Package 'ergm.sign'

November 21, 2025

Type Package

Title Exponential-Family Models for Signed Networks

Version 0.1.0

Description
Extends the 'ergm.multi' packages from the Statnet suite to fit (temporal) exponential-family random graph models for signed networks. The framework models positive and negative ties as interdependent, which allows estimation and testing of structural balance theory. The package also includes options for descriptive summaries, visualization, and simulation of signed networks. See Krivitsky, Koehly, and Marcum (2020) <doi:10.1007/s11336-020-09720-7> and Fritz, C., Mehrl, M., Thurner, P. W., & Kauermann, G. (2025) <doi:10.1017/pan.2024.21>.

Encoding UTF-8

LazyData true

LazyLoad yes

RoxygenNote 7.3.3

RdMacros Rdpack

Imports statnet.common, network, ergm, ergm.multi, dplyr, magrittr, tergm, Rdpack, intergraph, graphlayouts, vegan, igraph, purrr, graphics, methods, utils

LinkingTo ergm.multi, ergm

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

Depends R (>= 3.5.0)

License MIT + file LICENSE

NeedsCompilation yes

Author Marc Schalberger [cre],
Cornelius Fritz [aut],
Pavel Krivitsky [ctb]

Maintainer Marc Schalberger <m.schalberger@fu-berlin.de>

Repository CRAN

Date/Publication 2025-11-21 15:20:02 UTC

Contents

dse-ergmTerm *Dyadwise shared enemies*

Description

This term adds one network statistic to the model for each element in ‘d‘ where the i th such statistic equals the number of dyads in the network with exactly ‘d[i]‘ shared enemies. For a directed network, multiple shared enemies definitions are possible.

Usage

```
# binary: dse(d, type="OTP", in_order=FALSE)
```

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \rightarrow k \rightarrow j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \rightarrow k \rightarrow i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \rightarrow k, j \rightarrow k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \rightarrow i, k \rightarrow j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

`dsf-ergmTerm` *Dyadwise shared friends*

Description

This term adds one network statistic to the model for each element in ‘d‘ where the i th such statistic equals the number of dyads in the network with exactly ‘d[i]‘ shared friends. For a directed network, multiple shared friends definitions are possible.

Usage

```
# binary: dsf(d, type="OTP", in_order=FALSE)
```

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

`ergm.sign`	*ergm.sign: A Package for Exponential Random Graph Models for Signed Networks*

Description

The ergm.sign package implements tools to simulate and estimate Signed Exponential Random Graph Models and Temporal Signed Exponential Random Graph Models.

Author(s)

Marc Schalberger

`ese-ergmTerm`	*Edgewise shared enemies*

Description

This term adds one network statistic to the model for each element in ‘d‘ where the i th such statistic equals the number of edges in the network with exactly ‘d[i]‘ shared enemies. For a directed network, multiple shared enemy definitions are possible.

Usage

```
# binary: ese(d, type="OTP", L.base=NULL, in_order=FALSE)
```

Arguments

`d`	a vector of distinct integers
`type`	A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.
`base`	specify the base of the triad, either by '+' and '-' or 1 and -1

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘):

vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

`esf-ergmTerm` *Edgewise shared friends*

Description

This term adds one network statistic to the model for each element in ‘d‘ where the i th such statistic equals the number of edges in the network with exactly ‘d[i]‘ shared friends. For a directed network, multiple shared friend definitions are possible.

Usage

```
# binary: esf(d, type="OTP", L.base=NULL, in_order=FALSE)
```

Arguments

`d`	a vector of distinct integers
`type`	A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.
`base`	specify the base of the triad, either by '+' and '-' or 1 and -1

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff

$j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

`fixL-ergmConstraint` *Logical layer constraint*

Description

This layer-aware constraint limits the sample space to those networks for which the specified logical layers are unchanged

Usage

```
# fixL(Ls)
```

See Also

[‘ergmConstraint‘] for index of constraints and hints currently visible to the package.

`GoF` *Conduct Goodness-of-Fit Diagnostics for a Signed ERGM*

Description

Computes the goodness-of-fit (GoF) for a fitted signed exponential random graph model (SERGM). The function simulates new networks using the fitted model and compares key network statistics from the observed network with those from the simulated ones.

Usage

```
GoF(model, nsim = 200, seed = NULL)
```

Arguments

`model`	A fitted signed ERGM (SERGM) object.
`nsim`	Integer; number of simulated networks to generate. Defaults to 200.
`seed`	Optional integer seed for reproducibility. Passed to `set.seed`.

Details

The following diagnostics are plotted:

- Positive degree distribution
- Negative degree distribution
- Edgewise shared enemies distribution (positive edges)
- Edgewise shared enemies distribution (negative edges)
- Edgewise shared friends distribution (positive edges)
- Edgewise shared friends distribution (negative edges)

Value

Produces six diagnostic boxplots comparing observed and simulated statistics for the fitted model.

See Also

`ergm`, `mple_sign`

Examples

```
data("tribes")
fit <- mple_sign(tribes ~ Pos(~edges) + Neg(~edges))
GoF(fit, nsim = 100)
```

`gwdse-ergmTerm` *Geometrically weighted dyadwise shared enemies distribution*

Description

This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared enemies distribution with decay parameter. Note that the GWDSE statistic is equal to the sum of GWNSE plus GWESE. For a directed network, multiple shared friend definitions are possible.

Usage

```
# binary: gwdse(decay, fixed=FALSE, cutoff=30, type="OTP", in_order=FALSE)
```

Arguments

- `decay` — nonnegative decay parameter for the shared enemy or selected directed analogue count; required if ‘fixed=TRUE‘ and ignored with a warning otherwise.
- `fixed` — optional argument indicating whether the ‘decay‘ parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is ‘FALSE‘ , which means the scale parameter is not fixed and thus the model is a curved exponential family.
- `cutoff` — This optional argument sets the number of underlying DSE terms to use in computing the statistics when ‘fixed=FALSE‘, in order to reduce the computational burden. Its default value can also be controlled by the ‘gw.cutoff‘ term option control parameter. (See ‘?control.ergm‘.)
- `type` — A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

`gwdsf-ergmTerm` *Geometrically weighted dyadwise shared friends distribution*

Description

This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared friends distribution with decay parameter. Note that the GWDSF statistic is equal to the sum of GWNSF plus GWESF. For a directed network, multiple shared friend definitions are possible.

Usage

```
# binary: gwdsf(decay, fixed=FALSE, cutoff=30, type="OTP", in_order=FALSE)
```

Arguments

`decay`	nonnegative decay parameter for the shared friend or selected directed analogue count; required if ‘fixed=TRUE‘ and ignored with a warning otherwise.
`fixed`	optional argument indicating whether the ‘decay‘ parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is ‘FALSE‘ , which means the scale parameter is not fixed and thus the model is a curved exponential family.
`cutoff`	This optional argument sets the number of underlying DSF terms to use in computing the statistics when ‘fixed=FALSE‘, in order to reduce the computational burden. Its default value can also be controlled by the ‘gw.cutoff‘ term option control parameter. (See ‘?control.ergm‘.)
`type`	A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \rightarrow k \rightarrow j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \rightarrow k \rightarrow i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \rightarrow k, j \rightarrow k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \rightarrow i, k \rightarrow j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

---

gwese-ergmTerm *Geometrically weighted edgewise shared enemy distribution*

Description

This term adds a statistic equal to the geometrically weighted edgewise (not dyadwise) shared enemy distribution with decay parameter. For a directed network, multiple shared enemy definitions are possible.

Usage

```
# binary: gwese(decay, fixed=FALSE, cutoff=30, type="OTP", base=NULL, in_order=FALSE)
```

Arguments

decay	nonnegative decay parameter for the shared enemy or selected directed analogue count; required if ‘fixed=TRUE‘ and ignored with a warning otherwise.
fixed	optional argument indicating whether the ‘decay‘ parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is ‘FALSE‘ , which means the scale parameter is not fixed and thus the model is a curved exponential family.
cutoff	This optional argument sets the number of underlying ESE terms to use in computing the statistics when ‘fixed=FALSE‘, in order to reduce the computational burden. Its default value can also be controlled by the ‘gw.cutoff‘ term option control parameter. (See ‘?control.ergm‘.)
type	A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.
base	specify the base of the triad, either by ’+’ and ’-’ or 1 and -1

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

`gwesf-ergmTerm` *Geometrically weighted edgewise shared friend distribution*

Description

This term adds a statistic equal to the geometrically weighted edgewise (not dyadwise) shared friend distribution with decay parameter. For a directed network, multiple shared friend definitions are possible.

Usage

```
# binary: gwesf(decay, fixed=FALSE, cutoff=30, type="OTP", base=NULL, in_order=FALSE)
```

Arguments

`decay`	nonnegative decay parameter for the shared friend or selected directed analogue count; required if ‘fixed=TRUE‘ and ignored with a warning otherwise.
`fixed`	optional argument indicating whether the ‘decay‘ parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is ‘FALSE‘ , which means the scale parameter is not fixed and thus the model is a curved exponential family.

`cutoff` This optional argument sets the number of underlying ESF terms to use in computing the statistics when ‘fixed=FALSE‘, in order to reduce the computational burden. Its default value can also be controlled by the ‘gw.cutoff‘ term option control parameter. (See ‘?control.ergm‘.)

`type` A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.

`base` specify the base of the triad, either by ’+’ and ’-’ or 1 and -1

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

`gwnse-ergmTerm` *Geometrically weighted non-edgewise shared enemey distribution*

Description

This term adds a statistic equal to the geometrically weighted nonedgewise (that is, over dyads that do not have an edge) shared enemy distribution with decay parameter. For a directed network, multiple shared enemy definitions are possible.

Usage

```
# binary: gwnse(decay, fixed=FALSE, cutoff=30, type="OTP", base=NULL, in_order=FALSE)
```

Arguments

`decay` nonnegative decay parameter for the shared enemy or selected directed analogue count; required if ‘fixed=TRUE‘ and ignored with a warning otherwise.

`fixed` optional argument indicating whether the ‘decay‘ parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is ‘FALSE‘ , which means the scale parameter is not fixed and thus the model is a curved exponential family.

`cutoff` This optional argument sets the number of underlying NSE terms to use in computing the statistics when ‘fixed=FALSE‘, in order to reduce the computational burden. Its default value can also be controlled by the ‘gw.cutoff‘ term option control parameter. (See ‘?control.ergm‘.)

`type` A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.

`base` specify the base of the triad, either by '+' and '-' or 1 and -1

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

gwnsf-ergmTerm *Geometrically weighted non-edgewise shared friend distribution*

Description

This term adds a statistic equal to the geometrically weighted nonedgewise (that is, over dyads that do not have an edge) shared friend distribution with decay parameter. For a directed network, multiple shared friend definitions are possible.

Usage

```
# binary: gwnsf(decay, fixed=FALSE, cutoff=30, type="OTP", base=NULL, in_order=FALSE)
```

Arguments

`decay`	nonnegative decay parameter for the shared friend or selected directed analogue count; required if ‘fixed=TRUE‘ and ignored with a warning otherwise.
`fixed`	optional argument indicating whether the ‘decay‘ parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is ‘FALSE‘ , which means the scale parameter is not fixed and thus the model is a curved exponential family.
`cutoff`	This optional argument sets the number of underlying NSF terms to use in computing the statistics when ‘fixed=FALSE‘, in order to reduce the computational burden. Its default value can also be controlled by the ‘gw.cutoff‘ term option control parameter. (See ‘?control.ergm‘.)
`type`	A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.
`base`	specify the base of the triad, either by '+' and '-' or 1 and -1

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

`InitErgmTerm.delese` *Delayed edgewise shared enemies*

Description

This term adds one network statistic to the model that counts the number of positive or negative edges in the current network whose endpoints had exactly ‘d‘ shared enemies (common negative ties) in the previous network.

For directed networks, different definitions of shared enemies can be specified using the ‘type‘ argument. For undirected networks, only one configuration applies.

Usage

```
# binary: delese(d = 1, base = "+", type = "OTP")
```

Arguments

`d` Integer. The exact number of shared enemies to count for edges in the current network.

`base` Character indicating which edges in the current network are used as the base: ‘"+"‘ for positive ties or ‘"-"‘ for negative ties.

`type` For directed networks, the definition of shared enemies:

- **"OTP"** Outgoing two-path (\(i -> k -> j \))
- **"ITP"** Incoming two-path (\(j -> k -> i \))
- **"RTP"** Reciprocated two-path (\(i <- k <- j \))
- **"OSP"** Outgoing shared partner (\(i -> k, j -> k \))
- **"ISP"** Incoming shared partner (\(k -> i, k -> j \))

Ignored for undirected networks.

Details

For each edge in the current network (positive or negative, depending on ‘base‘), this term checks how many nodes were connected negatively to both endpoints in the previous network.

`InitErgmTerm.delesf` *Delayed edgewise shared friends*

Description

This term adds one network statistic to the model that counts the number of positive or negative edges in the current network whose endpoints had exactly ‘d‘ shared friends (common positive ties) in the previous network.

For directed networks, different definitions of shared friends can be specified using the ‘type‘ argument. For undirected networks, only one configuration applies.

Usage

```
# binary: delese(d = 1, base = "+", type = "OTP")
```

Arguments

`d`	Integer. The exact number of shared friends to count for edges in the current network.
`base`	Character indicating which edges in the current network are used as the base: ‘"+"‘ for positive ties or ‘"-"‘ for negative ties.
`type`	For directed networks, the definition of shared friends: **"OTP"** Outgoing two-path (\(i -> k -> j \)) **"ITP"** Incoming two-path (\(j -> k -> i \)) **"RTP"** Reciprocated two-path (\(i <- k <- j \)) **"OSP"** Outgoing shared partner (\(i -> k, j -> k \)) **"ISP"** Incoming shared partner (\(k -> i, k -> j \)) Ignored for undirected networks.

Details

For each edge in the current network (positive or negative, depending on ‘base‘), this term checks how many nodes were connected positively to both endpoints in the previous network.

`InitErgmTerm.delnodematch`
Delayed node matching on attribute (lag-1)

Description

Create a nodematch term where node attributes come from the previous network's node attribute ‘attr‘. The previous network used is the one indexed by the current ‘GroupID‘ (equivalent to ‘lag=1‘ previously). This constructs a temporary node attribute ‘delnodecov_<attr>‘ on the current network (copied from the previous net) and calls ‘nodematch‘ on that attribute.

Usage

```
# binary: delnodematch(attr)
```

Arguments

`attr`	character attribute name to copy from the previous network into the current.

`InitErgmTerm.delrecip` *Delayed reciprocity*

Description

For the current network layer ‘base‘, this term equals 1 for each directed edge i->j currently present where the reverse edge j->i was present in the previous network’s same layer. The previous network used is the one indexed by the current ‘GroupID‘ (i.e., the behaviour is the same as the prior ‘lag=1‘ implementation). The term is provided as an edgecov (1/0).

Usage

```
# binary: delrecip(base)
```

Arguments

`base`	character or numeric name/identifier of the layer to examine in the current network.

`InitErgmTerm.gwdelese` *Geometrically weighted delayed edgewise shared enemies*

Description

This term calculates the number of shared enemies based on the previous network. It then applies a geometric transformation to these counts to reduce the influence of large counts. Specifically, if the decay parameter `decay` is provided, the weighting function used is

$$f(k; decay) = 1 - (1 - e^{-decay})^k,$$

where `k` is the count of shared partners and `decay` controls how quickly the weight decreases as counts increase.

Usage

```
# binary: gwdelese(decay, base, type = "OTP")
```

Arguments

`decay` Numeric decay parameter controlling weighting intensity. If `NULL`, raw counts are used.

`base` Character indicating which edges in the current network are used as the base: ‘"+"‘ for positive ties or ‘"-"‘ for negative ties.

`type` Character specifying which shared partner pattern to use; one of `"OTP"` (outgoing two-path), `"ITP"` (incoming two-path), `"RTP"` (reciprocated two-path), `"OSP"` (outgoing shared partner), `"ISP"` (incoming shared partner).

InitErgmTerm.gwdelesf *Geometrically weighted delayed edgewise shared friends*

Description

This term calculates the number of shared friends based on the previous network. It then applies a geometric transformation to these counts to reduce the influence of large counts. Specifically, if the decay parameter `decay` is provided, the weighting function used is

$$f(k; decay) = 1 - (1 - e^{-decay})^k,$$

where `k` is the count of shared partners and `decay` controls how quickly the weight decreases as counts increase. If `decay` is no

Usage

```
# binary: gwdelesf(decay, base, type = "OTP")
```

Arguments

`decay` Numeric decay parameter controlling weighting intensity. If `NULL`, raw counts are used.

`base` Character indicating which edges in the current network are used as the base: ‘"+"‘ for positive ties or ‘"-"‘ for negative ties.

`type` Character specifying which shared partner pattern to use; one of `"OTP"` (outgoing two-path), `"ITP"` (incoming two-path), `"RTP"` (reciprocated two-path), `"OSP"` (outgoing shared partner), `"ISP"` (incoming shared partner).

`InitErgmTerm.Neg` *Evaluation of negative edges*

Description

Evaluates the terms in ‘formula‘ of the negative edges and sums the results elementwise.

Usage

```
# binary: Neg(formula)
```

Arguments

`formula`	a one-sided [ergm()]-style formula with the terms to be evaluated

`InitErgmTerm.Pos` *Evaluation of positive edges*

Description

Evaluates the terms in ‘formula‘ of the positive edges and sums the results elementwise.

Usage

```
# binary: Pos(formula)
```

Arguments

`formula`	a one-sided [ergm()]-style formula with the terms to be evaluated

`mple_sign` *Fit an ERGM with MPLE using a logistic regression model*

Description

Returns a fitted logistic regression model used to calculate the maximum pseudolikelihood estimate (MPLE) of an exponential random graph model (ERGM).

Usage

```
mple_sign(formula, control = control.ergm(), seed = NULL, ...)
```

Arguments

`formula`	An ERGM formula with the network on the left-hand side.
`control`	A list of control parameters for `ergmMPLE`. By default, the covariance method is set to "Godambe".
`seed`	Optional integer to set the random seed for reproducibility when simulating networks for Godambe covariance estimation.
`...`	Additional arguments passed to `ergmMPLE`.

Details

The MPLE is calculated by first computing matrices of positive and negative change statistics. These are then used to estimate the MPLE via logistic regression. Optionally, the covariance can be estimated using the Godambe method.

Value

An object of class `ergm`.

See Also

`ergmMPLE`, `ergm`, `glm`

Examples

```
data(tribes)
mple_sign(tribes ~ Pos(~edges) + Neg(~edges))
```

`network.sign` *Create Signed Network Object*

Description

Turn adjacency matrices or edgelists into static or dynamic signed networks.

Usage

```
network.sign(
  mat = NULL,
  pos.mat = NULL,
  neg.mat = NULL,
  directed = FALSE,
  loops = FALSE,
  matrix.type = c("adjacency", "edgelist"),
  vertex.names = NULL,
  vertex.attr = NULL,
  dual.sign = FALSE,
```

```
  timepoints = NULL,
  tie.breaker = c("zero", "positive", "negative", "first", "last"),
  ...
)
```

Arguments

mat	(List of) signed adjacency matrices or edgelists. For dynamic networks, provide a list. Adjacency matrices must contain only -1, 0, or 1. Edgelists must have three columns: "From", "To", and "Sign".
pos.mat	Optional. Positive adjacency matrix or list of matrices.
neg.mat	Optional. Negative adjacency matrix or list of matrices. If provided, these are treated as two separate layers of the same network.
directed	Logical; should edges be interpreted as directed? Defaults to FALSE.
loops	Logical; should loops be allowed? Defaults to FALSE.
matrix.type	Either "adjacency" or "edgelist".
vertex.names	Optional. A vector or list of vertex names.
vertex.attr	Optional. Additional vertex attributes.
dual.sign	Logical. Allow positive and negative edges simultaneously between the same pair.
timepoints	Optional. Pooling definition for dynamic networks.
tie.breaker	How to resolve ties when pooling signed matrices.
...	Additional arguments passed to ‘network::network‘.

Value

A signed network of class ‘static.sign‘ or ‘dynamic.sign‘.

---

networks.sign — *Combine Signed Networks into a Multi- or Dynamic-Network Object*

Description

Creates a composite network object from multiple signed networks, suitable for ERGM modeling. Can represent either a multilayer or dynamic signed network structure.

Usage

```
networks.sign(..., dynamic = FALSE, dual.sign = FALSE)
```

Arguments

`...`	One or more signed networks (objects of class `"static.sign"`), or a list of such networks.
`dynamic`	Logical. If `TRUE`, treat input as a dynamic network; otherwise as a multilayer network. Defaults to `FALSE`.
`dual.sign`	Logical. If `TRUE`, disables the layer fixing constraint. Defaults to `FALSE`.

Value

A combined network object of class `"multi.sign"` or `"dynamic.sign"`, with the appropriate ERGM constraint formula.

Examples

```
data("tribes")
multi_net <- networks.sign(tribes, tribes)
dyn_net <- networks.sign(list(tribes, tribes), dynamic = TRUE)
```

---

nse-ergmTerm *Non-edgewise shared enemies*

Description

This term adds one network statistic to the model for each element in ‘d‘ where the i th such statistic equals the number of non-edges in the network with exactly ‘d[i]‘ shared enemies. For a directed network, multiple shared enemy definitions are possible.

Usage

```
# binary: nse(d, type="OTP", base=NULL, in_order=FALSE)
```

Arguments

`d`	a vector of distinct integers
`type`	A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.
`base`	specify the base of the triad, either by ’+’ and ’-’ or 1 and -1

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

`nsf-ergmTerm` *Non-edgewise shared friends*

Description

This term adds one network statistic to the model for each element in ‘d‘ where the i th such statistic equals the number of non-edges in the network with exactly ‘d[i]‘ shared friends. For a directed network, multiple shared friend definitions are possible.

Usage

```
# binary: nsf(d, type="OTP", base=NULL, in_order=FALSE)
```

Arguments

`d`	a vector of distinct integers
`type`	A string indicating the type of shared partner or path to be considered for directed networks: ‘"OTP"‘ (default for directed), ‘"ITP"‘, ‘"RTP"‘, ‘"OSP"‘, and ‘"ISP"‘; has no effect for undirected. See the section below on Shared partner types for details.
`base`	specify the base of the triad, either by '+' and '-' or 1 and -1

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the ‘type‘ argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the ‘relevent‘ package): - Outgoing Two-path (‘"OTP"‘): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner". - Incoming Two-path (‘"ITP"‘): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner" - Reciprocated Two-path (‘"RTP"‘): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$. - Outgoing Shared Partner (‘"OSP"‘): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$. - Incoming Shared Partner (‘"ISP"‘): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$. By default, outgoing two-paths (‘"OTP"‘) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see [‘options?ergm‘][ergm-options]), ‘cache.sp‘, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

[‘ergmTerm‘] for index of model terms currently visible to the package.

Keywords: directed, undirected, binary

---

plot.dynamic.sign *Visualization for Dynamic Signed Networks*

Description

`plot.dynamic.sign()` visualizes a dynamic signed network over multiple timepoints.

Usage

```
## S3 method for class 'dynamic.sign'
plot(
  x,
  col_pos = "#008000",
  col_neg = "#E3000F",
  neg.lty = 1,
  inv_weights = TRUE,
  time = NULL,
  titles = NULL,
  fix.pos = TRUE,
  ...
)
```

Arguments

`x`	A signed network object of class `dynamic.sign`.
`col_pos`	Color for positive edges. Default is 'green3'.
`col_neg`	Color for negative edges. Default is 'red3'.
`neg.lty`	Line type for negative edges. Default is "solid". Other options are "dotted" and "dashed".
`inv_weights`	Logical. If TRUE, edge weights are inverted (1/weights) so positive edges pull nodes closer together. Default is TRUE.
`time`	A vector of integers indicating which timepoints should be visualized. Defaults to all.
`titles`	A character vector of names for the timepoints.
`fix.pos`	Logical. If TRUE, the layout is fixed across timepoints based on the first timepoint. Default is TRUE.
`...`	Additional arguments passed to the plot function.

Value

A list of plots, one for each selected timepoint.

Layout

Uses a force-directed graph layout based on stress majorization, implemented in the `graphlayouts` package via `layout_with_stress()`. Similar to Kamada-Kawai, but generally faster and with better results.

`plot.static.sign` *Visualization for Signed Networks*

Description

Functions to visualize signed networks in static or dynamic form.

Usage

```
## S3 method for class 'static.sign'
plot(
  x,
  col_pos = "#008000",
  col_neg = "#E3000F",
  neg.lty = 1,
  inv_weights = TRUE,
  coord = NULL,
  ...
)
```

Arguments

`x`	A signed network object of class `static.sign`.
`col_pos`	Color for positive edges. Default is 'green3'.
`col_neg`	Color for negative edges. Default is 'red3'.
`neg.lty`	Line type for negative edges. Default is "solid". Other options are "dotted" and "dashed".
`inv_weights`	Logical. If TRUE, edge weights are inverted (1/weights) so positive edges pull nodes closer together. Default is TRUE.
`coord`	Optional matrix of coordinates for node positions. If NULL, layout is computed using stress majorization.
`...`	Additional arguments passed to the plot function.

Value

A plot of the signed network.

Layout

Uses a force-directed graph layout based on stress majorization, implemented in the `graphlayouts` package via `layout_with_stress()`. Similar to Kamada-Kawai, but generally faster and with better results.

Static signed networks

`plot.static.sign()` visualizes a single (static) signed network.

References

Gansner ER, Koren Y, North S (2004). "Graph drawing by stress majorization." In *International Symposium on Graph Drawing*, 239–250. Springer.

See Also

UnLayer, layout_with_stress

Examples

```
data("tribes")
plot(tribes, col_pos = "green", col_neg = "red")
```

rebels *Conflict Events in Syrian Civil War*

Description

A dynamic network of combat events in the Syrian civil war between 2017 and 2025. The raw data comes from the Armed Conflict Location & Event Data Project Raleigh et al. (2010).

Format

An undirected `dynamic.sign` object with no loops and eight timepoints.

References

Fritz C, Mehrl M, Thurner PW, Kauermann G (2023). “All that glitters is not gold: Relational events models with spurious events.” *Network Science*, **11**(2), 184–204., Raleigh C, Linke r, Hegre H, Karlsen J (2010). “Introducing ACLED: An armed conflict location and event dataset.” *Journal of peace research*, **47**(5), 651–660.

Examples

```
data(rebels)
```

rebels_pooled *Conflict Events in Syrian Civil War*

Description

A pooled dynamic network of combat events in the Syrian civil war between 2017 and 2019 with 4 timepoints. The raw data comes from the Armed Conflict Location & Event Data Project Raleigh et al. (2010).

Format

An undirected `dynamic.sign` object with no loops and eight timepoints.

References

Fritz C, Mehrl M, Thurner PW, Kauermann G (2023). “All that glitters is not gold: Relational events models with spurious events.” *Network Science*, **11**(2), 184–204., Raleigh C, Linke r, Hegre H, Karlsen J (2010). “Introducing ACLED: An armed conflict location and event dataset.” *Journal of peace research*, **47**(5), 651–660.

Examples

```
data(rebels)
```

snctrl *Statnet Control*

Description

A utility to facilitate argument completion of control lists, reexported from ‘statnet.common‘.

Currently recognised control parameters

This list is updated as packages are loaded and unloaded.

Package ergm:

control.ergm drop, init, init.method, main.method, force.main, main.hessian, checkpoint, resume, MPLE.samplesize, init.MPLE.samplesize, MPLE.type, MPLE.maxit, MPLE.nonvar, MPLE.nonident, MPLE.nonident.tol, MPLE.covariance.samplesize, MPLE.covariance.method, MPLE.covariance.sim.burnin, MPLE.covariance.sim.interval, MPLE.check, MPLE.constraints.ignore, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.interval, MCMC.burnin, MCMC.samplesize, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burn MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.return.stats, MCMC.runtime.traceplot, MCMC.maxedges, MCMC.addto.se, MCMC.packagenames, SAN.maxit, SAN.nsteps.times, SAN, MCMLE.termination, MCMLE.maxit, MCMLE.conv.min.pval, MCMLE.confidence, MCMLE.confidence.boost, MCMLE.confidence.boo MCMLE.confidence.boost.lag, MCMLE.NR.maxit, MCMLE.NR.reltol, obs.MCMC.mul, obs.MCMC.samplesize.m obs.MCMC.samplesize, obs.MCMC.effectiveSize, obs.MCMC.interval.mul, obs.MCMC.interval, obs.MCMC.burnin.mul, obs.MCMC.burnin, obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, obs.MCMC.impute.min_informative, obs.MCMC.impute.default_density, MCMLE.min.depfac, MCMLE.sampsize.boost.pow, MCMLE.MCMC.precision, MCMLE.MCMC.max.ESS.frac, MCMLE.metric, MCMLE.method, MCMLE.dampening, MCMLE.dampening.min.ess, MCMLE.dampening.level, MCMLE.steplength.margin, MCMLE.steplength, MCMLE.steplength.parallel, MCMLE.sequential, MCMLE.density.guard.min, MCMLE.density.guard, MCMLE.effectiveSize, obs.MCMLE.effectiveSize, MCMLE.interval, MCMLE.burnin, MCMLE.samplesize.per_theta, MCMLE.samplesize.min, MCMLE.samplesize, obs.MCMLE.samplesize.per_theta, obs.MCMLE.samplesize.min, obs.MCMLE.samplesize, obs.MCMLE.interval, obs.MCMLE.burnin, MCMLE.steplength.solver, MCMLE.last.boost, MCMLE.steplength.esteq, MCMLE.steplength.miss.sample, MCMLE.steplength.min, MCMLE.effectiveSize.interval_drop, MCMLE.save_intermediates, MCMLE.nonvar, MCMLE.nonident, MCMLE.nonident.tol, SA.phase1_n, SA.initial_gain, SA.nsubphases, SA.min_iterations, SA.max_iterations, SA.phase3_n, SA.interval, SA.burnin, SA.samplesize, CD.samplesize.per_theta, obs.CD.samplesize.per_theta, CD.nsteps, CD.multiplicity, CD.nsteps.obs, CD.multiplicity.obs, CD.maxit, CD.conv.min.pval, CD.NR.maxit, CD.NR.reltol, CD.metric, CD.method, CD.dampening, CD.dampening.min.ess, CD.dampening.level, CD.steplength.margin, CD.steplength, CD.adaptive.epsilon, CD.steplength.esteq, CD.steplength.miss.sample, CD.steplength.min, CD.steplength.parallel, CD.steplength.solver, loglik, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.ergm.bridge bridge.nsteps, bridge.target.se, bridge.bidirectional, drop, MCMC.burnin, MCMC.burnin.between, MCMC.interval, MCMC.samplesize, obs.MCMC.burnin, obs.MCMC.burnin.between, obs.MCMC.interval, obs.MCMC.samplesize, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.ergm.godfather term.options

control.ergm3 drop, init, init.method, main.method, force.main, main.hessian, checkpoint, resume, MPLE.samplesize, init.MPLE.samplesize, MPLE.type, MPLE.maxit, MPLE.nonvar, MPLE.nonident, MPLE.nonident.tol, MPLE.covariance.samplesize, MPLE.covariance.method, MPLE.covariance.sim.burnin, MPLE.covariance.sim.interval, MPLE.check, MPLE.constraints.ignore, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.interval, MCMC.burnin, MCMC.samplesize, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burn MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.return.stats, MCMC.runtime.traceplot, MCMC.maxedges, MCMC.addto.se, MCMC.packagenames, SAN.maxit, SAN.nsteps.times, SAN, MCMLE.termination, MCMLE.maxit, MCMLE.conv.min.pval, MCMLE.confidence, MCMLE.confidence.boost, MCMLE.confidence.boo MCMLE.confidence.boost.lag, MCMLE.NR.maxit, MCMLE.NR.reltol, obs.MCMC.mul, obs.MCMC.samplesize.m obs.MCMC.samplesize, obs.MCMC.effectiveSize, obs.MCMC.interval.mul, obs.MCMC.interval, obs.MCMC.burnin.mul, obs.MCMC.burnin, obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, obs.MCMC.impute.min_informative, obs.MCMC.impute.default_density, MCMLE.min.depfac, MCMLE.sampsize.boost.pow, MCMLE.MCMC.precision, MCMLE.MCMC.max.ESS.frac, MCMLE.metric, MCMLE.method, MCMLE.dampening, MCMLE.dampening.min.ess, MCMLE.dampening.level, MCMLE.steplength.margin, MCMLE.steplength, MCMLE.steplength.parallel, MCMLE.sequential, MCMLE.density.guard.min, MCMLE.density.guard, MCMLE.effectiveSize, obs.MCMLE.effectiveSize, MCMLE.interval, MCMLE.burnin, MCMLE.samplesize.per_theta, MCMLE.samplesize.min, MCMLE.samplesize, obs.MCMLE.samplesize.per_theta, obs.MCMLE.samplesize.min, obs.MCMLE.samplesize, obs.MCMLE.interval, obs.MCMLE.burnin, MCMLE.steplength.solver, MCMLE.last.boost, MCMLE.steplength.esteq, MCMLE.steplength.miss.sample, MCMLE.steplength.min, MCMLE.effectiveSize.interval_drop, MCMLE.save_intermediates, MCMLE.nonvar, MCMLE.nonident, MCMLE.nonident.tol, SA.phase1_n, SA.initial_gain, SA.nsubphases, SA.min_iterations, SA.max_iterations, SA.phase3_n, SA.interval, SA.burnin, SA.samplesize, CD.samplesize.per_theta, obs.CD.samplesize.per_theta, CD.nsteps, CD.multiplicity, CD.nsteps.obs, CD.multiplicity.obs, CD.maxit, CD.conv.min.pval, CD.NR.maxit, CD.NR.reltol, CD.metric, CD.method, CD.dampening, CD.dampening.min.ess, CD.dampening.level, CD.steplength.margin, CD.steplength, CD.adaptive.epsilon, CD.steplength.esteq, CD.steplength.miss.sample, CD.steplength.min, CD.steplength.parallel, CD.steplength.solver, loglik, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.gof.ergm nsim, MCMC.burnin, MCMC.interval, MCMC.batch, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.gof.formula nsim, MCMC.burnin, MCMC.interval, MCMC.batch, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.logLik.ergm bridge.nsteps, bridge.target.se, bridge.bidirectional, drop, MCMC.burnin, MCMC.interval, MCMC.samplesize, obs.MCMC.samplesize, obs.MCMC.interval, obs.MCMC.burnin, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.san SAN.maxit, SAN.tau, SAN.invcov, SAN.invcov.diag, SAN.nsteps.alloc, SAN.nsteps, SAN.samplesize, SAN.prop, SAN.prop.weights, SAN.prop.args, SAN.packagenames, SAN.ignore.finite.offsets, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.simulate MCMC.burnin, MCMC.interval, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.simulate.ergm MCMC.burnin, MCMC.interval, MCMC.scale, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxr MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.simulate.formula MCMC.burnin, MCMC.interval, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxr MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.simulate.formula.ergm MCMC.burnin, MCMC.interval, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxr MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

Package ergm.multi:

control.gofN nsim, obs.twostage, array.max, simulate, obs.simulate, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.gofN.ergm nsim, obs.twostage, array.max, simulate, obs.simulate, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

Package tergm:

control.simulate.formula.tergm MCMC.burnin.min, MCMC.burnin.max, MCMC.burnin.pval, MCMC.burnin.add, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.maxchanges, term.options, MCMC.packagenames

control.simulate.network MCMC.burnin.min, MCMC.burnin.max, MCMC.burnin.pval, MCMC.burnin.add, MCMC.prop.form, MCMC.prop.diss, MCMC.prop.weights.form, MCMC.prop.weights.diss, MCMC.prop.args.form, MCMC.prop.args.diss, MCMC.maxedges, MCMC.maxchanges, term.options, MCMC.packagenames control.simulate.stergm MCMC.burnin.min, MCMC.burnin.max, MCMC.burnin.pval, MCMC.burnin.add, MCMC.prop.form, MCMC.prop.diss, MCMC.prop.weights.form, MCMC.prop.weights.diss, MCMC.prop.args.form, MCMC.prop.args.diss, MCMC.maxedges, MCMC.maxchanges, term.options, MCMC.packagenames

control.simulate.tergm MCMC.burnin.min, MCMC.burnin.max, MCMC.burnin.pval, MCMC.burnin.add, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.maxchanges, term.options, MCMC.packagenames

control.stergm init.form, init.diss, init.method, force.main, MCMC.prop.form, MCMC.prop.diss, MCMC.prop.weights.form, MCMC.prop.args.form, MCMC.prop.weights.diss, MCMC.prop.args.diss, MCMC.maxedges, MCMC.maxchanges, MCMC.packagenames, CMLE.MCMC.burnin, CMLE.MCMC.interval, CMLE.ergm, CMLE.form.ergm, CMLE.diss.ergm, CMLE.NA.impute, CMLE.term.check.override, EGMME.main.method, EGMME.initialfit.control, EGMME.MCMC.burnin.min, EGMME.MCMC.burnin.max, EGMME.MCMC.burnin.pval, EGMME.MCMC.burnin.add, MCMC.burnin, MCMC.burnin.mul, SAN.maxit, SAN.nsteps.times, SAN, SA.restarts, SA.burnin, SA.plot.progress, SA.max.plot.points, SA.plot.stats, SA.init.gain, SA.gain.decay, SA.runlength, SA.interval.mul, SA.init.interval, SA.min.interval, SA.max.interval, SA.phase1.minruns, SA.phase1.tries, SA.phase1.jitter, SA.phase1.max.q, SA.phase1.backoff.rat, SA.phase2.levels.max, SA.phase2.levels.min, SA.phase2.max.mc.se, SA.phase2.repeats, SA.stepdown.maxn, SA.stepdown.p, SA.stop.p, SA.stepdown.ct, SA.phase2.backoff.rat, SA.keep.oh, SA.keep.min.runs, SA.keep.min, SA.phase2.jitter.mul, SA.phase2.maxreljump, SA.guard.mul, SA.par.eff.pow, SA.robust, SA.oh.memory, SA.refine, SA.se, SA.phase3.samplesize.runs, SA.restart.on.err, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.tergm init, init.method, force.main, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.maxchanges, MCMC.packagenames, CMLE.MCMC.burnin, CMLE.MCMC.interval, CMLE.ergm, CMLE.NA.impute, CMLE.term.check.override, EGMME.main.method, EGMME.initialfit.control EGMME.MCMC.burnin.min, EGMME.MCMC.burnin.max, EGMME.MCMC.burnin.pval, EGMME.MCMC.burnin.add, MCMC.burnin, MCMC.burnin.mul, SAN.maxit, SAN.nsteps.times, SAN, SA.restarts, SA.burnin, SA.plot.progress, SA.max.plot.points, SA.plot.stats, SA.init.gain, SA.gain.decay, SA.runlength, SA.interval.mul, SA.init.interval, SA.min.interval, SA.max.interval, SA.phase1.minruns, SA.phase1.tries, SA.phase1.jitter, SA.phase1.max.q, SA.phase1.backoff.rat, SA.phase2.levels.max, SA.phase2.levels.min, SA.phase2.max.mc.se, SA.phase2.repeats, SA.stepdown.maxn, SA.stepdown.p, SA.stop.p, SA.stepdown.ct, SA.phase2.backoff.rat, SA.keep.oh, SA.keep.min.runs, SA.keep.min, SA.phase2.jitter.mul, SA.phase2.maxreljump, SA.guard.mul, SA.par.eff.pow, SA.robust, SA.oh.memory, SA.refine, SA.se, SA.phase3.samplesize.run SA.restart.on.err, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.tergm.godfather term.options

See Also

[statnet.common::snctrl()]

---

sponsor *Common Sponsor Data for Syrian Civil War Factions*

Description

A data frame containing binary indicators for whether each faction in the Syrian civil war is sponsored by a common external actor.

Format

A matrix with 68 rows and 68 columns.

References

Fritz C, Mehrl M, Thurner PW, Kauermann G (2023). "All that glitters is not gold: Relational events models with spurious events." *Network Science*, **11**(2), 184–204.

Examples

```
data(sponsor)
```

`summary.static.sign` *Network Attributes for Signed Networks*

Description

Print descriptive statistics of a signed network.

Usage

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

## S3 method for class 'dynamic.sign'
summary(object, time = NULL, names = NULL, ...)
```

Arguments

`object`	A signed network object of class `dynamic.sign`.
`...`	Additional arguments.
`time`	Integer vector of timepoints to summarize. Defaults to all.
`names`	Character vector of names for timepoints. If NULL, uses "Time 1", "Time 2", etc.

Value

A data frame or matrix with network attributes.

Static signed networks

`summary.static.sign()` summarizes a single (static) signed network.

See Also

network.sign, UnLayer

Examples

```
data("tribes")
summary(tribes)
```

`summary_formula.dynamic.sign` *Summary formula method for dynamic signed networks*

Description

Calculates statistics for dynamic.sign objects at specified timepoints.

Usage

```
## S3 method for class 'dynamic.sign'
summary_formula(object, at, ..., basis = NULL)
```

Arguments

`object`	A formula with a dynamic.sign network as LHS.
`at`	Numeric vector of timepoints. Defaults to all if missing.
`...`	Additional arguments passed to summary_formula for network objects.
`basis`	Optional dynamic.sign network. If NULL, uses LHS network.

Value

Matrix of statistics for each timepoint.

`tribes` *Read Highland Tribes*

Description

A static network of political alliances and enmities among the 16 Gahuku-Gama sub-tribes of Eastern Central Highlands of New Guinea, documented by Read (1954).

Format

An undirected `static.sign` object with no loops.

References

Taken from UCINET IV, which cites the following: Hage P, Harary F (1983). *Structural Models in Anthropology*, Cambridge Studies in Social and Cultural Anthropology. Cambridge University Press. ISBN 9780521273114., Read KE (1954). “Cultures of the central highlands, New Guinea.” *Southwestern Journal of Anthropology*, **10**(1), 1–43.

Examples

```
data(tribes)
```

UnLayer — *Multilayer network to single layer network.*

Description

Turn a multilayer network object into a single layer network object.

Usage

```
UnLayer(net, color_pos = "#008000", color_neg = "#E3000F", neg.lty = 2)
```

Arguments

net	A signed network object of class `static.sign` or `dynamic.sign`.
color_pos	Color for positive edges. Default is '#008000'.
color_neg	Color for negative edges. Default is '#E3000F'.
neg.lty	Line type for negative edges. Default is 2.

Value

Single layer network object or a list of network objects for dynamic.sign.

See Also

network.sign

Examples

```
data("tribes")
tribes_sgl <- UnLayer(tribes)
```

Index