Package 'Bergm.ws'

November 26, 2026

Type Package

Title Bayesian Multi-Layer ERGMs for Weighted and Signed Weighted Networks

Version 0.1.0

Description Bayesian hierarchical multi-layer dissolution exponential random graph models for weighted and signed weighted networks. A weighted network is decomposed into nested binary layers by thresholding its edge weights, and each transition between consecutive layers is modelled conditionally on the observed lower layers: on the layers alone with bergm_w(), or jointly with the edge signs with bergm_ws(). Layer-specific parameters are partially pooled through a Normal-inverse-Wishart hierarchical prior and estimated with an adaptive approximate exchange algorithm: joint or layer-wise updates, delayed acceptance with a Gaussian approximation of the likelihood, the hierarchical mean integrated out of the exchange step, and a proposal covariance initialised from the Fisher information, learned during the burn-in and pooled across chains. The package also provides a model builder that checks that every statistic is computed on its own layer, prior and posterior predictive checks with goodness-of-fit statistics not used in the model, convergence diagnostics and a check of the length of the auxiliary chains.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports ergm (>= 4.6.0), ergm.multi (>= 0.2.0), network, coda, stats, parallel

Suggests knitr, quarto

NeedsCompilation no

Maintainer Alberto Caimo <alberto.caimo1@ucd.ie>

Config/roxygen2/version 8.1.1

Author Alberto Caimo [aut, cre] (ORCID: <https://orcid.org/0000-0001-8956-7166>), Isabella Gollini [aut] (ORCID: <https://orcid.org/0000-0002-7738-5688>)

Repository CRAN

Date/Publication 2026-11-26 10:10:30 UTC

Contents

Bergm.ws-package 2
bergm_w 4
summary.bergm_w 7
toy_ws 8
ws_aux_check 9
ws_data 10
ws_model 11
ws_predictive 12
ws_prior 14
ws_simulate 15

Index **16**

`Bergm.ws-package` *Bayesian multi-layer dissolution ERGMs for weighted and signed weighted networks*

Description

A weighted network is decomposed into nested binary layers $x_0 \supseteq x_1 \supseteq \cdots \supseteq x_K$ by thresholding its edge weights. Each transition $x_{k-1} \to x_k$ is modelled by an exponential random graph model whose statistics are computed on layer k, conditionally on the observed lower layers: `bergm_w` fits the model to the layers alone. When the sign z of every edge of x_0 is also recorded, `bergm_ws` models the signs jointly with the layers:

$$p(z, x_{1:K} \mid \tilde{x}_{0:K-1}, \varphi_{1:K}) \propto \exp\Big\{ \sum_{k=1}^{K} \varphi_k^\top s(z, x_k) \Big\} \prod_{k=1}^{K} 1\{x_k \subseteq \tilde{x}_{k-1}\},$$

where $\tilde{x}_{k-1}$ is the *observed* preceding layer. The model is therefore conditional on the observed lower layers. The layer-specific parameters are partially pooled by a Normal-inverse-Wishart prior, $\varphi_k \mid \mu, \Sigma \sim N(\mu, \Sigma)$, $\mu \sim N(\mu_0, \Sigma_0)$, $\Sigma \sim IW(\nu_0, S_0)$, and the posterior is sampled with an adaptive approximate exchange algorithm (`bergm_w`, `bergm_ws`): delayed acceptance with a Gaussian approximation of the likelihood, so that most proposals that would be rejected are discarded without simulating a network; the hierarchical mean integrated out of the exchange step; and a proposal covariance initialised from the Fisher information, learned during the burn-in, pooled across chains and frozen for the retained draws.

Details

The typical workflow is:

1. `ws_data`: build the layers from a weight matrix, a sign matrix and the thresholds (`ws_simulate` creates a toy example; `toy_ws` is one).
2. `ws_model`: write the model as an **ergm.multi** formula on the layers `x_1`, ..., `x_K` and the signs `z`, or once as a template with `x_k`. The formula is checked: every statistic must use exactly one layer and every layer the same statistics; nearly collinear statistics are flagged.
3. `ws_prior`, `ws_predictive` with `type = "prior"`: prior and prior predictive check.
4. `bergm_w` or `bergm_ws`: adaptive exchange sampler, then `summary.bergm_w` for estimates and convergence diagnostics, `as.mcmc.list()` for the draws.
5. `ws_predictive`: posterior predictive check, including statistics not used in the model; `ws_aux_check`: are the auxiliary chains long enough?

Author(s)

Alberto Caimo, Paolo Dalena, Isabella Gollini

References

Caimo, A. and Friel, N. (2011). Bayesian inference for exponential random graph models. *Social Networks*, 33, 41–55.

Caimo, A. and Gollini, I. (2020). A multi-layer exponential random graph modelling approach for weighted networks. *Computational Statistics & Data Analysis*, 142, 106825.

Caimo, A., Dalena, P, and Gollini, I. (2026). Multi-layer dissolution exponential-family models for weighted signed networks. arxiv.org/abs/2511.03420

Murray, I., Ghahramani, Z. and MacKay, D. (2006). MCMC for doubly-intractable distributions. *Proceedings of UAI*.

Christen, J. A. and Fox, C. (2005). Markov chain Monte Carlo using an approximation. *Journal of Computational and Graphical Statistics*, 14, 795–810.

Haario, H., Saksman, E. and Tamminen, J. (2001). An adaptive Metropolis algorithm. *Bernoulli*, 7, 223–242.

Krivitsky, P. N., Koehly, L. M. and Marcum, C. S. (2020). Exponential-family random graph models for multi-layer networks. *Psychometrika*, 85, 630–659.

van Dyk, D. A. and Park, T. (2008). Partially collapsed Gibbs samplers: theory and methods. *Journal of the American Statistical Association*, 103, 790–796.

See Also

```
vignette("Bergm.ws", package = "Bergm.ws")
```

bergm_w *Bayesian multi-layer dissolution ERGMs for weighted and signed weighted networks*

Description

`bergm_w()` fits the model to a weighted network without signs, with statistics computed on each layer x_k; `bergm_ws()` fits it to a signed weighted network, with the signs modelled jointly with the layers and statistics computed on the positive and negative edges of each layer.

Both sample the posterior of the layer-specific parameters φ_k and of the hierarchical parameters (μ, Σ) with an adaptive approximate exchange algorithm. Each exchange step simulates an auxiliary network from the joint model on the observed supports, starting at the observed network, so that the intractable normalising constant cancels; μ and Σ are then drawn from their Normal and inverse-Wishart full conditionals.

Compared with the basic algorithm, proposals are screened by delayed acceptance, the hierarchical mean is integrated out of the exchange step, and the proposal covariance is learned from the chains during the burn-in, pooled across chains and then frozen. None of these changes the posterior.

Usage

```
bergm_w(data, formula = NULL, scale = NULL, labels = NULL, prior = NULL,
        update = c("joint", "layer"), n_iter = 10000,
        burnin = floor(n_iter / 4), thin = 1, n_chains = 4, cores = 1,
        aux_iters = NULL, init = "mple", prop_cov = "information",
        adapt_cov = TRUE, pool_adaptation = TRUE, adapt_every = NULL,
        adapt_start = NULL, adapt_scaling = 0.1, target_accept = 0.234,
        delayed = TRUE, collapse_mu = TRUE, n_info = 200, n_newton = 2,
        dispersion = 4, seed = NULL, verbose = TRUE)

bergm_ws(data, formula = NULL, scale = NULL, labels = NULL, prior = NULL,
         update = c("joint", "layer"), n_iter = 10000,
         burnin = floor(n_iter / 4), thin = 1, n_chains = 4, cores = 1,
         aux_iters = NULL, init = "mple", prop_cov = "information",
         adapt_cov = TRUE, pool_adaptation = TRUE, adapt_every = NULL,
         adapt_start = NULL, adapt_scaling = 0.1, target_accept = 0.234,
         delayed = TRUE, collapse_mu = TRUE, n_info = 200, n_newton = 2,
         dispersion = 4, seed = NULL, verbose = TRUE)
```

Arguments

`data`	a `ws_data` object, or a `ws_model` (then `formula`, `scale` and `labels` are not needed). `bergm_w()` uses only the weights of signed data; `bergm_ws()` needs signed data.
`formula, scale, labels`	the **ergm.multi** formula on the layers `x_1`, …, `x_K` (or the template with `x_k`) and, for `bergm_ws()`, the signs `z`; multipliers and names of the statistics. See `ws_model`.

`prior` a `ws_prior`; default `ws_prior(model)`.

`update` `"joint"`: one exchange step for all Kp parameters per iteration, with one auxiliary network; robust to posterior correlations both within and between layers. `"layer"`: one exchange step per layer with the other layers held fixed (K auxiliary networks per iteration); adequate when parameters of different layers are nearly uncorrelated.

`n_iter`, `burnin`, `thin` iterations per chain, burn-in and thinning.

`n_chains` number of chains. The first starts at the initial value, the others at draws from $N(\varphi^{(0)}, \texttt{dispersion} \times B^{(0)})$, so that convergence can be checked with $\hat{R}$.

`cores` chains run in parallel with `parallel::mclapply` when `cores > 1` (not on Windows).

`aux_iters` MCMC steps for each auxiliary network. The default is ten proposals per dyad-layer pair that can change (at least 2000); check it with `ws_aux_check`.

`init` `"mple"` (default): penalised maximum pseudo-likelihood, i.e. the logistic pseudo-likelihood times the Gaussian prior, which is always finite; `"zero"`; or a numeric vector.

`prop_cov` initial proposal covariance $B^{(0)}$: `"information"` (default) uses $\{\mathrm{Cov}(s) + P\}^{-1}$, with $\mathrm{Cov}(s)$ the covariance of statistics simulated at the starting value (the Fisher information) and P the prior precision; `"diagonal"` uses $0.01I$; or a matrix.

`adapt_cov` if `TRUE`, the proposal covariance is re-estimated from the second half of the chain's history at every adaptation step (adaptive Metropolis), mixed 95:5 with $B^{(0)}$ so that it cannot collapse.

`pool_adaptation` with several chains and `adapt_cov = TRUE`, the burn-in has three phases. (1) During the first three quarters every chain adapts its own proposal covariance and scale. (2) The within-chain covariances are averaged, the pooled matrix is given to every chain, and during the last quarter of the burn-in the chains only tune the proposal scale. (3) The scale is set to the geometric mean of the chains' scales and the proposal is frozen. The retained draws of all chains then come from the same Markov kernel, with a covariance estimated from four times as many draws, and a chain that adapted to a poor local shape early on is corrected.

`adapt_every`, `adapt_start` adaptation interval and first adaptation iteration; by default `max(20, burnin %/% 20)` and twice that. Adaptation stops at the end of the burn-in, so the retained draws come from a Markov chain with a fixed kernel.

`adapt_scaling` the proposal scale is multiplied by $1 \pm$ `adapt_scaling` according to the acceptance rate in the last window.

`target_accept` target acceptance rate.

`delayed` delayed acceptance (Christen and Fox, 2005). Each proposal is first screened with a Gaussian approximation of the log-likelihood, $g^\top(\varphi - c) - (\varphi - c)^\top H(\varphi - c)/2$, where H and g are the Fisher information and the score at the starting value c, together with the exact prior; the auxiliary network is simulated only for proposals that pass, and a second stage with the exchange ratio divided by

the approximate likelihood ratio keeps the posterior exact. Most proposals that would be rejected are discarded without simulation; on the toy example this gives about 2.5 times more effective samples per second.

`collapse_mu` if `TRUE` (default) the exchange step targets $p(\varphi \mid \Sigma, y)$, with the hierarchical mean integrated out: the prior ratio uses $\varphi \sim N(1_K \otimes \mu_0, I_K \otimes \Sigma + J_K \otimes \Sigma_0)$. μ is then drawn from its full conditional given the new φ and Σ, and Σ given φ and μ: a partially collapsed Gibbs sampler (van Dyk and Park, 2008) with the same stationary distribution. It removes the dependence between φ and μ that slows the plain Gibbs scheme when the layers move together; on the toy example it gives about 1.7 times more effective samples for the same number of iterations.

`n_info` networks simulated to estimate the Fisher information.

`n_newton` moment-matching Newton steps from the penalised MPLE towards the posterior mode before the chains start: $\varphi \leftarrow \varphi + \{\mathrm{Cov}(s) + P\}^{-1}\{s_{obs} - E_\varphi(s) - P(\varphi - m)\}$.

`dispersion` variance inflation of the starting points of chains 2, 3, ...

`seed` random seed (each chain gets its own seed).

`verbose` print progress.

Details

The initial proposal scale is $2.38/\sqrt{2d}$ for a block of d parameters: the usual random-walk scale, reduced by $\sqrt{2}$ because the exchange step adds noise to the acceptance ratio with about the same variance as the exact log-likelihood ratio.

Statistics enter in the rescaled form of `ws_model`; use `coef(fit, scale = "ergm")` for coefficients on the **ergm** scale.

Value

An object of class `"bergm_w"` (and `"bergm_ws"` for signed networks): the model, the prior, the chains (draws of φ, μ and the lower triangle of Σ, acceptance indicators, final proposal scales and covariances), the settings and the initialisation (starting value, initial proposal covariance and a fit check of the starting value).

See Also

`summary.bergm_w`, `ws_predictive`, `ws_aux_check`

Examples

```
data(toy_ws)
y <- ws_data(toy_ws$weights,
             toy_ws$thresholds,
             toy_ws$signs,
             toy_ws$attributes)

x <- ws_data(toy_ws$weights,
             toy_ws$thresholds)
```

```
f <- ~ L(~edges, ~(z & x_k)) +
       L(~edges, ~(!z & x_k)) +
       gwespL(0.5,
              fixed = TRUE,
              L.base = ~(z & x_k),
              Ls.path = list(~(z & x_k), ~(z & x_k)))

fit_ws <- bergm_ws(y,
                   f,
                   scale = "gwesp",
                   n_iter = 300,
                   burnin = 100,
                   n_chains = 2,
                   seed = 1)
summary(fit_ws)

fit_w <- bergm_w(x,
                 ~ L(~edges, ~x_k) + L(~triangle, ~x_k),
                 n_iter = 300,
                 burnin = 100,
                 n_chains = 2,
                 seed = 1)
summary(fit_w)
```

`summary.bergm_w` *Posterior summaries and convergence diagnostics*

Description

Posterior means, standard deviations and quantiles of the hierarchical means μ, the layer-specific parameters φ_k and the between-layer standard deviations $\Sigma_{rr}^{1/2}$, with the potential scale reduction factor $\hat{R}$ and the effective sample size of every parameter, the acceptance rate of each chain after burn-in and the largest posterior correlation between parameters of different layers.

Usage

```
## S3 method for class 'bergm_w'
summary(object, probs = c(0.025, 0.5, 0.975), ...)

## S3 method for class 'summary.bergm_w'
print(x, digits = 3, ...)

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

## S3 method for class 'bergm_w'
```

```
coef(object, what = c("phi", "mu"), scale = c("model", "ergm"), ...)

## S3 method for class 'bergm_w'
as.mcmc.list(x, what = c("phi", "mu", "sd"), ...)
```

Arguments

`object`, `x`	an `bergm_w` (or its summary).
`probs`	posterior quantiles to report.
`digits`	digits for printing.
`what`	`"phi"`: layer-specific parameters; `"mu"`: hierarchical means; `"sd"`: between-layer standard deviations.
`scale`	`"model"`: the (possibly rescaled) scale of the model; `"ergm"`: coefficients of the unscaled **ergm** statistics.
`...`	ignored.

Details

$\hat{R}$ uses `coda::gelman.diag` without discarding further draws and needs at least two chains; values above 1.05, or effective sample sizes below 100, call for longer chains. A large correlation between parameters of different layers suggests `update = "joint"`.

Value

`summary` returns an object of class `"summary.bergm_w"`; `coef` a $K \times p$ matrix (or a vector for `what = "mu"`); `as.mcmc.list` a coda `mcmc.list`.

See Also

`bergm_w`

`toy_ws` *A small simulated signed weighted network*

Description

A signed weighted network on 40 nodes simulated with `ws_simulate(n = 40, seed = 33)`. It is entirely synthetic: three modules of nodes with strong positive edges inside them, a minority faction whose edges with the others tend to be negative and weak, and signs that are noisier for weak edges. As in co-expression networks built from interaction confidence scores, the share of positive edges grows with the weight (77%, 87%, 94% and 97% in the four layers).

Usage

```
data(toy_ws)
```

Format

A list with components

`weights` 40 x 40 symmetric matrix of edge weights in (0, 1).

`signs` 40 x 40 symmetric matrix of signs (± 1).

`thresholds` the four thresholds defining layers x_0 to x_3.

`attributes` data frame with the node attributes `module` (three modules) and `pathway` (a binary annotation that does not affect the edges).

Examples

```
data(toy_ws)
dat <- ws_data(toy_ws$weights, toy_ws$thresholds, toy_ws$signs, toy_ws$attributes)
dat
ws_data(toy_ws$weights, toy_ws$thresholds, attributes = toy_ws$attributes)
```

---

ws_aux_check *Are the auxiliary chains long enough?*

Description

The exchange algorithm needs auxiliary networks drawn from the model. They are obtained by running `aux_iters` MCMC steps from the observed network, so they are only approximately exact. At the posterior mean, this function compares the statistics of `n` such networks with those of `n` networks from chains `factor` times longer.

Usage

```
ws_aux_check(fit, n = 100, factor = 20, seed = NULL)

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

Arguments

`fit`	an `bergm_w`.
`n`	networks per chain length.
`factor`	length of the long chains relative to `aux_iters`.
`seed`	random seed.
`x`	an `ws_aux_check` object.
`...`	ignored.

Value

A data frame with, for each statistic, the means of the short and long chains, the long-chain standard deviation, the standardised difference of the means and its z-score. A statistic is flagged when the difference exceeds a quarter of a standard deviation and its z-score exceeds the Bonferroni threshold $\Phi^{-1}(1 - 0.025/m)$ for m statistics, so that a single large z-score among many statistics is not mistaken for a problem; flagged statistics suggest increasing `aux_iters`. With fewer than about 100 networks the Monte Carlo error of the means is large.

See Also

`bergm_w`

`ws_data` *Nested layers from edge weights, with optional signs*

Description

Thresholds a weighted network into nested binary layers $x_k = \{ij : w_{ij} \geq t_k\}$, $k = 0, \ldots, K$, optionally records the sign of every edge of x_0, and builds the multilayer network and the support constraints used by the model: x_k varies on the dyads of the observed x_{k-1}, and the signs, when given, on the dyads of x_0. Without signs the data are for `bergm_w`, with signs for `bergm_ws`.

Usage

```
ws_data(weights, thresholds, signs = NULL, attributes = NULL)

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

Arguments

`weights`	symmetric numeric matrix of edge weights (for example interaction confidence scores); `NA` means no edge.
`signs`	optional symmetric matrix whose sign gives the sign of each edge (for example a correlation matrix). Only dyads of x_0 are used, and they must have a non-zero sign.
`thresholds`	strictly increasing vector of length $K + 1$; the first defines the baseline layer x_0.
`attributes`	optional data frame of node attributes, one row per node, available to the terms of the model formula.
`x`	an `ws_data` object.
`...`	ignored.

Value

An object of class `"ws_data"`: a list with the multilayer network `y` (signs, if any, and layers $x_1, \ldots, x_K$), the layers `x` ($x_0, \ldots, x_K$), the matrix `levels` (for each dyad, the highest layer containing it, or -1), the sign matrix `signs`, the support `constraint` formula and the inputs.

See Also

ws_model

Examples

```
data(toy_ws)

y <- ws_data(toy_ws$weights, toy_ws$thresholds, toy_ws$signs, toy_ws$attributes)
y

x <- ws_data(toy_ws$weights, toy_ws$thresholds)
x
```

---

ws_model *Specify a multi-layer dissolution ERGM with an ergm.multi formula*

Description

Takes an **ergm.multi** formula on the layers of a ws_data object and checks it: every statistic must refer to exactly one layer, the statistics must be grouped by layer, and every layer must have the same statistics, because the hierarchical prior pools the j-th statistic of all layers. Statistics equal to zero in the observed network and statistics whose change statistics are nearly collinear on the dyads that can change are reported (their parameters are then separated mostly by the prior).

Usage

```
ws_model(data, formula, scale = NULL, labels = NULL)

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

Arguments

data	a ws_data object.
formula	a one-sided **ergm.multi** formula. The layers are called x_1, ..., x_K and, for signed data, the signs z. A formula that uses x_k instead of numbered layers is a template: it is written once and repeated for $k = 1, \ldots, K$.
scale	multipliers of the statistics: NULL (none), "gwesp" (every GWESP statistic with fixed decay α is multiplied by $e^{-\alpha}$), or a numeric vector of length p (recycled over the layers) or Kp.
labels	optional names of the p statistics of a layer; by default they are derived from the **ergm** names, for example edges+ for L(~edges, ~(z & x_k)) and gwesp-++ for a GWESP term on negative edges with positive shared partners.
x	a ws_model object.
...	ignored.

Details

Any **ergm** or **ergm.multi** term can be used, applied to a layer with `L()` or with the layer-logic terms of **ergm.multi**. In signed data, `(z & x_k)` selects the positive edges of layer k and `(!z & x_k)` its negative edges; for example

```
~ L(~edges, ~(z & x_k)) + L(~edges, ~(!z & x_k)) +
  gwespL(0.5, fixed = TRUE, L.base = ~(z & x_k),
         Ls.path = list(~(z & x_k), ~(z & x_k)))
```

models the persistence of positive and negative edges and positive triangle closure in every layer. Without signs the layers are used directly, as in `L(~edges, ~x_k)`. Objects named in the formula (a decay value, a vector of attributes) are taken from its environment.

The coefficient of a statistic multiplied by `scale` is on the scale of the multiplied statistic; with `scale = "gwesp"` a GWESP coefficient is the change in log-odds for an edge with many shared partners, on the same scale in every layer, which helps the hierarchical prior pool comparable quantities. `coef(fit, scale = "ergm")` returns coefficients on the original **ergm** scale.

A warning is given when the j-th statistics of different layers are not the same term.

Value

An object of class `"ws_model"` with the expanded formula (`rhs`), the template if one was used, the statistic labels, the observed statistics (`s_obs`, and `s_obs_scaled` after multiplication by `scale`), the multipliers `scale` and the change-statistic correlations `change_cor`.

See Also

`ws_prior`, `bergm_w`

---

ws_predictive *Prior and posterior predictive checks*

Description

Simulates networks from the prior or from the posterior and compares them with the observed network, both on the model statistics and on statistics that are not in the model: for every layer, the numbers of triangles of each sign type (`+++`, `-++`, `+--`, `---`) and the standard deviation of the positive degrees.

Usage

```
ws_predictive(object, type = c("posterior", "prior"), prior = NULL,
              nsim = 200, burnin = NULL, gof = TRUE, seed = NULL,
              verbose = TRUE)

## S3 method for class 'ws_predictive'
print(x, digits = 2, ...)
```

Arguments

object	an bergm_w (posterior or prior check) or an ws_model (prior check only).
type	"posterior" or "prior".
prior	prior for the prior check (default: that of the fit, or ws_prior(model)).
nsim	number of simulated networks, one per parameter draw.
burnin	MCMC steps per simulated network, started at the observed network; default five times the auxiliary chain length.
gof	also compute the statistics not in the model (needs the simulated networks, slower).
seed	random seed.
verbose	print progress.
x	an ws_predictive object.
digits	digits for printing.
...	ignored.

Details

The simulated networks are conditional on the observed lower layers (see Bergm.ws-package), so each transition is checked given what is observed below it. The mid-p value is the share of simulated values below the observed one plus half the share of ties; values near 0 or 1 flag statistics that the model does not reproduce.

Value

An object of class "ws_predictive" with the simulated (stats, gof) and observed (obs, gof_obs) statistics and the parameter draws used.

See Also

bergm_w

Examples

```
data(toy_ws)

y <- ws_data(toy_ws$weights, toy_ws$thresholds, toy_ws$signs, toy_ws$attributes)

f <- ~ L(~edges, ~(z & x_k)) +
       L(~nodematch("pathway"), ~(z & x_k)) + L(~edges, ~(!z & x_k))

mod <- ws_model(y, f)

pp <- ws_predictive(mod, type = "prior", nsim = 50, seed = 1)
pp
```

`ws_prior` *Hierarchical prior for the layer-specific parameters*

Description

Normal-inverse-Wishart prior: $\varphi_k \mid \mu, \Sigma \sim N(\mu, \Sigma)$ independently for $k = 1, \ldots, K$, $\mu \sim N(\mu_0, \Sigma_0)$ and $\Sigma \sim IW(\nu_0, S_0)$, with $E[\Sigma] = S_0/(\nu_0 - p - 1)$.

Usage

```
ws_prior(model, mu0 = 0, Sigma0 = 3, nu0 = NULL, S0 = 1)

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

Arguments

`model`	an `ws_model` (or the number of statistics per layer).
`mu0`	prior mean of μ, recycled.
`Sigma0`	prior covariance of μ: a scalar (times the identity) or a matrix.
`nu0`	inverse-Wishart degrees of freedom; default $p + 9$, so that $E[\Sigma] = S_0/8$, a between-layer standard deviation of about 0.35 when $S_0 = I$.
`S0`	inverse-Wishart scale: a scalar (times the identity) or a matrix.
`x`	an `ws_prior` object.
`...`	ignored.

Details

With few layers, the between-layer covariance Σ is determined mostly by its prior: check the sensitivity of the conclusions to `nu0` and `S0`.

Value

An object of class `"ws_prior"`.

Examples

```
data(toy_ws)

y <- ws_data(toy_ws$weights, toy_ws$thresholds, toy_ws$signs, toy_ws$attributes)

m <- ws_model(y, ~ L(~edges, ~(z & x_k)) + L(~edges, ~(!z & x_k)))

ws_prior(m)
```

`ws_simulate` *Simulate a small weighted network, signed or not*

Description

Generates a synthetic weighted network with modular structure, for examples and tests. Nodes belong to modules placed on a circle; the weight of a dyad decreases with the distance between its nodes, so that strong edges are clustered. With `signed = TRUE`, a minority faction has mostly negative and weak edges with the other nodes, and signs are flipped more often for weak edges, so that the share of positive edges increases with the weight.

Usage

```
ws_simulate(n = 40, K = 3, n_modules = 3, density = 0.30,
            retention = c(0.55, 0.50, 0.50), signed = TRUE,
            faction_share = 0.18, sign_noise = 0.08, seed = NULL)
```

Arguments

`n`	number of nodes.
`K`	number of layer transitions (layers $x_0, \ldots, x_K$).
`n_modules`	number of modules.
`density`	share of dyads in the baseline layer x_0.
`retention`	share of the edges of x_{k-1} kept in x_k, recycled to length `K`.
`signed`	if `FALSE`, no signs and no faction effect.
`faction_share`	probability that a node belongs to the minority faction.
`sign_noise`	probability of flipping the sign of the weakest edges; it decreases linearly to zero for the strongest ones.
`seed`	optional random seed.

Value

A list with `weights` (symmetric matrix in (0, 1)), `signs` (symmetric matrix of ± 1, only if `signed`), `thresholds` (length `K + 1`) and `attributes` (data frame with `module` and `pathway`), ready for `ws_data`.

See Also

`toy_ws`, `ws_data`

Examples

```
sim <- ws_simulate(n = 30, seed = 1)

y <- ws_data(sim$weights, sim$thresholds, sim$signs, sim$attributes)
y
```

Index

∗ **datasets**
toy_ws, 8
∗ **package**
Bergm.ws-package, 2

as.mcmc.list.bergm_w *(*summary.bergm_w*)*, 7

Bergm.ws *(*Bergm.ws-package*)*, 2
Bergm.ws-package, 2
bergm_w, *2, 3*, 4, *8–10*, *12, 13*
bergm_ws, *2, 3*, *10*
bergm_ws *(*bergm_w*)*, 4

coef.bergm_w *(*summary.bergm_w*)*, 7

print.bergm_w *(*summary.bergm_w*)*, 7
print.summary.bergm_w *(*summary.bergm_w*)*, 7
print.ws_aux_check *(*ws_aux_check*)*, 9
print.ws_data *(*ws_data*)*, 10
print.ws_model *(*ws_model*)*, 11
print.ws_predictive *(*ws_predictive*)*, 12
print.ws_prior *(*ws_prior*)*, 14

summary.bergm_w, *3*, *6*, 7

toy_ws, *3*, 8, *15*

ws_aux_check, *3*, *5, 6*, 9
ws_data, *3, 4*, 10, *11*, *15*
ws_model, *3, 4*, *6*, *11*, 11, *13, 14*
ws_predictive, *3*, *6*, 12
ws_prior, *3*, *5*, *12*, 14
ws_simulate, *3*, 15