Package: ernm (via r-universe)

October 27, 2026

Type Package

Title Exponential-Family Random Network Models

Version 1.0.5

Date 2026-06-23

Description Estimation of fully and partially observed
Exponential-Family Random Network Models (ERNM).
Exponential-family Random Graph Models (ERGM) and Gibbs Fields
are special cases of ERNMs and can also be estimated with the
package. Please cite Fellows and Handcock (2012),
``Exponential-family Random Network Models'' available at
<doi:10.48550/arXiv.1208.0121>.

License LGPL-2.1

Depends R (>= 3.5.0), methods, network, Rcpp

Imports dplyr, ggplot2, graphics, moments, rlang, stats, tidyr, trust

Suggests spelling, testthat

LinkingTo BH, Rcpp

Copyright 2014-2025 Ian Fellows

Encoding UTF-8

Language en-US

LazyData true

LazyLoad yes

RcppModules ernm

RoxygenNote 7.3.3

Config/pak/sysreqs libicu-dev

Repository https://duncan-clark.r-universe.dev

Date/Publication 2026-06-23 16:26:08 UTC

RemoteUrl https://github.com/duncan-clark/ernm

RemoteRef HEAD

RemoteSha c5090b8f921a6590b50762271ebbd77f4d7134c7

Contents

as.network.Rcpp_UndirectedNet . . . 2
calculateStatistics . . . 4
coef.ernm . . . 5
createCppModel . . . 6
createCppSampler . . . 7
DirectedNet-class . . . 9
dutch_school . . . 9
ernm . . . 10
ernm-formula . . . 11
ernm-terms . . . 12
ernm_gof . . . 14
ernmFit . . . 15
ernmPackageSkeleton . . . 17
extract-methods . . . 17
fullErnmLikelihood . . . 19
FullErnmModel . . . 20
logLik.ernm . . . 20
marErnmLikelihood . . . 21
mcmcEss . . . 21
mcmcse . . . 22
MissingErnmModel . . . 22
plot.ernm . . . 23
print.ernm . . . 23
print.ErnmSummary . . . 24
samplike . . . 24
simulateStatistics . . . 25
summary.ernm . . . 26
taperedErnmLikelihood . . . 27
UndirectedNet-class . . . 28
vcov.ernm . . . 28

Index **29**

`as.network.Rcpp_UndirectedNet`
Convert an Rcpp_UndirectedNet to a network object

Description

Converts a native ernm network into a network object from the network package.

Converts a native ernm network into a network object from the network package.

'BinaryNet' covers ERNM's native network objects exposed via Rcpp modules: 'Rcpp_DirectedNet' and 'Rcpp_UndirectedNet'. This page documents the classes and common coercion/plot/subsetting methods.

Usage

```
## S3 method for class 'Rcpp_UndirectedNet'
as.network(x, ...)

## S3 method for class 'Rcpp_DirectedNet'
as.network(x, ...)

## S3 method for class 'Rcpp_DirectedNet'
plot(x, ...)

## S3 method for class 'Rcpp_UndirectedNet'
plot(x, ...)

as.BinaryNet(x, ...)
```

Arguments

x	the object
...	unused

Value

an undirected network object

a directed network object

No return value, invisibly NULL

No return value, invisibly NULL

either an Rcpp_UndirectedNet or Rcpp_DirectedNet object

Classes

* ‘Rcpp_DirectedNet‘ – directed binary network * ‘Rcpp_UndirectedNet‘ – undirected binary network

Coercion

* ‘as.network()‘ – convert to a ‘network‘ object (package **network**) * ‘as.BinaryNet()‘ – convert from a ‘network‘ (or return-as-is)

Methods

* ‘plot()‘ – plot via ‘plot.network‘

See Also

[network::network], [network::plot.network]

Examples

```
edge_list <- matrix(c(1,3),ncol=2)
# create a network with an edge from 1 -> 3
ernm_net <- new(UndirectedNet,edge_list,5)

# convert to a network object from the network package
network_net <- as.network(ernm_net)
network_net
edge_list <- matrix(c(1,3),ncol=2)
# create a network with an edge from 1 -> 3
ernm_net <- new(DirectedNet,edge_list,5)

# convert to a network object from the network package
network_net <- as.network(ernm_net)
network_net

# create ring network with 5 vertices
edge_list <- matrix(c(1,2,2,3,3,4,4,5,5,1),ncol=2, byrow=TRUE)
ernm_net <- new(DirectedNet,edge_list,5)

# basic plot
plot(ernm_net)

# change vertex point size (see plot.network)
plot(ernm_net, vertex.cex=.5)

# create ring network with 5 vertices
edge_list <- matrix(c(1,2,2,3,3,4,4,5,5,1),ncol=2, byrow=TRUE)
ernm_net <- new(UndirectedNet,edge_list,5)

# basic plot
plot(ernm_net)

# change vertex point size (see plot.network)
plot(ernm_net, vertex.cex=.5)

data(samplike)

# convert Sampson's monks into a native ernm network
net <- as.BinaryNet(samplike)
net[["group"]]
net[1:5,1:5]
```

`calculateStatistics` *Calculate network statistics*

Description

Calculates all network statistics

Usage

```
calculateStatistics(formula)
```

Arguments

formula	An ernm formula (see ernm-formula)

Value

a named vector of statistics

Examples

```
## Not run:
data(samplike)
calculateStatistics(samplike ~ edges() +
 nodeCount("group") +
 nodeMatch("group") +
 homophily("group") +
 triangles())

## End(Not run)
```

---

coef.ernm *Access ERNM parameters*

Description

Access ERNM parameters

Usage

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

Arguments

object	object
...	unused

Value

parameter vector

createCppModel *Creates a C++ representation of an ERNM model*

Description

Creates a C++ representation of an ERNM model

Usage

```
createCppModel(
  formula,
  ignoreMnar = TRUE,
  cloneNet = TRUE,
  theta = NULL,
  modelArgs = list(modelClass = "Model")
)
```

Arguments

formula	the model formula (see ernm-formula)
ignoreMnar	ignore missing not at random offsets
cloneNet	should the network be cloned
theta	the model parameters.
modelArgs	additional arguments for the model, e.g. tapering parameters

Value

a Model object

Examples

```
## Not run:
edge_list <- matrix(numeric(),ncol=2)
net <- new(UndirectedNet,edge_list,5)

rcpp_model <- createCppModel(net ~ edges(), theta = 0)

rcpp_sampler <- new(UndirectedMetropolisHastings, rcpp_model)

# Run MCMC to generate 30 networks with burnin=10 and an interval of 20 steps between each network
networks <- rcpp_sampler$generateSample(10,20,30)

sapply(networks, function(net) net$nEdges()) # number of edges in each network

## End(Not run)
```

createCppSampler *Create a C++ MCMC sampler*

Description

Create a C++ MCMC sampler

Usage

```
createCppSampler(
  formula,
  modelArgs = list(modelClass = "Model"),
  dyadToggle = NULL,
  dyadArgs = list(),
  vertexToggle = NULL,
  vertexArgs = list(),
  nodeSamplingPercentage = 0.2,
  ignoreMnar = TRUE,
  theta = NULL,
  ...
)
```

Arguments

formula	the model formula (see ernm-formula)
modelArgs	additional arguments for the model, e.g. tapering parameters
dyadToggle	the method of sampling to use. Defaults to alternating between nodal-tie-dyad and neighborhood toggling.
dyadArgs	list of args for dyad
vertexToggle	the method of vertex attribute sampling to use.
vertexArgs	list of args for vertex
nodeSamplingPercentage	how often the nodes should be toggled
ignoreMnar	ignore missing not at random offsets
theta	parameter values
...	additional parameters to be passed to createCppModel

Details

Available dyad toggles are:

'RandomDyad' : chooses a dyad randomly to toggle

'TieDyad' : chooses a dyad randomly with probability .5 and a random edge with probability .5

'NodeTieDyad' : chooses a random vertex and then chooses an out dyad from that vertex with probability .5 and an out edge with probability .5

'Neighborhood' : The neighborhood proposal starts by choosing a random vertex (a) and then selecting two random neighbors (b and c) Then a random non-a neighbor of b is also selected (d). The neighborhood toggle selects the dyad b- for toggling with probability 50% and the dyad d-c otherwise.

'Compound_NodeTieDyad_Neighborhood' : chooses 'NodeTieDyad' with probability .5 and 'Neighborhood' otherwise. This is the default proposal.

'Tetrad' : A toggle that preserves network degrees. This is useful when degrees are considered fixed.

'RandomMissingDyad' : RandomDyad, but restricted to missing dyads.

'NodeTieDyadMissing' : NodeTieDyad, but restricted to missing dyads.

'NeighborhoodMissing' : Neighborhood, but restricted to missing dyads.

'Compound_NodeTieDyadMissing_NeighborhoodMissing' : Compound_NodeTieDyad_Neighborhood, but restricted to missing dyads.

Available vertex toggles are:

'DefaultVertex' : The default toggle.

'DefaultVertexMissing' : DefaultVertex, but restricted to missing values.

Value

a MetropolisHastings object

Examples

```
## Not run:
edge_list <- matrix(numeric(),ncol=2)
net <- new(UndirectedNet,edge_list,5)
net[["group"]] <- c("a","b","a","b","a")

# create a simple ernm model sampler
sampler <- createCppSampler(net ~ edges() + nodeCount("group") | group, theta = c(0,0))

# generate network statistics for 10 networks with a burn in of 100 steps and 200 steps between them
sampler$generateSampleStatistics(100,200,10)

# generate a network a further 100 steps later
sampler$generateSample(0,100,1)[[1]]

# make a sampler using Tie-Dyad proposals for the graph
td_sampler <- createCppSampler(net ~ edges() + nodeCount("group") | group,
  theta = c(0,0),
  dyadToggle = "TieDyad")
td_sampler$generateSampleStatistics(100,200,10)

## End(Not run)
```

DirectedNet-class *DirectedNet class*

Description

An S4 (old-style) class representing a directed network.

dutch_school *Dutch School Data*

Description

This dataset contains network and actor attributes collected in early adolescence. It is provided by Andrea Knecht and stored in the package.

Usage

```
data(dutch_school)

data("dutch_school")
```

Format

An object of class `list` of length 4.

Data taken from https

//www.stats.ox.ac.uk/~snijders/siena/tutorial2010_data.htm. Processed as undirected networks.

Source

Knecht, A. (2004). *Network and actor attributes in early adolescence*. DANS Data Station Social Sciences and Humanities. DOI: doi:10.17026/dansz9bh2bp.

References

Snijders, T.A.B., Steglich, C.E.G., and van de Bunt, G.G. (2010), Introduction to actor-based models for network dynamics, Social Networks 32, 44-60, http://dx.doi.org/10.1016/j.socnet.2009.02.004.

ernm *Fits an ERNM model*

Description

ernm() fits exponential family random network models, which is an extension of exponential family random graph models, where nodal covariates may be considered stochastic. Additionally, the function may be used to fit models where the nodal covariates are random, but the graph is fixed (i.e. ALAAMs)

Usage

```
ernm(
  formula,
  tapered = TRUE,
  tapering_r = 3,
  modelArgs = list(),
  nodeSamplingPercentage = 0.2,
  modelType = NULL,
  likelihoodArgs = list(),
  fullToggles = c("Compound_NodeTieDyad_Neighborhood", "DefaultVertex"),
 missingToggles = c("Compound_NodeTieDyadMissing_NeighborhoodMissing", "VertexMissing"),
  ...
)
```

Arguments

formula	an ernm model formula (see ernm-formula)
tapered	should the model be tapered
tapering_r	the tapering parameter (tau = 1/(tapering_r^2 +5))
modelArgs	additional arguments for the model, e.g. tapering parameters that override the defaults
nodeSamplingPercentage	how often are nodal variates toggled
modelType	either FullErnmModel or MissingErnmModel if NULL will check for missingness
likelihoodArgs	additional arguments for the ernmLikelihood
fullToggles	a character vector of length 2 indicating the dyad and vertex toggle types for the unconditional simulations
missingToggles	a character vector of length 2 indicating the dyad and vertex toggle types for the conditional simulations
...	additional parameters for ernmFit

Value

a fitted model

References

Fellows, Ian Edward. Exponential family random network models. University of California, Los Angeles, 2012.

Examples

```
## Not run:
data(samplike)

# fit a tapered model to the samplike dataset where group is considered random
fit <- ernm(samplike ~ edges() + nodeCount("group") + nodeMatch("group") | group)
summary(fit)

# fit an untapered model. The homophily term is a degeneracy robust version
# of nodeMatch, which should be used instead of nodeMatch when tapering is not
# present. See Fellows (2012)
fit2 <- ernm(samplike ~ edges() + nodeCount("group") + homophily("group") | group, tapered=FALSE)
summary(fit2)

## End(Not run)

## Not run:
# standard ergms may be fit within ernm
library(network)
data(flo)
flomarriage <- network(flo,directed=FALSE)
fit_flo <- ernm(flomarriage ~ edges() + star(2) + triangles(), tapered=FALSE)
summary(fit_flo)

# ALAAMs can be fit by specifying that edges are considered fixed using noDyad
fit3 <- ernm(samplike ~ nodeCount("group") + nodeMatch("group") | group + noDyad)
summary(fit3)

## End(Not run)
```

`ernm-formula` *ERNM formula*

Description

ERNM formula

Formula

ERNM models are specified using a formula interface. The formula expresses what statistics are in the model, along with any parameters that the statistics take. It also defines what nodal covariates are considered random and if the edges are considered random. Finally, it specifies the network that the model is defined on.

The format of the formula follows the pattern

..network.. ~ ..statistics.. | ..random..

The ..network.. place should be a single object that can be coerced into a native ernm network using as.BinaryNet.

The ..statistics.. place may contain any statistic supported by the package (see `ernm-terms`). Multiple statistics may be added by separating them with "+".

The ..random.. place defines what is considered random within the network. By default the dyads are considered random, however, this may be changed by including noDyad in the random place. Additionally, the names of vertex attributes can be added to make them stochastic.

For example, a simple erdos-renyi random graph model can be specified on the samplike dataset (see data(samplike)) by

samplike ~ edges()

A simple multinomial ALAAM model for the group variable can be specified by setting the dyads to be fixed and group to be random.

samplike ~ nodeCount("group") | noDyad + group

A full ernm model with terms for edges, triangles, and homopily looks like

samplike ~ edges() + triangles() + nodeCount("group") + homophily("group")

`ernm-terms` *ERNM model terms*

Description

ERNM model terms

Statistic Descriptions

`edges` **(directed) (undirected)** *Edges:* This term adds one network statistic equal to the number of edges (i.e. nonzero values) in the network.

`reciprocity()` **(directed)** A count of the number of pairs of actors i and j for which $(i \to j)$ and $(j \to i)$ both exist.

`star(k, direction="in")` **(directed) (undirected)** The `k` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `k`. The ith such statistic counts the number of distinct `k[i]`-stars in the network, where a k-star is defined to be a node N and a set of k different nodes $\{O_1, \ldots, O_k\}$ such that the ties $\{N, O_i\}$ exist for $i = 1, \ldots, k$. For directed networks, direction indicates whether the count is of in-stars (direction="in") or out-stars (direction="out")

`triangles()` **(directed) (undirected)** This term adds one statistic to the model equal to the number of triangles in the network. For an undirected network, a triangle is defined to be any set $\{(i,j),(j,k),(k,i)\}$ of three edges. For a directed network, a triangle is defined as any set of three edges $(i \to j)$ and $(j \to k)$ and either $(k \to i)$ or $(k \leftarrow i)$.

`transitivity()` **(undirected)** The Soffer-Vazquez transitivity. This is clustering metric that adjusts for large degree differences and is described by C in Equation 6 of #' https://pubmed.ncbi.nlm.nih.gov/16089694/. Note The approximation of the number of possible shared neighbors between node i and j of min(d_i,d_j) - 1 in this implementation.

`nodeMatch(name)` **(directed) (undirected)** For categorical network nodal variable 'name,' the number of edges between nodes with the same variable value.

`nodeMix(name)` **(directed) (undirected)** For categorical network nodal variable 'name,' adds one statistic for each combination of levels of the variable equal to the count of edges between those levels.

`homophily(name)` **(directed) (undirected)** A degeneracy robust homophily term for use in untapered models. See Fellows (2012).

`degree(d, direction="undirected", lessThanOrEqual=FALSE)` **(directed) (undirected)** The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the ith such statistic equals the number of nodes in the network of degree `d[i]`, i.e. with exactly `d[i]` edges. For directed networks if direction="undirected" degree is counted as the sum of the in and out degrees of a node. If direction="in" then in-degrees are used and direction="out" indicates out-degrees.

If lessThanOrEqual=TRUE, then the count is the number of nodes with degree less than or equal to d.

`nodeCov(name, direction="undirected")` **(directed) (undirected)** The `name` argument is a character string giving the name of a numeric attribute in the network's vertex attribute list. This term adds a single network statistic to the model equaling the sum of `name(i)` and `name(j)` for all edges (i, j) in the network. For categorical variables, levels are coded as 1,..,nlevels'. If direction="in", only in-edges are counted. If direction="out" only out-edges are counted.

`gwesp(alpha)` **(directed) (undirected)** This term is just like `gwdsp` except it adds a statistic equal to the geometrically weighted *edgewise* (not dyadwise) shared partner distribution with decay parameter `alpha` parameter, which should be non-negative.

`gwdegree(alpha, direction="undirected")` **(directed) (undirected)** This term adds one network statistic to the model equal to the weighted degree distribution with decay controlled by the `decay` parameter. The `alpha` parameter is the same as theta_s in equation (14) in Hunter (2007).

For directed networks if direction="undirected" degree is counted as the sum of the in and out degrees of a node. If direction="in" then in-degrees are used ans direction="out" indicates out-degrees.

`gwdsp(alpha)` **(directed) (undirected)** This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution with decay parameter `decay` parameter, which should be non-negative.

`esp(d, type=2)` **(directed) (undirected)** This term adds one network statistic to the model for each element in `d` where the ith such statistic equals the number of *edges* (rather than dyads) in the network with exactly `d[i]` shared partners. This term can be used with directed and undirected networks. For directed networks the count depends on type:

type = 1 : from -> to -> nbr -> from

type = 2 : from -> to <- nbr <- from (homogeneous)

type = 3 : either type 1 or 2

type = 4 : all combinations of from -> to <-> nbr <-> from

`geoDist(long, lat, distCuts=Inf)` **(undirected)** given nodal variables for longitude and latitude, calculates the sum of the great circle distance between connected nodes. distCuts splits this into separate statistics that count the sum of the minimum of the cut point and the distance.

References

Fellows, Ian Edward. Exponential family random network models. University of California, Los Angeles, 2012.

`ernm_gof` *Goodness of fit for ERNM model*

Description

Goodness of fit plot for ERNM models, particularly suited for comparing models

Usage

```
ernm_gof(
  models,
  observed_network = NULL,
  stats_formula,
  style = "histogram",
  scales = "fixed",
  print = TRUE,
  n_sim = 10000,
  burnin = 10000,
  interval = 100,
  hist_bins = NULL,
  title = ""
)
```

Arguments

`models`	named list of ernm models to be to be compared (can be length 1
`observed_network`	the observed network
`stats_formula`	the formula for the statistics (see `ernm-formula`)
`style`	the style of the plot, either 'histogram' or 'boxplot'
`scales`	the scales of the plot, either 'fixed' or 'free'
`print`	whether to print the plot
`n_sim`	the number of simulations to run
`burnin`	the burnin for the MCMC simulation
`interval`	the sampling interval for MCMC simulation
`hist_bins`	number of bins for histogram; if `NULL`, integer statistics use unit-width bins
`title`	optional plot title

Details

Goodness of fit in ERNM is done by comparing simulated networks from the ernm model to the observed network. If the observed network is typical of the simulated networks it is considered to be well fit.

Value

A list containing goodness-of-fit plots and simulated statistics

Examples

```
## Not run:
data(samplike)
fit_basic <- ernm(samplike ~ edges() + nodeCount("group") + nodeMatch("group") | group)
fit_tri <- ernm(samplike ~ edges() + nodeCount("group") + nodeMatch("group") + triangles() | group)

# how well is the triangle term fit?
gof <- ernm_gof(
  list(
    basic = fit_basic,
    with_triangles = fit_tri
  ),
  observed_network = samplike,
  stats_formula = samplike ~ triangles(),
  n_sim = 100
)

# look at the fit over all edgewise shared partners
gof <- ernm_gof(
  list(
    basic = fit_basic,
    with_triangles = fit_tri
  ),
  style="boxplot",
  observed_network = samplike,
  stats_formula = samplike ~ esp(1:10),
  n_sim = 100
)

## End(Not run)
```

---

ernmFit *Fit an ernm*

Description

This is a lower level MCMC-MLE fitting function for ERNM. Users should generally use the ernm() function instead.

Usage

```
ernmFit(
  sampler,
  theta0,
  mcmcBurnIn = 10000,
  mcmcInterval = 100,
  mcmcSampleSize = 10000,
  minIter = 3,
  maxIter = 40,
  objectiveTolerance = 0.5,
  gradTolerance = 0.25,
  meanStats,
  verbose = 1,
  method = c("bounded", "newton")
)
```

Arguments

sampler	the ErnmModel
theta0	initial starting values
mcmcBurnIn	MCMC burn in
mcmcInterval	MCMC interval
mcmcSampleSize	MCMC sample size
minIter	minimum number of MCMC-MLE iterations
maxIter	maximum number of MCMC-MLE iterations
objectiveTolerance	convergence criteria on change in log likelihood ratio
gradTolerance	convergence criteria on scaled gradient
meanStats	optional target statistics for the mean value parameters
verbose	level of verbosity 0, 1, or 2
method	the optimization method to use. "bounded" uses trust regions around the MCMC sample and is generally preferable. See Fellows (2012) for details.

Value

an ernm object

References

Fellows, Ian Edward. Exponential family random network models. University of California, Los Angeles, 2012.

ernmPackageSkeleton *Create an ERNM package skeleton*

Description

Creates a skeleton for a package extending the ernm package by copying an example package.

Usage

```
ernmPackageSkeleton(path = ".")
```

Arguments

path	A character string specifying the directory where the package skeleton will be created.

Value

A logical value indicating whether the copy was successful.

extract-methods *Subsetting and assignment for ernm network objects*

Description

These methods allow standard subsetting (‘[‘) and assignment (‘[<-‘) for ‘Rcpp_DirectedNet‘ and ‘Rcpp_UndirectedNet‘ objects.

Usage

```
## S4 method for signature 'Rcpp_DirectedNet'
x[i, j, ..., maskMissing = TRUE, drop = TRUE]

## S4 method for signature 'Rcpp_UndirectedNet'
x[i, j, ..., maskMissing = TRUE, drop = TRUE]

## S4 replacement method for signature 'Rcpp_DirectedNet'
x[i, j, ...] <- value

## S4 replacement method for signature 'Rcpp_UndirectedNet'
x[i, j, ...] <- value
```

Arguments

x	an 'Rcpp_DirectedNet' or 'Rcpp_UndirectedNet' object.
i, j	index vectors.
...	currently unused.
maskMissing	Logical. Should missing values be masked by NA?
drop	Ignored (present for compatibility).
value	Values to assign (for '[<-' only).

Value

A modified object or extracted submatrix depending on the method.

Examples

```
# convert the Sampson's monks network into a native ernm network
data(samplike)
sampnet <- as.BinaryNet(samplike)
sampnet

# get the number of nodes and edges in the network
sampnet$size()
sampnet$nEdges()

# Extract and assign vertex attributes
sampnet[["group"]]
sampnet[["newvar"]] <- rnorm(18)
sampnet[["newvar"]]

# get the edge matrix between the first 5 vertices
sampnet[1:5,1:5]

# add an edge 2 --> 3
sampnet[2,3] <- TRUE

# Make the dyad 4 --> 1 missing
sampnet[4,1] <- NA
sampnet[1:5,1:5]

# get the in and out degrees for each vertex
sampnet$inDegree(1:18)
sampnet$outDegree(1:18)
```

`fullErnmLikelihood` *Likelihood for a fully observed ernm*

Description

This function approximates the likelihood around a sample generated at parameters theta0. The likelihood is only "trusted" within a vicinity of theta0. The size of this vicinity is controlled by requiring a minimum effective sample size (minESS) at theta + (theta - tehta0) * damping.

Usage

```
fullErnmLikelihood(
  theta,
  sample,
  theta0,
  stats,
  minEss = 5,
  damping = 0.05,
  method = c("cumulant", "sample"),
  order = 3
)
```

Arguments

`theta`	parameters
`sample`	mcmc sample
`theta0`	parameter values which generated sample
`stats`	observed statistics
`minEss`	minimum effective sample size
`damping`	a damping parameter
`method`	the method of partition function approximation to use
`order`	the order of the cumulant approximation

Value

a list with value, gradient, and hessian

FullErnmModel *creates an ERNM likelihood model*

Description

creates an ERNM likelihood model

Usage

```
FullErnmModel(sampler, logLik, ...)
```

Arguments

sampler	a sampler
logLik	a log likelihood function (optional)
...	additional parameters for the log likelihood

Value

a FullyObservedModel object

logLik.ernm *MCMC approximate log-likelihood*

Description

MCMC approximate log-likelihood

Usage

```
## S3 method for class 'ernm'
logLik(object, ...)
```

Arguments

object	an ernm object
...	unused

Examples

```
## Not run:
fit <- ernm(samplike ~ edges() + nodeCount("group") + nodeMatch("group") | group)
logLik(fit)
AIC(fit)
BIC(fit)

## End(Not run)
```

marErnmLikelihood *Likelihood for an ernm with missing data*

Description

This function approximates the likelihood around a sample generated at parameters theta0. The likelihood is only "trusted" within a vicinity of theta0. The size of this vicinity is controlled by requiring a minimum effective sample size (minESS) at theta + (theta - tehta0) * damping.

Usage

```
marErnmLikelihood(theta, sample, theta0, stats, minEss = 5, damping = 0.1)
```

Arguments

theta	parameters
sample	mcmc sample
theta0	parameter values which generated sample
stats	observed statistics
minEss	minimum effective sample size
damping	a damping parameter

Value

a list with value, gradient, and hessian

mcmcEss *MCMC effective sample size*

Description

Computes the effective sample size from a statistic vector.

Usage

```
mcmcEss(x)
```

Arguments

x	A numeric vector.

Value

A numeric value representing the effective sample size.

References

Kass, R. E., Carlin, B. P., Gelman, A., & Neal, R. M. (1998). "Markov Chain Monte Carlo in Practice: A Roundtable Discussion." *The American Statistician*, 52(2), 93-100. DOI: doi:10.2307/2685466

`mcmcse` *MCMC standard error by batch*

Description

Computes the MCMC standard error from a statistic vector using a batching method.

Usage

```
mcmcse(x, expon = 0.5)
```

Arguments

`x`	A numeric vector of statistics.
`expon`	A numeric value controlling the batch size; default is 0.5.

Value

A numeric value representing the estimated standard error.

Examples

```
x <-
```

`MissingErnmModel` *Creates an ERNM likelihood model with missing data*

Description

Creates an ERNM likelihood model with missing data

Usage

```
MissingErnmModel(observedSampler, unobservedSampler, ...)
```

Arguments

observedSampler	a C++ sampler
unobservedSampler	a C++ sampler conditional upon the observed values
...	additional parameters for the log likelihood

Value

a MarModel object

---

plot.ernm *Plot an ernm object*

Description

Plot an ernm object

Usage

```
## S3 method for class 'ernm'
plot(x, ...)
```

Arguments

x	the object
...	passed to plot function

Value

No return value, plots the likelihood history

---

print.ernm *Print ernm object*

Description

Print ernm object

Usage

```
## S3 method for class 'ernm'
print(x, rowwise = TRUE, ...)
```

Arguments

x	x
rowwise	if TRUE, print mean values row-wise, otherwise column-wise
...	unused

Value

No return value, prints summary

---

print.ErnmSummary *Print a ERNM summary object*

Description

Print a ERNM summary object

Usage

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

Arguments

x	the object
...	parameters passed to print.data.frame

Value

x invisibly

---

samplike *Sampson's Monks Data*

Description

This dataset represents the social network of relationships among monks in a monastery, as studied by Samuel F. Sampson. The data were collected during a period of instability and document both positive and negative interactions among the monks.

Usage

```
data(samplike)

data("samplike")
```

Format

An object of class `network` of length 5.

Details

The study recorded friendships, antagonisms, and other social relationships among the monks before and after a significant schism occurred in the monastery.

NOTE COPIED FROM ERGM PACKAGE

Mislabeling in Versions Prior to 3.6.1: In `ergm` version 3.6.0 and earlier, the adjacency matrices of the datasets reflected an older ordering of the names.

Source

Sampson, S. F. (1969). *Crisis in a cloister*. Unpublished Ph.D. dissertation, Cornell University.

References

White, H.C., Boorman, S.A. and Breiger, R.L. (1976). *Social structure from multiple networks. I. Blockmodels of roles and positions.* American Journal of Sociology, 81(4), 730-780.

See Also

florentine, network, plot.network, ergm

`simulateStatistics` *Simulate statistics*

Description

Generates a MCMC chain for an ernm model and returns the sample statistics

Usage

```
simulateStatistics(
  formula,
  theta,
  nodeSamplingPercentage = 0.2,
  mcmcBurnIn = 10000,
  mcmcInterval = 100,
  mcmcSampleSize = 100,
  ignoreMnar = TRUE,
  modelArgs = list(modelClass = "Model"),
  ...
)
```

Arguments

formula	the model formula (see ernm-formula)
theta	model parameters
nodeSamplingPercentage	how often the nodes should be toggled
mcmcBurnIn	burn in
mcmcInterval	interval
mcmcSampleSize	sample size
ignoreMnar	ignore missing not at random offsets
modelArgs	additional arguments for the model, e.g. tapering parameters
...	additional arguments to createCppSampler

Value

a matrix of statistics

Examples

```
## Not run:
edge_list <- matrix(numeric(),ncol=2)
net <- new(UndirectedNet,edge_list,5)
net[["group"]] <- c("a","b","a","b","a")

# generate sample statistics from a simple ernm model with positive homophily
stats <- simulateStatistics(net ~ edges() + nodeCount("group") + homophily("group") | group,
  theta = c(0,0,2),
  mcmcBurnIn=10)
colMeans(stats)

## End(Not run)
```

---

summary.ernm *Summary for ernm object*

Description

Summary for ernm object

Usage

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

Arguments

object	object
...	unused

Value

an ErnmSummary object

`taperedErnmLikelihood` *Ernm likelihood for a TaperedModel*

Description

This function approximates the likelihood around a sample generated at parameters theta0. The likelihood is only "trusted" within a vicinity of theta0. The size of this vicinity is controlled by requiring a minimum effective sample size (minESS) at theta + (theta - tehta0) * damping.

Usage

```
taperedErnmLikelihood(
  theta,
  centers,
  tau,
  sample,
  theta0,
  stats,
  minEss = 5,
  damping = 0.05
)
```

Arguments

`theta`	parameters
`centers`	center of statistics
`tau`	tapering parameter
`sample`	mcmc sample
`theta0`	parameter values which generated sample
`stats`	observed statistics
`minEss`	minimum effective sample size
`damping`	a damping parameter

Value

a list with value, gradient, and hessian

UndirectedNet-class *UndirectedNet class*

Description

An S4 (old-style) class representing an undirected network.

vcov.ernm *Parameter covariance matrix*

Description

Parameter covariance matrix

Usage

```
## S3 method for class 'ernm'
vcov(object, ...)
```

Arguments

object	object
...	unused

Value

covariance matrix

Index

∗ **datasets**
    dutch_school, 9
    samplike, 24
[,DirectedNet-method *(*extract-methods*)*, 17
[,Rcpp_DirectedNet-method *(*extract-methods*)*, 17
[,Rcpp_UndirectedNet-method *(*extract-methods*)*, 17
[,UndirectedNet-method *(*extract-methods*)*, 17
[<-,DirectedNet-method *(*extract-methods*)*, 17
[<-,Rcpp_DirectedNet-method *(*extract-methods*)*, 17
[<-,Rcpp_UndirectedNet-method *(*extract-methods*)*, 17
[<-,UndirectedNet-method *(*extract-methods*)*, 17

as.BinaryNet *(*as.network.Rcpp_UndirectedNet*)*, 2
as.network.Rcpp_DirectedNet *(*as.network.Rcpp_UndirectedNet*)*, 2
as.network.Rcpp_UndirectedNet, 2

BinaryNet *(*as.network.Rcpp_UndirectedNet*)*, 2

calculateStatistics, 4
coef.ernm, 5
createCppModel, 6
createCppSampler, 7

DirectedNet *(*as.network.Rcpp_UndirectedNet*)*, 2
DirectedNet-class, 9
dutch_school, 9

ernm, 10
ernm-formula, 11
ernm-terms, 12
ernm_gof, 14
ernmFit, 15
ernmPackageSkeleton, 17
extract-methods, 17

fullErnmLikelihood, 19
FullErnmModel, 20

logLik.ernm, 20

marErnmLikelihood, 21
mcmcEss, 21
mcmcse, 22
MissingErnmModel, 22

plot.ernm, 23
plot.Rcpp_DirectedNet *(*as.network.Rcpp_UndirectedNet*)*, 2
plot.Rcpp_UndirectedNet *(*as.network.Rcpp_UndirectedNet*)*, 2
print.ernm, 23
print.ErnmSummary, 24

Rcpp_DirectedNet-class *(*as.network.Rcpp_UndirectedNet*)*, 2
Rcpp_UndirectedNet-class *(*as.network.Rcpp_UndirectedNet*)*, 2

samplike, 24
sampson *(*samplike*)*, 24
simulateStatistics, 25

summary.ernm, 26

taperedErnmLikelihood, 27

UndirectedNet
(as.network.Rcpp_UndirectedNet), 2
UndirectedNet-class, 28

vcov.ernm, 28