U/V) when present; otherwise falls back to mean-trajectory alignment and emits an informational note. 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 unipartite networks, U and V are aligned independently using separate Procrustes rotations. For symmetric networks, only U is present and aligned.

For bipartite networks, U and V are aligned jointly: separate rotation matrices are computed for U and V, and the G interaction matrix is updated as G_aligned = t(R_U) %*% G %*% R_V to preserve the product U %*% G %*% t(V).

If U is a 2D matrix (static model with a single time point), it is returned unchanged with an informational message.

Value

If return_fit = FALSE (default): a list with components U (aligned sender positions), V (aligned receiver positions, if applicable), and G (updated 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))
fit <- lame(Y_bin, Xdyad = YX_bin_list$X, R = 2,
            family = "binary", dynamic_uv = TRUE,
            burn = 5, nscan = 5, odens = 1,
            verbose = FALSE)
aligned <- procrustes_align(fit)
str(aligned$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
)
```

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

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.

`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

---

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 implied by the family (e.g., sqrt(p*(1-p)) for binary).

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"), ...)
```

Arguments

object	an ame_als object.
type	"response" (default) or "working".
...	ignored.

Details

type = "response" (default) returns observed Y minus the response-scale fitted values, so residuals() reconciles with fitted(). 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

Extract residuals from LAME model

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 split-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(..., chains = K), 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 per-(k,t) split-R-hat values for comparison.

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
fit_list <- lapply(c(1L, 2L), function(s) {
  lame(YX_bin_list$Y, YX_bin_list$X, family = "binary", R = 0,
       dynamic_beta = "dyad", seed = s,
       nscan = 60, burn = 15, odens = 5, verbose = FALSE)
})
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

Usage

```
rmvnorm(n, mu, Sigma, Sigma.chol = NULL)
```

Arguments

`n`	sample size
`mu`	multivariate mean vector
`Sigma`	covariance matrix
`Sigma.chol`	Cholesky factorization of `Sigma`

Value

a matrix with n rows

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

---

rSab_fc *Gibbs update for additive effects covariance*

Description

Gibbs sampling for the covariance matrix of additive row and column effects in the AME model. This function implements the inverse-Wishart posterior update for the covariance matrix Sab.

Usage

```
rSab_fc(a, b, Sab0=NULL, eta0=NULL, rvar=TRUE, cvar=TRUE, symmetric=FALSE)
```

Arguments

a	vector of row random effects (additive sender effects)
b	vector of column random effects (additive receiver effects)
Sab0	prior scale matrix. **Convention (Hoff/amen):** Sab0 is passed *raw*; the inverse-Wishart scale used internally is $\eta_0 \cdot Sab0$, so the prior mean of Sab is $\eta_0 Sab0/(\eta_0 - 3) \approx Sab0$ and the prior mode is $\eta_0 Sab0/(\eta_0 + 3)$. Every call site in the package (ame() unipartite and bipartite, lame() unipartite and bipartite, and the raSab_bin/cbin/frn joint updates) passes Sab0 raw under this same convention, so a given prior$Sab0 means the same prior everywhere. Default is diag(2) (weakly informative).
eta0	prior degrees of freedom for the prior distribution. Default is 4, which is the minimum for a proper prior with 2x2 matrix.
rvar	logical: should row variance be updated? (default TRUE)
cvar	logical: should column variance be updated? (default TRUE)
symmetric	logical: is this a symmetric network? (default FALSE)

Details

The function implements different update strategies:

- Full update: When both rvar and cvar are TRUE, updates the full 2x2 covariance matrix using an inverse-Wishart distribution
- Row variance only: When only rvar is TRUE, updates only Sab[1,1]
- Column variance only: When only cvar is TRUE, updates only Sab[2,2]
- Symmetric case: When symmetric is TRUE, draws a single shared variance for the row and column effects (inverse-gamma update) with zero covariance

Value

Updated covariance matrix Sab (2x2 matrix with variances on diagonal and covariance off-diagonal)

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

`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_cube, V_cube, E, G, rho_uv, sigma_uv, s2)
```

Arguments

`U_cube`	3D array (nA x RA x T)
`V_cube`	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

Value

List with updated U_cube, V_cube

rUV_dynamic_fc *Gibbs sampling of dynamic U and V with AR(1) evolution*

Description

Updates latent factor positions U and V that evolve over time according to an AR(1) process: u_{i,t} = rho * u_{i,t-1} + epsilon_{i,t}

Usage

```
rUV_dynamic_fc(U, V, ET, rho_uv, sigma_uv, s2, shrink=TRUE, symmetric=FALSE)
```

Arguments

U	3D array of current U positions (n x R x T)
V	3D array of current 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 (default TRUE)
symmetric	whether the network is symmetric (default FALSE)

Value

list with updated U and V arrays

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

rUV_dynamic_fc_cpp *Update dynamic latent positions using AR(1) process*

Description

Update dynamic latent positions using AR(1) process

Usage

```
rUV_dynamic_fc_cpp(
  U_current,
  V_current,
  ET,
  rho_uv,
  sigma_uv,
  s2,
  shrink,
  symmetric
)
```

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

Value

List with updated U and V arrays

---

rUV_dynamic_snap_fc *Gibbs sampling of dynamic U and V with snap-shift dynamics*

Description

Like rUV_dynamic_fc but lets each actor's latent position either drift under the AR(1) prior or jump under a diffuse N(0, kappa^2) snap prior, chosen per actor-period by a Gaussian log-marginal model selection.

Usage

```
rUV_dynamic_snap_fc(
  U,
  V,
  ET,
  rho_uv,
  sigma_uv,
  s2,
```

```
  kappa,
  pi_snap,
  delta_u_current = NULL,
  delta_v_current = NULL,
  shrink = TRUE,
  symmetric = FALSE
)
```

Arguments

U	3D array of current U positions (n x R x T)
V	3D array of current 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
pi_snap	prior snap probability
delta_u_current	current sender-side snap indicators. If NULL, a zero matrix is used.
delta_v_current	current receiver-side snap indicators. If NULL, a zero matrix is used.
shrink	whether to apply shrinkage (default TRUE)
symmetric	whether the network is symmetric (default FALSE)

Value

list with updated U, V arrays and delta_u, delta_v snap indicators

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

---

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
)
```

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

Value

List with updated U, V arrays and delta_u, delta_v snap indicators

`rUV_dynamic_t_fc` *Gibbs sampling of dynamic U and V with heavy-tailed (Student-t) innovations*

Description

Like `rUV_dynamic_fc` but the AR(1) innovations are Student-t via a scale-mixture of normals, a continuous heavy-tailed alternative to snap-shift.

Usage

```
rUV_dynamic_t_fc(
  U,
  V,
  ET,
  rho_uv,
  sigma_uv,
  s2,
  nu,
  lambda_u = NULL,
  lambda_v = NULL,
  shrink = TRUE,
  symmetric = FALSE
)
```

Arguments

`U`	3D array of current U positions (n x R x T)
`V`	3D array of current 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 (default TRUE)
`symmetric`	whether the network is symmetric (default FALSE)

Value

list with updated U, V arrays and lambda_u, lambda_v local scales

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

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
)
```

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

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)
```

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

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
```

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)

rZ_nrm_fc *Simulate missing values in a normal AME model*

Description

Simulates missing values of a sociomatrix under a normal AME model

Usage

```
rZ_nrm_fc(Z, EZ, rho,s2, Y)
```

Arguments

Z	a square matrix, the current value of Z
EZ	expected value of Z
rho	dyadic correlation
s2	dyadic variance
Y	square relational matrix

Value

a square matrix, equal to Y at non-missing values

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

rZ_pois_fc *Gibbs update for latent variable in a Poisson AME model*

Description

Updates the latent variable Z in a Poisson AME model using a Metropolis-Hastings step. The model assumes y_{i,j} ~ Poisson(exp(z_{i,j})) where z_{i,j} is the latent variable representing the log mean.

Usage

```
rZ_pois_fc(Z, EZ, rho, s2, Y)
```

Arguments

Z	a square matrix, the current value of the latent variable
EZ	expected value of Z (regression effects + random effects)
rho	dyadic correlation
s2	dyadic variance (overdispersion parameter)
Y	square relational matrix of observed counts

Value

updated value of Z

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

---

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
)
```

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).

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, e_ij ~ N(0, 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, and for symmetric networks each dyad contributes exactly once (resid(i,j) = a_i + a_j + e_ij).

Usage

```
sample_dynamic_ab_cpp(
  a_current,
  b_current,
  Z_array,
  EZ_array,
  rho_ab,
  sigma_ab,
  s2,
  symmetric
)
```

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

Value

List with updated a and b arrays

---

sample_rho_ab_cpp *Sample AR(1) parameter for dynamic additive effects*

Description

Sample AR(1) parameter for dynamic additive effects

Usage

```
sample_rho_ab_cpp(
  a_mat,
  b_mat,
  sigma_ab,
  rho_current,
  symmetric,
  prior_mean = 0,
  prior_sd = -1
)
```

Arguments

a_mat	Matrix of row effects (n x T)
b_mat	Matrix of column effects (n x T)
sigma_ab	Innovation standard deviation
rho_current	Current value of rho
symmetric	Whether the network is symmetric
prior_mean	Prior mean for rho. Used only when prior_sd > 0.
prior_sd	Prior SD for rho. prior_sd < 0 (the default) selects a Jeffreys-like flat prior; a positive value switches to a truncated Normal(prior_mean, prior_sd^2) prior.

Value

Updated rho value

---

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_rho_uv` *Sample AR(1) parameter for dynamic latent factors*

Description

Independence Metropolis-Hastings update coherent with the stationary AR(1) initial condition used by the dynamic UV kernels. The proposal is the conjugate Normal implied by the transitions and the Normal(prior_mean, prior_sd^2) prior, drawn truncated to (-0.99, 0.99) via the inverse CDF; the acceptance ratio supplies the t = 1 stationary factor proportional to `(1 - rho^2)^(nR / 2) * exp(-(1 - rho^2) * S1 / (2 * sigma^2))` that the Gaussian proposal omits. Defaults (prior_mean = 0, prior_sd = 1) preserve the historical behaviour; lame::lame() passes the user-set prior$rho_uv_mean / prior$rho_uv_sd explicitly.

Usage

```
sample_rho_uv(
  U_cube,
  V_cube,
  sigma_uv,
  rho_current,
  symmetric,
  prior_mean = 0,
  prior_sd = 1
)
```

Arguments

U_cube	3D array of U positions (n x R x T)
V_cube	3D array of V positions (n x R x T)
sigma_uv	Innovation standard deviation
rho_current	Current value of rho
symmetric	Whether network is symmetric
prior_mean	Prior mean for rho (default 0)
prior_sd	Prior SD for rho (default 1)

Value

Updated rho value

`sample_sigma_ab_cpp` *Sample innovation variance for dynamic additive effects*

Description

Sample innovation variance for dynamic additive effects

Usage

```
sample_sigma_ab_cpp(
  a_mat,
  b_mat,
  rho_ab,
  symmetric,
  prior_shape = 2,
  prior_scale = 1
)
```

Arguments

`a_mat`	Matrix of row effects (n x T)
`b_mat`	Matrix of column effects (n x T)
`rho_ab`	AR(1) parameter
`symmetric`	Whether the network is symmetric
`prior_shape`	Shape parameter for inverse gamma prior
`prior_scale`	Scale parameter for inverse gamma prior

Value

Updated sigma_ab value

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.

sample_sigma_uv *Sample innovation variance for dynamic latent factors*

Description

Sample innovation variance for dynamic latent factors

Usage

```
sample_sigma_uv(U_cube, V_cube, rho_uv, symmetric)
```

Arguments

U_cube	3D array of U positions (n x R x T)
V_cube	3D array of V positions (n x R x T)
rho_uv	AR(1) parameter
symmetric	Whether network is symmetric

Value

Updated sigma_uv value

---

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.

sampsonmonks *Sampson's monastery data*

Description

Several dyadic variables measured on 18 members of a monastery.

Format

A socioarray whose dimensions represent nominators, nominatees 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_m2` and `like_m1` are evaluations of likeing at one and two timepoints previous to when the other relations were measured.

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 multiple network realizations from a fitted AME model. This function performs conditional posterior predictive simulation: it draws from stored MCMC samples when they are available and uses posterior means for latent components that were not retained.

Unlike `print.ame`, which only displays fitted quantities, `simulate.ame` draws new networks from the posterior predictive distribution.

Usage

```
## S3 method for class 'ame'
simulate(
  object,
  nsim = 100,
  seed = NULL,
  newdata = NULL,
  burn_in = 0,
  thin = 1,
  return_latent = FALSE,
  ...
)
```

Arguments

`object`	fitted model object of class "ame"
`nsim`	number of networks to simulate (default: 100)
`seed`	random seed for reproducibility
`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)
`...`	additional arguments (not currently used)

Details

Model:

The AME model represents networks through a latent variable framework:

$$Y_{ij} \sim F(Z_{ij})$$

where F is the observation model (e.g., probit for binary) and Z is the latent network:

$$Z_{ij} = \beta^T x_{ij} + a_i + b_j + u_i^T v_j + \epsilon_{ij}$$

Components:

- β: regression coefficients for dyadic/nodal covariates
- a_i, b_j: additive sender and receiver random effects
- u_i, v_j: multiplicative latent factors (dimension R)
- ϵ_{ij}: dyadic random effects with correlation ρ

Simulation:

For each simulated network k = 1, ..., nsim:

1. **Parameter draw:** Draw parameter set $\theta^{(k)}$ from MCMC chains:
   - Sample iteration s uniformly from stored MCMC samples
   - Extract $\beta^{(s)}$, variance components $(v_a^{(s)}, v_b^{(s)}, v_e^{(s)}, \rho^{(s)})$
2. **Random effects:** Sample new random effects from posterior distributions:
   - $a_i^{(k)} \sim N(0, v_a^{(s)})$ for i = 1, ..., n (row effects)
   - $b_j^{(k)} \sim N(0, v_b^{(s)})$ for j = 1, ..., m (column effects)
   - Fresh draws from the posterior variance carry random-effect uncertainty into the simulated networks
3. **Latent network:** Build expected latent positions:

   $$E[Z_{ij}^{(k)}] = \beta^{(s)T} x_{ij} + a_i^{(k)} + b_j^{(k)} + \hat{u}_i^T \hat{v}_j$$

   where $\hat{u}_i, \hat{v}_j$ are posterior mean latent factors
4. **Dyadic correlation:** Add correlated noise structure:

   $$Z_{ij}^{(k)} = E[Z_{ij}^{(k)}] + \epsilon_{ij}^{(k)}$$

   where ϵ has covariance structure:

   $$Cov(\epsilon_{ij}, \epsilon_{ji}) = \rho^{(s)} v_e^{(s)}$$

   $$Var(\epsilon_{ij}) = v_e^{(s)}$$
5. **Observation model:** Generate the observed network:
   - Binary: $Y_{ij}^{(k)} = I(Z_{ij}^{(k)} > 0)$
   - Normal: $Y_{ij}^{(k)} = Z_{ij}^{(k)}$

- Poisson: $Y_{ij}^{(k)} \sim Poisson(\exp(Z_{ij}^{(k)}))$
- Other families use appropriate link functions

Sources of uncertainty:

The simulation captures three types of uncertainty:

1. **Parameter uncertainty:** Different MCMC samples yield different $\beta, v_a, v_b, v_e, \rho$
2. **Random effect uncertainty:** Fresh draws from $N(0, v_a), N(0, v_b)$ for each simulation
3. **Dyadic uncertainty:** Correlated random noise ϵ_{ij}

The resulting simulations propagate uncertainty from the stored parameter draws and from fresh dyadic/random-effect draws. Latent components that were not stored as MCMC draws are held at their posterior means.

Latent-factor draws:

Multiplicative effects (U, V) use posterior means unless the fit retained compatible latent-factor draws. Storing full latent-factor chains can require substantial additional memory.

Symmetric Networks:

For symmetric networks, the model enforces $a_i = b_i$ and $u_i = v_i$, and the latent matrix Z is symmetrized before generating observations.

Value

A list with components:

Y list of nsim simulated networks in the same format as the original data

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)
fit <- ame(YX_bin$Y, Xdyad = YX_bin$X, burn = 10, nscan = 100, odens = 1,
           family = "binary", verbose = FALSE)

# 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 and binary fits, reciprocal dyads use the fitted residual rho. 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 multiple longitudinal network realizations from a fitted LAME (Longitudinal AME) model. This function performs conditional posterior predictive simulation for dynamic networks: it draws from stored MCMC samples when available and uses posterior means for latent components that were not retained.

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",
  ...
)
```

Arguments

object	fitted model object of class "lame"
nsim	number of network trajectories to simulate (default: 100)
seed	random seed for reproducibility
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"** start from posterior mean (default) **"random"** random initialization **"data"** use first time point from original data
...	additional arguments (not currently used)

Details

Mathematical Framework for Longitudinal Networks:

The LAME model extends AME to multiple time periods T with potential temporal dependencies. For each time t = 1, ..., T:

$$Y_{ij,t} \sim F(Z_{ij,t})$$

where the latent network evolves as:

$$Z_{ij,t} = \beta^T x_{ij,t} + a_{i,t} + b_{j,t} + u_{i,t}^T v_{j,t} + \epsilon_{ij,t}$$

Temporal Dynamics:

LAME can incorporate three types of temporal dependencies:

1. **Static Effects:** Parameters constant over time
   - $a_{i,t} = a_i$, $b_{j,t} = b_j$ for all t
   - $u_{i,t} = u_i$, $v_{j,t} = v_j$ for all t
2. **Dynamic Additive Effects:** AR(1) process for random effects

$$a_{i,t} = \rho_{ab} a_{i,t-1} + \eta_{i,t}, \quad \eta_{i,t} \sim N(0, \sigma_a^2(1-\rho_{ab}^2))$$

$$b_{j,t} = \rho_{ab} b_{j,t-1} + \xi_{j,t}, \quad \xi_{j,t} \sim N(0, \sigma_b^2(1-\rho_{ab}^2))$$

   where ρ_{ab} is the temporal correlation parameter
3. **Dynamic Multiplicative Effects:** AR(1) for latent factors

$$u_{i,t} = \rho_{uv} u_{i,t-1} + \omega_{i,t}$$

$$v_{j,t} = \rho_{uv} v_{j,t-1} + \psi_{j,t}$$

Uncertainty Quantification Process for Trajectories:

For each simulated trajectory k = 1, ..., nsim:

Step 1: Parameter Sampling

- Draw MCMC iteration s uniformly from stored posterior samples
- Extract static parameters: $\beta^{(s)}$, variance components
- Extract temporal parameters if applicable: $\rho_{ab}^{(s)}$, $\rho_{uv}^{(s)}$

Step 2: Initialize at t = 1

Depending on start_from parameter:

- "posterior": Use posterior means as starting values
- "random": Draw from stationary distribution
- For additive effects: $a_{i,1}^{(k)} \sim N(0, \sigma_a^2)$
- For multiplicative effects: Initialize from prior

Step 3: Evolve Through Time

For each t = 2, ..., T:

a) **Update Dynamic Effects** (if applicable):

$$a_{i,t}^{(k)} = \rho_{ab}^{(s)} a_{i,t-1}^{(k)} + \eta_{i,t}^{(k)}$$

where $\eta_{i,t}^{(k)} \sim N(0, \sigma_a^2(1 - [\rho_{ab}^{(s)}]^2))$

The innovation variance $\sigma_a^2(1-\rho_{ab}^2)$ ensures stationarity

b) **Construct Latent Network**:

$$E[Z_{ij,t}^{(k)}] = \beta^{(s)T} x_{ij,t} + a_{i,t}^{(k)} + b_{j,t}^{(k)} + u_{i,t}^T v_{j,t}$$

c) **Add Dyadic Noise**:

$$Z_{ij,t}^{(k)} = E[Z_{ij,t}^{(k)}] + \epsilon_{ij,t}^{(k)}$$

with correlation structure preserved from AME model

d) **Generate Observations**: Apply appropriate link function based on family

Sources of Uncertainty in Longitudinal Context:

1. **Cross-sectional uncertainty** (as in AME):
   - Parameter uncertainty from MCMC
   - Random effect variability
   - Dyadic noise
2. **Temporal uncertainty**:
   - Uncertainty in temporal correlation parameters ρ_{ab}, ρ_{uv}
   - Innovation noise in AR(1) processes
   - Propagation of uncertainty through time (compounds over periods)
3. **Initial condition uncertainty**:
   - Different starting values lead to different trajectories
   - Captured through start_from options

Interpretation of Multiple Trajectories:

Each simulated trajectory represents one possible evolution of the network conditional on the stored fit. Variation across trajectories captures:

- Model parameter uncertainty
- Stochastic variation in temporal evolution
- Accumulated uncertainty over time periods

The ensemble of trajectories provides prediction intervals that widen over time, reflecting increasing uncertainty in longer-term forecasts.

Special Considerations:

1. **Temporal Correlation:** Higher ρ values create smoother trajectories with more persistence
2. **Stationarity:** The AR(1) innovation variance is scaled to maintain stationary marginal distributions
3. **Missing Time Points:** If simulating beyond observed data (n_time > T_observed), covariates are recycled or set to zero with appropriate warnings

Limitations:

As with simulate.ame, multiplicative effects use posterior means unless the fit retained compatible latent-factor draws. Storing complete MCMC chains for $u_{i,t}, v_{j,t}$ at all time points is memory-intensive for large networks and long time series.

Value

A list with components:

Y list of nsim simulated longitudinal network trajectories, each element is a list of T networks

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
fit <- lame(Y_list, family = "normal",
            nscan = 50, burn = 10, odens = 1, verbose = FALSE, plot = FALSE)

# 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.

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)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE)

# Get posterior samples of regression coefficients
beta_post <- simulate_posterior(fit, "beta", n_samples = 50)
```

---

simY_pois *Simulate a Poisson relational matrix*

Description

Simulates a relational matrix from a Poisson distribution given the log mean values

Usage

```
simY_pois(EZ)
```

Arguments

EZ	square matrix giving the log expected values of the relational matrix

Value

a square matrix of counts

Author(s)

Cassy Dorff, Shahryar Minhas, Tosin Salau

---

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, or a lame_snap_als fit. MCMC fits should be run with retained snap draws for posterior uncertainty; ALS fits contribute their snap_prob scores as one score row.
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.

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, or a `lame_snap_als` fit. MCMC fits should be run with retained snap draws for posterior uncertainty; ALS fits contribute their `snap_prob` scores as one score row.
`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`.

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, or a lame_snap_als fit. MCMC fits should be run with retained snap draws for posterior uncertainty; ALS fits contribute their snap_prob scores as one score row.
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.

---

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, or a lame_snap_als fit. MCMC fits should be run with retained snap draws for posterior uncertainty; ALS fits contribute their snap_prob scores as one score row.
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.

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, approximate p-values, and 95% credible intervals for dyadic, sender, and receiver covariates. Note: the z-values are computed as posterior mean / posterior SD, and the p-values are derived from a normal approximation. These are convenient screening statistics but are not formal frequentist test statistics. For rigorous inference, use the credible intervals or examine the full posterior via the BETA matrix directly.

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.

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

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, approximate p-values, and 95% credible intervals for dyadic, sender, and receiver covariates. Note: the z-values are computed as posterior mean / posterior SD, and the p-values are derived from a normal approximation. These are convenient screening statistics but are not formal frequentist test statistics. For rigorous inference, use the credible intervals or examine the full posterior via the BETA matrix directly.

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`.

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
`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

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.

`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 but is *not* a classical test. It is a two-sided Normal approximation based on the posterior mean and marginal posterior standard deviation, matching the calculation in `summary(fit)`. Use it as a compact signal that the marginal posterior is far from zero, and report it alongside the `conf.low` / `conf.high` credible interval. When sign certainty matters, compute it 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)
fit <- lame(YX_bin_list$Y, YX_bin_list$X, family = "binary", R = 0,
            nscan = 100, burn = 20, odens = 5, verbose = FALSE)
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 sandwich covariance (`vcov.ame_als`) by default, or from the bootstrap object attached to `x$bootstrap` when present (preferred, fully propagated). `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

Only the intercept and dyadic-covariate coefficients are returned, matching `coef(fit)` on the sandwich-covered subset. Additive (`a`, `b`), multiplicative (`U`, `V`), and node-covariate 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"` or `"sandwich"`.

Examples

```
data(YX_bin_list)
Y1 <- 1 * (YX_bin_list$Y[[1]] > 0); diag(Y1) <- NA
fit <- ame_als(Y = Y1, Xdyad = YX_bin_list$X[[1]],
               family = "binary", 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"),
  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, or "all" (default) for both
include	Character vector of specific parameter names to include. Raw variance-component names ("va", "ve", "rho", ...) and their display labels ("Error Variance", ...) are both accepted. 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 to exclude as burn-in when calculating statistics (default 0, assumes burn-in already removed)
thin	Thinning interval for display (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)

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)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE)

# 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")
```

---

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)
fit <- lame(YX_bin_list$Y, YX_bin_list$X, family = "binary",
            burn = 5, nscan = 20, odens = 1, verbose = FALSE)
# 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`), it delegates to `ame_als_refit` for a fast warm-started refit;
- 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	Size of nodes, or "degree" to scale by degree (default 3)
label.nodes	Logical; if TRUE, show node labels (default TRUE)
label.size	Size of node labels (default 3)
show.usernames	Integer: number of top-degree nodes to label, or NULL for default behavior
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 colour-blind-safe Okabe-Ito 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:

- Nodes close together in latent space tend to have similar connection patterns
- The distance between sender position (U) and receiver position (V) relates to the likelihood of a tie
- Clustering in the latent space 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)
fit <- ame(YX_nrm$Y, Xdyad = YX_nrm$X, R = 2,
           nscan = 100, burn = 10, odens = 1, verbose = FALSE)

# 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.

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()`. 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.

---

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

`Xbeta_bip_cpp` *Compute Xbeta product for bipartite networks*

Description

Computes sum_k beta_k * X_k for a single time slice

Usage

```
Xbeta_bip_cpp(X, beta)
```

Arguments

`X`	3D array (nA x nB x p) of covariates for one time period
`beta`	Coefficient vector of length p

Value

nA x nB matrix

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, 188
Xcol, 191
Xdyad, 191
Xrow, 191
Y, 192
YX_bin, 192
YX_bin_list, 193
YX_bin_long, 193
YX_cbin, 194
YX_frn, 194
YX_nrm, 195
YX_ord, 195

ab_plot, 6, *7*, *44*, *116*, *118*
ab_plot.ame_als, 7, *27*
absdiff, *21*
absdiff *(*nodematch*)*, 113
addhealthc3, 8
addhealthc9, 9
als_dynamic_beta, 9, *94*, *97*
als_start_vals, 11, *85*, *87*
ame, 11, *11*, *23–27*, *29*, *35*, *38*, *69*, *86*, *88*, *94*, *96*, *97*, *116*, *119*, *174*
ame_als, *11*, *15*, 22, *27–29*, *31*, *39*, *49*, *95*, *97*, *116*, *121*, *137*, *183*, *187*
ame_als_bootstrap, *24–26*, 27, *27*, *30*, *31*, *44*, *50*, *97*, *164*, *179*, *187*, *188*
ame_als_refit, *28*, *29*, 30, *183*
ame_memory_settings, 32
ame_memory_usage, 32
ame_options, 33
ame_parallel, 34, *99*
array_to_list, 35
as_draws, 36
as_draws.ame, 37
as_draws.ame_als *(*as_draws.ame*)*, 37
as_draws.lame *(*as_draws.ame*)*, 37
as_lame_y, *20*, 38
autoplot.ame *(*autoplot.lame*)*, 39
autoplot.ame_als, 39
autoplot.lame, 39, *39*
autoplot.lame_als *(*autoplot.ame_als*)*, 39

boot_ame *(*ame_als_bootstrap*)*, 27
boot_ame-no-fitted, 41

check_format, 41
coef.als_dynamic_beta, 43
coef.ame, *21*, 43
coef.ame_als, 44
coef.boot_ame, 45
coef.lame, *94*
coef.lame *(*coef.ame*)*, 43
coldwar, 45
combine_ame_chains, 46
compact_ame, 46
compute_mcmc_diagnostics, 47
compute_XtX_Xty_bip_cpp, 48
comtrade, 48
confint, *15*, *90*
confint.ame, *44*, 49
confint.ame_als, *24*, *39*, 49, *97*
confint.boot_ame, *29*, *45*, 50, *50*
confint.lame, *176*
confint.lame *(*confint.ame*)*, 49

design_array_listwisedel, 51
detect_change_point, 52
dutchcollege, 53
dynamic_beta_prior_summary, 53, *84*

el2sm, 55
evaluate_heldout, 56

fitted, *41*
fitted.ame, 57, *137*
fitted.ame_als, 58
fitted.boot_ame *(*boot_ame-no-fitted*)*, 41

fitted.lame, 58
forecast_pit, 59
formula.ame, 60
formula.lame *(formula.ame)*, 60

get_design_rep, 61
get_EZ_dynamic_beta_cpp, 62
get_fit_object, 63
get_start_vals, 65
ginv, *24*
glance, 66
glance.ame, 66
glance.ame_als, 67
glance.lame *(glance.ame)*, 66
glance.lame_als *(glance.ame_als)*, 67
gof, *18*, *21*, 68, *94*, *109*
gof_plot, *18*, *21*, *69*, 70, *73*, *75*, *94*, *109*, *116*, *118*
gof_plot.ame_als, *27*, *70*, 73
gof_stats, 74
gof_stats_bipartite, 76
gof_stats_unipartite, 77
gof_temporal, 78

init_dynamic_ab_cpp, 79
init_dynamic_positions, 80
IR90s, 81

lame, *10, 11*, *15, 16*, *21*, *23*, *25, 26*, *29*, *35*, *38*, *69*, 81, *95–97*, *99*, *101, 102*, *118*, *170–172*, *176*
lame_als, *10, 11*, *26–29*, *39*, *90*, *94*, 95, *116*, *121*, *137*
lame_multi, *35*, 98
lame_parallel, 99, *99*
lame_resume, *89, 90*, *94*, 100
lame_snap_als, *26*, *90*, *94*, *97*, 101, *170–172*
latent_positions, *16*, *21*, *27*, *44*, *94*, 105, *133*
lazegalaw, 107
lfo, 107
list_to_array, 108
logLik.ame, 109, *112*
logLik.ame_als, 109
logLik.lame *(logLik.ame)*, 109
loo, 110, *110*
loo.ame, 110
loo.ame_als *(loo.ame)*, 110
loo.lame *(loo.ame)*, 110

mhalf, 112

nobs.ame, 112, *113*
nobs.ame_als, 113
nobs.lame *(nobs.ame)*, 112
nodefactor, *21*
nodefactor *(nodematch)*, 113
nodematch, *21*, 113

per_actor_slopes, 114
plot.ame, 115
plot.ame_als, 116
plot.lame, 117
posterior_options, *15*, *87, 88*, *106*, 118, *169*
posterior_quantiles, 119
predict.ame, *44*, *57*, 120, *125*, *137*
predict.ame_als, 121
predict.lame, 122
prediction_draws_long, 123
print.als_dynamic_beta, 124
print.ame, 124
print.ame.sim, 126
print.ame_als, 126
print.boot_ame, 127
print.gof_temporal, 127
print.lame, 128
print.lame.sim *(print.ame.sim)*, 126
print.lame_multi, 128
print.lfo_lame, 129
print.per_actor_slopes, 129
print.summary.ame, 130, *174*
print.summary.ame_als, 130
print.summary.boot_ame *(summary.boot_ame)*, 175
print.summary.lame, 131, *176*
prior_summary, 131
procrustes_align, *16*, *21*, *94*, *106*, 132

rbeta_ab_bip_gibbs_cpp, 134
read_log_lik, 135
reconstruct_EZ, 135
reconstruct_UVPM *(reconstruct_EZ)*, 135
residuals, *41*
residuals.ame, *57*, 136
residuals.ame_als, 137
residuals.boot_ame *(boot_ame-no-fitted)*, 41
residuals.lame, 138
rhat_dynamic_beta, 138

rmvnorm, 139
rSab_fc, 140
rUV_dynamic_bip_fc_cpp, 141
rUV_dynamic_fc, 142
rUV_dynamic_fc_cpp, 142
rUV_dynamic_snap_fc, 143
rUV_dynamic_snap_fc_cpp, 144
rUV_dynamic_t_fc, 146
rUV_dynamic_t_fc_cpp, 147
rUV_sym_fc, 148
rZ_bin_bip_batch_cpp, 149
rZ_nrm_batch_cpp, 149
rZ_nrm_fc, 150
rZ_pois_fc, 150

sample_beta_dynamic_cpp, 151
sample_beta_static_cpp, 152
sample_dynamic_ab_cpp, 153
sample_rho_ab_cpp, 154
sample_rho_beta_cpp, 155
sample_rho_uv, 156
sample_sigma_ab_cpp, 157
sample_sigma_beta_cpp, 158
sample_sigma_uv, 158
sampler_describe, 159
sampsonmonks, 160
sheep, 160
simulate.ame, *125*, 161
simulate.ame_als, *27*, *73*, 164
simulate.lame, 164
simulate_posterior, 168
simY_pois, 169
snap_category_summary, 170
snap_index_draws, 171
snap_index_summary, 171
snap_rank_summary, 172
summary.ame, *21*, *44*, *125*, 173
summary.ame.sim, 174
summary.ame_als, 174
summary.boot_ame, 175
summary.lame, *94*, 175
summary.lame.sim, 177

tidy, 177
tidy.ame, 178
tidy.ame_als, 179
tidy.boot_ame, 180
tidy.lame *(*tidy.ame*)*, 178
tidy.lame_als *(*tidy.ame_als*)*, 179
trace_plot, *27*, *116*, *118*, 180

update, *182*, *183*
update.ame, 182, *183*
update.ame_als, 183
update.lame *(*update.ame*)*, 182
update.lame_als *(*update.ame_als*)*, 183
uv_plot, *27*, *106*, *116*, *118*, *133*, 184

vcov.ame, *44*, 186
vcov.ame_als, *26*, *39*, *49*, *50*, *179*, 187
vcov.boot_ame, *45*, 188
vcov.lame *(*vcov.ame*)*, 186
vignette_data, 188, *191*, *192*

waic, *189*
waic *(*loo*)*, 110
waic.ame, 189
waic.ame_als *(*waic.ame*)*, 189
waic.lame *(*waic.ame*)*, 189

Xbeta, 190
Xbeta_bip_cpp, 190
Xcol, 191
Xdyad, 191
Xrow, 191

Y, 192
YX_bin, 192
YX_bin_list, 193
YX_bin_long, 193, *193*
YX_cbin, 194
YX_frn, 194
YX_nrm, 195
YX_ord, 195