# Package 'ERPM'

July 21, 2025

**Type** Package

**Title** Exponential Random Partition Models

**Version** 0.2.0

**Date** 2024-05-03

**Description** Simulates and estimates the Exponential Random Partition Model presented in the paper Hoffman, Block, and Snijders (2023) <doi:10.1177/00811750221145166>. It can also be used to estimate longitudinal partitions, following the model proposed in Hoffman and Chabot (2023) <doi:10.1016/j.socnet.2023.04.002>. The model is an exponential family distribution on the space of partitions (sets of non-overlapping groups) and is called in reference to the Exponential Random Graph Models (ERGM) for networks.

**License** GPL (>= 3)

**Depends** R (>= 4.2)

**Imports** numbers, utils, stats, igraph, RColorBrewer, snowfall

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.1

**Collate** 'erpm-package.R' 'functions_utility.R' 'functions_Metropolis.R' 'functions_burninthining.R' 'functions_change_statistics.R' 'functions_estimate.R' 'functions_exactcalculations.R' 'functions_exchange_algorithm.R' 'functions_loglikelihood.R' 'functions_output.R' 'functions_phase1.R' 'functions_phase2.R' 'functions_phase3.R' 'functions_statistics.R' 'functions_visualisation.R' 'outcomeObjects.R'

**URL** https://github.com/stocnet/ERPM

**BugReports** https://github.com/stocnet/ERPM/issues

**NeedsCompilation** no

**Author** Marion Hoffman [cre, aut, cph] (ORCID:
<https://orcid.org/0000-0002-0741-7760>),
Alexandra Amani [aut],
Nico Keiser [aut]

**Maintainer** Marion Hoffman <marion.hoffman.31@gmail.com>

**Repository** CRAN

**Date/Publication** 2024-05-10 17:53:16 UTC

# Contents

---

`Bell_constraints` *Function to calculate the number of partitions with groups of sizes between smin and smax*

---

## Description

Function to calculate the number of partitions with groups of sizes between smin and smax

## Usage

```
Bell_constraints(n, smin, smax)
```

## Arguments

| | |
|---|---|
| `n` | number of nodes |
| `smin` | minimum group size possible in the partition |
| `smax` | minimum group size possible in the partition |

## Value

a numeric

## Examples

```
n <- 6
size_min <- 2
size_max <- 4
Bell_constraints(n,size_min,size_max)
```

## calculate_denominator_Dirichlet_restricted

*Calculate Dirichlet denominator*

### Description

Recursive function to calculate the denominator for the model with a single statistic for the number of groups and a given parameter value. The set of possible partitions can be restricted to partitions with groups of a certain size.

### Usage

```
calculate_denominator_Dirichlet_restricted(n, smin, smax, alpha, results)
```

### Arguments

| | |
|---|---|
| n | number of nodes |
| smin | minimum size for a group |
| smax | maximum size for a group |
| alpha | parameter value |
| results | a list |

### Value

a numeric

## calculate_proba_Dirichlet_restricted

*Calculate Dirichlet probability*

### Description

Calculate the probability of observing a partition with a given number of groups for a model with a single statistic for the number of groups and a given parameter value. The set of possible partitions can be restricted to partitions with groups of a certain size.

### Usage

```
calculate_proba_Dirichlet_restricted(alpha, stat, n, smin, smax)
```

### Arguments

| | |
|---|---|
| alpha | parameter value |
| stat | observed stat (number of groups) |
| n | number of nodes |
| smin | minimum size for a group |
| smax | maximum size for a group |

### Value

a numeric

---

`check_sizes` *Function to determine whether a partition contains the allowed group sizes*

---

### Description

Function to determine whether a partition contains the allowed group sizes

### Usage

```
check_sizes(partition, sizes.allowed, numgroups.allowed)
```

### Arguments

| | |
|---|---|
| partition | observed partition |
| sizes.allowed | vector containing possible group sizes in the partition |
| numgroups.allowed | vector containing possible number of groups in the partition |

### Value

boolean

`computeStatistics` *Compute Statistics*

## Description

Function that computes the statistic vector for a given partition and a given model

## Usage

```
computeStatistics(partition, nodes, effects, objects)
```

## Arguments

| | |
|---|---|
| `partition` | vector, A partition |
| `nodes` | data frame, Node set |
| `effects` | list with a vector "names", and a vector "objects", Effects/sufficient statistics |
| `objects` | list with a vector "name", and a vector "object", Objects used for statistics calculation |

## Value

the statistics

`computeStatistics_multiple` *Compute Statistics multiple*

## Description

Function that computes the statistic vector for given (multiple) partitions and a given model

## Usage

```
computeStatistics_multiple(
  partitions,
  presence.tables,
  nodes,
  effects,
  objects,
  single.obs = NULL
)
```

### Arguments

| | |
|---|---|
| partitions | Observed partitions |
| presence.tables | to indicate which nodes were present when |
| nodes | Node set (data frame) |
| effects | Effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | Objects used for statistics calculation (list with a vector "name", and a vector "object") |
| single.obs | equal NULL by default |

### Value

A list

---

## compute_averagesize *Compute the average size of a random partition*

---

### Description

Recursive function to compute the average size of a random partition for a given number of nodes

### Usage

```
compute_averagesize(num.nodes)
```

### Arguments

| | |
|---|---|
| num.nodes | number of nodes |

### Value

a numeric

### Examples

```
n <- 6
compute_averagesize(n)
```

## compute_numgroups_denominator

*Compute denominator for model with number of groups*

### Description

Recursive function to compute the value of the denominator for the model with a single statistic which is the number of groups

### Usage

```
compute_numgroups_denominator(num.nodes, alpha)
```

### Arguments

| | |
|---|---|
| num.nodes | number of nodes |
| alpha | parameter value |

### Value

a numeric

## correlation_between

*Between groups correlation*

### Description

This function computes the correlation between the group averages of the two attributes.

### Usage

```
correlation_between(partition, attribute1, attribute2)
```

### Arguments

| | |
|---|---|
| partition | A partition (vector) |
| attribute1 | A vector containing the values of the first attribute |
| attribute2 | A vector containing the values of the second attribute |

### Value

A number corresponding to the correlation coefficient

## Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
at <- c(3,5,23,2,1,0,3,9,2)
at2 <- c(3,5,20,2,1,0,0,9,0)
correlation_between(p,at,at2)
```

---

`correlation_within` *Within groups correlation*

---

## Description

This function computes the correlation between the two attributes for individuals in the same group.

## Usage

```
correlation_within(partition, attribute1, attribute2, group)
```

## Arguments

| | |
|---|---|
| `partition` | A partition (vector) |
| `attribute1` | A vector containing the values of the first attribute |
| `attribute2` | A vector containing the values of the second attribute |
| `group` | A number indicating the selected group |

## Value

A number corresponding to the correlation coefficient

## Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
at <- c(3,5,23,2,1,0,3,9,2)
at2 <- c(3,5,20,2,1,0,0,9,0)
correlation_within(p,at,at2,4)
```

## correlation_with_size *Correlation with size*

### Description

This function computes the correlation between an attribute and the size of the groups.

### Usage

```
correlation_with_size(partition, attribute, categorical)
```

### Arguments

| | |
|---|---|
| partition | A partition (vector) |
| attribute | A vector containing the values of the attribute |
| categorical | A Boolean (True or False) indicating if the attribute is categorical |

### Value

A number corresponding to the correlation coefficient if the attribute is numerical or the correlation ratio if the attribute is categorical.

### Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
at <- c(3,5,23,2,1,0,3,9,2)
correlation_with_size(p,at,categorical=FALSE)
```

## count_classes *Function to count the number of partitions with a certain group size structure, for all possible group size structure. Function to use after calling the "find_all_partitions" function.*

### Description

Function to count the number of partitions with a certain group size structure, for all possible group size structure. Function to use after calling the "find_all_partitions" function.

### Usage

```
count_classes(allpartitions)
```

### Arguments

| | |
|---|---|
| allpartitions | matrix containing all possible partitions for a nodeset |

## Value

integer(number of partitions with different group structures)

## Examples

```
#find partitions first
n <- 6
all_partitions <- find_all_partitions(n)
# count classes
counts_partition_classes <- count_classes(all_partitions)
```

---

CUP *CUP*

---

## Description

This function tests a partition statistic against a "conditional uniform partition null hypothesi: It compares a statistic computed on an observed partition and the same statistic computed on a set of permuted partition (partitions with the same group structure as the observed partition, with nodes being permuted).

## Usage

```
CUP(observation, fun, permutations = NULL, num.permutations = 1000)
```

## Arguments

| | |
|---|---|
| observation | A vector giving the observed partition |
| fun | A function used to compute a given partition statistic to be computed |
| permutations | A matrix, whose lines contain partitions which are permutations of the observed partition. This argument is NULL by default (in that case, the permutations are created automatically). |
| num.permutations | An integer indicating the number of permutations to generate, if they are not already given. 1000 permutations are generated by default. |

## Details

This test is similar to Conditional Uniform Graph tests in networks (we translate this into Condtional Uniform Partition tests).

## Value

The value of the statistic calculated for the observed partition, the mean value of the statistic among permuted partitions, the standard deviation of the statistic among permuted partitions, the proportion of permutation below the observed statistic, the proportion of permutation above the observed statistic, the lower boundary of the 95% CI, the upper boundary of the 95% CI

## Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
at <- c(0,1,1,1,1,0,0,0,0)
CUP(p,fun=function(x){same_pairs(x,at,'avg_pergroup')})
```

---

`draw_Metropolis_multiple` *Draw Metropolis multiple*

---

## Description

Function to sample the model with a Markov chain (single partition procedure).

## Usage

```
draw_Metropolis_multiple(
  theta,
  first.partitions,
  presence.tables,
  nodes,
  effects,
  objects,
  burnin,
  thining,
  num.steps,
  neighborhood = c(0.7, 0.3, 0),
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  return.all.partitions = FALSE,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `theta` | model parameters |
| `first.partitions` | starting partition for the Markov chain |
| `presence.tables` | matrix indicating which actors were present for each observations (mandatory) |
| `nodes` | node set (data frame) |
| `effects` | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| `objects` | objects used for statistics calculation (list with a vector "name", and a vector "object") |

| | |
|---|---|
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |
| num.steps | number of samples |
| neighborhood | = c(0.7,0.3,0), way of choosing partitions: probability vector (2 actors swap, merge/division, single actor move, single pair move, 2 pairs swap, 2 groups reshuffle) |
| numgroups.allowed | = NULL, # vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | = NULL, # vector containing the number of groups simulated |
| sizes.allowed | = NULL, vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | = NULL, vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| return.all.partitions | = FALSE, option to return the sampled partitions on top of their statistics (for GOF) |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

A list

## Examples

```
# define an arbitrary set of n = 6 nodes with attributes, and an arbitrary covariate matrix
n <- 6
nodes <- data.frame(label = c("A","B","C","D","E","F"),
                    gender = c(1,1,2,1,2,2),
                    age = c(20,22,25,30,30,31))
friendship <- matrix(c(0, 1, 1, 1, 0, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 0, 0,
                       0, 1, 1, 0, 0, 1,
                       0, 0, 0, 0, 1, 0), 6, 6, TRUE)

# specify whether nodes are present at different points of time
presence.tables <- matrix(c(1, 1, 1, 1, 1, 1,
                            0, 1, 1, 1, 1, 1,
                            1, 0, 1, 1, 1, 1), 6, 3)

# choose effects to be included in the estimated model
effects_multiple <- list(names = c("num_groups","same","diff","tie","inertia_1"),
                objects = c("partitions","gender","age","friendship","partitions"),
```

```
                objects2 = c("","","","",""))
objects_multiple <- list()
objects_multiple[[1]] <- list(name = "friendship", object = friendship)

# set parameter values for each of these effects
parameters <- c(-0.2,0.2,-0.1,0.5,1)

# set a starting point for the simulation
first.partitions <- matrix(c(1, 1, 2, 2, 2, 3,
                             NA, 1, 1, 2, 2, 2,
                             1, NA, 2, 3, 3, 1), 6, 3)


# generate the simulated sample
nsteps <- 50
sample <- draw_Metropolis_multiple(theta = parameters,
                                   first.partitions = first.partitions,
                                   nodes = nodes,
                                   presence.tables = presence.tables,
                                   effects = effects_multiple,
                                   objects = objects_multiple,
                                   burnin = 100,
                                   thining = 100,
                                   num.steps = nsteps,
                                   neighborhood = c(0,1,0),
                                   numgroups.allowed = 1:n,
                                   numgroups.simulated = 1:n,
                                   sizes.allowed = 1:n,
                                   sizes.simulated = 1:n,
                                   return.all.partitions = TRUE)
```

---

`draw_Metropolis_single` *Draw Metropolis single*

---

## Description

Function to sample the model with a Markov chain (single partition procedure).

## Usage

```
draw_Metropolis_single(
  theta,
  first.partition,
  nodes,
  effects,
  objects,
```

```
  burnin,
  thining,
  num.steps,
  neighborhood = c(0.7, 0.3, 0),
  numgroups.allowed = NULL,
  numgroups.simulated = NULL,
  sizes.allowed = NULL,
  sizes.simulated = NULL,
  return.all.partitions = FALSE
)
```

## Arguments

| | |
|---|---|
| theta | model parameters |
| first.partition | starting partition for the Markov chain |
| nodes | nodeset (data frame) |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |
| num.steps | number of samples |
| neighborhood | = c(0.7,0.3,0), way of choosing partitions: probability vector (2 actors swap, merge/division, single actor move, single pair move, 2 pairs swap, 2 groups reshuffle) |
| numgroups.allowed | = NULL, # vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | = NULL, # vector containing the number of groups simulated |
| sizes.allowed | = NULL, vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | = NULL, vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| return.all.partitions | = FALSE option to return the sampled partitions on top of their statistics (for GOF) |

## Value

A list

## Examples

```
# define an arbitrary set of n = 6 nodes with attributes, and an arbitrary covariate matrix
n <- 6
nodes <- data.frame(label = c("A","B","C","D","E","F"),
                    gender = c(1,1,2,1,2,2),
                    age = c(20,22,25,30,30,31))
friendship <- matrix(c(0, 1, 1, 1, 0, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 0, 0,
                       0, 1, 1, 0, 0, 1,
                       0, 0, 0, 0, 1, 0), 6, 6, TRUE)

# choose the effects to be included (see manual for all effect names)
effects <- list(names = c("num_groups","same","diff","tie"),
objects = c("partition","gender","age","friendship"))
objects <- list()
objects[[1]] <- list(name = "friendship", object = friendship)

# set parameter values for each of these effects
parameters <- c(-0.2, 0.2, -0.1, 0.5)


# generate simulated sample, by setting the desired additional parameters for the
# Metropolis sampler and choosing a starting point for the chain (first.partition)
nsteps <- 100
sample <- draw_Metropolis_single(theta = parameters,
                                 first.partition = c(1,1,2,2,3,3),
                                 nodes = nodes,
                                 effects = effects,
                                 objects = objects,
                                 burnin = 100,
                                 thining = 10,
                                 num.steps = nsteps,
                                 neighborhood = c(0,1,0),
                                 numgroups.allowed = 1:n,
                                 numgroups.simulated = 1:n,
                                 sizes.allowed = 1:n,
                                 sizes.simulated = 1:n,
                                 return.all.partitions = TRUE)


# or: simulate an estimated model
partition <- c(1,1,2,2,2,3) # the partition already defined for the (previous) estimation
nsimulations <- 1000
simulations <- draw_Metropolis_single(theta = estimation$results$est,
                                      first.partition = partition,
                                      nodes = nodes,
                                      effects = effects,
                                      objects = objects,
                                      burnin = 100,
                                      thining = 20,
```

```
                                    num.steps = nsimulations,
                                    neighborhood = c(0,1,0),
                                    sizes.allowed = 1:n,
                                    sizes.simulated = 1:n,
                                    return.all.partitions = TRUE)
```

---

`estimate_ERPM` *Estimate ERPM*

---

## Description

Function to estimate a given model for a given observed partition. All options of the algorithm can be specified here.

## Usage

```
estimate_ERPM(
  partition,
  nodes,
  objects,
  effects,
  startingestimates,
  gainfactor = 0.1,
  a.scaling = 0.8,
  r.truncation.p1 = -1,
  r.truncation.p2 = -1,
  burnin = 30,
  thining = 10,
  length.p1 = 100,
  min.iter.p2 = NULL,
  max.iter.p2 = NULL,
  multiplication.iter.p2 = 100,
  num.steps.p2 = 6,
  length.p3 = 1000,
  neighborhood = c(0.7, 0.3, 0),
  fixed.estimates = NULL,
  numgroups.allowed = NULL,
  numgroups.simulated = NULL,
  sizes.allowed = NULL,
  sizes.simulated = NULL,
  double.averaging = FALSE,
  inv.zcov = NULL,
  inv.scaling = NULL,
  parallel = FALSE,
  parallel2 = FALSE,
  cpus = 1,
```

```
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| partition | observed partition |
| nodes | nodeset (data frame) |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| startingestimates | first guess for the model parameters |
| gainfactor | numeric used to decrease the size of steps made in the Newton optimization |
| a.scaling | numeric used to reduce the influence of non-diagonal elements in the scaling matrix (for stability) |
| r.truncation.p1 | numeric used to limit extreme values in the covariance matrix (for stability) |
| r.truncation.p2 | numeric used to limit extreme values in the covariance matrix (for stability) |
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |
| length.p1 | number of samples in phase 1 |
| min.iter.p2 | minimum number of sub-steps in phase 2 |
| max.iter.p2 | maximum number of sub-steps in phase 2 |
| multiplication.iter.p2 | value for the lengths of sub-steps in phase 2 (multiplied by 2.52^k) |
| num.steps.p2 | number of optimisation steps in phase 2 |
| length.p3 | number of samples in phase 3 |
| neighborhood | way of choosing partitions: probability vector (actors swap, merge/division, single actor move) |
| fixed.estimates | if some parameters are fixed, list with as many elements as effects, these elements equal a fixed value if needed, or NULL if they should be estimated |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |

| | |
|---|---|
| double.averaging | option to average the statistics sampled in each sub-step of phase 2 |
| inv.zcov | initial value of the inverted covariance matrix (if a phase 3 was run before) to bypass the phase 1 |
| inv.scaling | initial value of the inverted scaling matrix (if a phase 3 was run before) to bypass the phase 1 |
| parallel | whether the phase 1 and 3 should be parallelized |
| parallel2 | whether there should be several phases 2 run in parallel |
| cpus | how many cores can be used |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

A list with the outputs of the three different phases of the algorithm

## Examples

```
# define an arbitrary set of n = 6 nodes with attributes, and an arbitrary covariate matrix
n <- 6
nodes <- data.frame(label = c("A","B","C","D","E","F"),
                    gender = c(1,1,2,1,2,2),
                    age = c(20,22,25,30,30,31))
friendship <- matrix(c(0, 1, 1, 1, 0, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 0, 0,
                       0, 1, 1, 0, 0, 1,
                       0, 0, 0, 0, 1, 0), 6, 6, TRUE)

# choose the effects to be included (see manual for all effect names)
effects <- list(names = c("num_groups","same","diff","tie"),
                objects = c("partition","gender","age","friendship"))
objects <- list()
objects[[1]] <- list(name = "friendship", object = friendship)

# define observed partition
partition <- c(1,1,2,2,2,3)


# estimate
startingestimates <- c(-2,0,0,0)
estimation <- estimate_ERPM(partition,
                            nodes,
                            objects,
                            effects,
                            startingestimates = startingestimates,
                            burnin = 100,
                            thining = 20,
                            length.p1 = 500, # number of samples in phase 1
```

```
                          multiplication.iter.p2 = 20, # iterations in phase 2
                          num.steps.p2 = 4, # number of phase 2 subphases
                          length.p3 = 1000) # number of samples in phase 3

# get results table
estimation
```

---

estimate_logL *Estimate log likelihood*

---

## Description

Function to estimate the log likelihood of a model for an observed partition

## Usage

```
estimate_logL(
  partition,
  nodes,
  effects,
  objects,
  theta,
  theta_0,
  M,
  num.steps,
  burnin,
  thining,
  neighborhoods = c(0.7, 0.3, 0),
  numgroups.allowed = NULL,
  numgroups.simulated = NULL,
  sizes.allowed = NULL,
  sizes.simulated = NULL,
  logL_0 = NULL,
  parallel = FALSE,
  cpus = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| partition | observed partition |
| nodes | node set (data frame) |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |

| | |
|---|---|
| theta | estimated model parameters |
| theta_0 | model parameters if all other effects than "num-groups" are fixed to 0 (basic Dirichlet partition model) |
| M | number of steps in the path-sampling algorithm |
| num.steps | number of samples in each step |
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |
| neighborhoods | = c(0.7,0.3,0) way of choosing partitions |
| numgroups.allowed | = NULL, # vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | = NULL, # vector containing the number of groups simulated |
| sizes.allowed | = NULL, vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | = NULL, vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| logL_0 | = NULL, if known, the value of the log likelihood of the basic dirichlet model |
| parallel | = FALSE, indicating whether the code should be run in parallel |
| cpus | = 1, number of cpus required for the parallelization |
| verbose | = FALSE, to print the current step the algorithm is in |

## Value

List with the log likelihood , AIC, lambda and the draws

## Examples

```
# estimate the log-likelihood and AIC of an estimated model (e.g. useful to compare two models)

# define an arbitrary set of n = 6 nodes with attributes, and an arbitrary covariate matrix
n <- 6
nodes <- data.frame(label = c("A","B","C","D","E","F"),
                    gender = c(1,1,2,1,2,2),
                    age = c(20,22,25,30,30,31))
friendship <- matrix(c(0, 1, 1, 1, 0, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 0, 0,
                       0, 1, 1, 0, 0, 1,
                       0, 0, 0, 0, 1, 0), 6, 6, TRUE)

# choose the effects to be included (see manual for all effect names)
effects <- list(names = c("num_groups","same","diff","tie"),
objects = c("partition","gender","age","friendship"))
```

```
objects <- list()
objects[[1]] <- list(name = "friendship", object = friendship)

# define observed partition
partition <- c(1,1,2,2,2,3)
# (an exemplary estimation is internally stored in order to save time)

# first: estimate the ML estimates of a simple model with only one parameter
# for number of groups (this parameter should be in the model!)
likelihood_function <- function(x){ exp(x*max(partition)) / compute_numgroups_denominator(n,x)}
curve(likelihood_function, from=-2, to=0)
parameter_base <- optimize(likelihood_function, interval=c(-2, 0), maximum=TRUE)
parameters_basemodel <- c(parameter_base$maximum,0,0,0)


# estimate logL and AIC
logL_AIC <- estimate_logL(partition,
                          nodes,
                          effects,
                          objects,
                          theta = estimation$results$est,
                          theta_0 = parameters_basemodel,
                          M = 3,
                          num.steps = 200,
                          burnin = 100,
                          thining = 20)
logL_AIC$logL
logL_AIC$AIC
```

---

`estimate_multipleERPM` *Estimate ERPM for multiple observations*

---

## Description

Function to estimate a given model for given observed (multiple) partitions. All options of the algorithm can be specified here.

## Usage

```
estimate_multipleERPM(
  partitions,
  presence.tables,
  nodes,
  objects,
  effects,
  startingestimates,
  gainfactor = 0.1,
```

```
  a.scaling = 0.8,
  r.truncation.p1 = -1,
  r.truncation.p2 = -1,
  burnin = 30,
  thining = 10,
  length.p1 = 100,
  min.iter.p2 = NULL,
  max.iter.p2 = NULL,
  multiplication.iter.p2 = 200,
  num.steps.p2 = 6,
  length.p3 = 1000,
  neighborhood = c(0.7, 0.3, 0),
  fixed.estimates = NULL,
  numgroups.allowed = NULL,
  numgroups.simulated = NULL,
  sizes.allowed = NULL,
  sizes.simulated = NULL,
  double.averaging = FALSE,
  inv.zcov = NULL,
  inv.scaling = NULL,
  parallel = FALSE,
  parallel2 = FALSE,
  cpus = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| partitions | observed partitions |
| presence.tables | XXX |
| nodes | nodeset (data frame) |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| startingestimates | first guess for the model parameters |
| gainfactor | numeric used to decrease the size of steps made in the Newton optimization |
| a.scaling | numeric used to reduce the influence of non-diagonal elements in the scaling matrix (for stability) |
| r.truncation.p1 | numeric used to limit extreme values in the covariance matrix (for stability) |
| r.truncation.p2 | numeric used to limit extreme values in the covariance matrix (for stability) |
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |

`length.p1` number of samples in phase 1

`min.iter.p2` minimum number of sub-steps in phase 2

`max.iter.p2` maximum number of sub-steps in phase 2

`multiplication.iter.p2`
value for the lengths of sub-steps in phase 2 (multiplied by 2.52^k)

`num.steps.p2` number of optimisation steps in phase 2

`length.p3` number of samples in phase 3

`neighborhood` way of choosing partitions: probability vector (actors swap, merge/division, single actor move)

`fixed.estimates`
if some parameters are fixed, list with as many elements as effects, these elements equal a fixed value if needed, or NULL if they should be estimated

`numgroups.allowed`
vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max)

`numgroups.simulated`
vector containing the number of groups simulated

`sizes.allowed` vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max)

`sizes.simulated`
vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max)

`double.averaging`
option to average the statistics sampled in each sub-step of phase 2

`inv.zcov` initial value of the inverted covariance matrix (if a phase 3 was run before) to bypass the phase 1

`inv.scaling` initial value of the inverted scaling matrix (if a phase 3 was run before) to bypass the phase 1

`parallel` whether the phase 1 and 3 should be parallelized

`parallel2` whether there should be several phases 2 run in parallel

`cpus` how many cores can be used

`verbose` logical: should intermediate results during the estimation be printed or not? Defaults to FALSE.

## Value

A list with the outputs of the three different phases of the algorithm

## Examples

```
# define an arbitrary set of n = 6 nodes with attributes, and an arbitrary covariate matrix
n <- 6
nodes <- data.frame(label = c("A","B","C","D","E","F"),
                    gender = c(1,1,2,1,2,2),
```

```
                    age = c(20,22,25,30,30,31))
friendship <- matrix(c(0, 1, 1, 1, 0, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 0, 0,
                       0, 1, 1, 0, 0, 1,
                       0, 0, 0, 0, 1, 0), 6, 6, TRUE)

# specify whether nodes are present at different points of time
presence.tables <- matrix(c(1, 1, 1, 1, 1, 1,
                            0, 1, 1, 1, 1, 1,
                            1, 0, 1, 1, 1, 1), 6, 3)

# choose effects to be included in the estimated model
effects_multiple <- list(names = c("num_groups","same","diff","tie","inertia_1"),
                objects = c("partitions","gender","age","friendship","partitions"),
                objects2 = c("","","","",""))
objects_multiple <- list()
objects_multiple[[1]] <- list(name = "friendship", object = friendship)

# define the observation
partitions <- matrix(c(1, 1, 2, 2, 2, 3,
                       NA, 1, 1, 2, 2, 2,
                       1, NA, 2, 3, 3, 1), 6, 3)


# estimate
startingestimates <- c(-2,0,0,0,0)
estimation <- estimate_multipleERPM(partitions,
                                    presence.tables,
                                    nodes,
                                    objects_multiple,
                                    effects_multiple,
                                    startingestimates = startingestimates,
                                    burnin = 100,
                                    thining = 50,
                                    gainfactor = 0.6,
                                    length.p1 = 200,
                                    multiplication.iter.p2 = 20,
                                    num.steps.p2 = 4,
                                    length.p3 = 1000)

# get results table
estimation
```

---

## exactestimates_numgroups

*Exact estimates number of groups*

---

## Description

This function finds the best estimate for a model only including the statistics of number of groups. It does a grid search for a vector of potential parameters, for all numbers of groups.

## Usage

```
exactestimates_numgroups(num.nodes, pmin, pmax, pinc)
```

## Arguments

| | |
|---|---|
| num.nodes | number of nodes |
| pmin | lowest parameter value |
| pmax | highest parameter value |
| pinc | increment between different parameter values |

## Value

a list

---

# find_all_partitions *Function to enumerate all possible partitions for a given n*

---

## Description

Function to enumerate all possible partitions for a given n

## Usage

```
find_all_partitions(n)
```

## Arguments

| | |
|---|---|
| n | number of nodes |

## Value

matrix where each line corresponds to a possible partition

## Examples

```
n <- 6
all_partitions <- find_all_partitions(n)
```

gridsearch_burninthining_multiple
*Grid - search burnin thining multiple*

## Description

Function that simulates the Markov chain for a given model and several sets of transitions (the neighborhoods), for multiple partitions. For each neighborhood, it calculates the autocorrelation of statistics for different thinings and the average statistics for different burn-ins. Then the best neighborhood can be selected along with good values for burn-in and thining

## Usage

```
gridsearch_burninthining_multiple(
  partitions,
  presence.tables,
  theta,
  nodes,
  effects,
  objects,
  num.steps,
  neighborhoods,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  max.thining,
  parallel = FALSE,
  cpus = 1
)
```

## Arguments

| | |
|---|---|
| partitions | Observed partitions |
| presence.tables | Presence of nodes |
| theta | Initial model parameters |
| nodes | Node set (data frame) |
| effects | Effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | Objects used for statistics calculation (list with a vector "name", and a vector "object") |
| num.steps | Number of samples wanted |
| neighborhoods | List of probability vectors (proba actors swap, proba merge/division, proba single actor move) |

`numgroups.allowed`
: vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max)

`numgroups.simulated`
: vector containing the number of groups simulated

`sizes.allowed`
: Vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max)

`sizes.simulated`
: Vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max)

`max.thining`
: Where to stop adding thining

`parallel`
: False, to run different neighborhoods in parallel

`cpus`
: Equal to 1

## Value

list

---

`gridsearch_burninthining_single` *Grid - search burnin thining single*

---

## Description

Function that simulates the Markov chain for a given model and several sets of transitions (the neighborhoods), for a single partition. For each neighborhood, it calculates the autocorrelation of statistics for different thinings and the average statistics for different burn-ins. Then the best neighborhood can be selected along with good values for burn-in and thining

## Usage

```
gridsearch_burninthining_single(
  partition,
  theta,
  nodes,
  effects,
  objects,
  num.steps,
  neighborhoods,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  max.thining,
  parallel = FALSE,
  cpus = 1
)
```

### Arguments

| | |
|---|---|
| partition | A partition (vector) |
| theta | Initial model parameters |
| nodes | Node set (data frame) |
| effects | Effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | Objects used for statistics calculation (list with a vector "name", and a vector "object") |
| num.steps | Number of samples wanted |
| neighborhoods | List of probability vectors (proba actors swap, proba merge/division, proba single actor move) |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | Vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | Vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| max.thining | Where to stop adding thining |
| parallel | False, to run different neighborhoods in parallel |
| cpus | Equal to 1 |

### Value

list

---

## gridsearch_burnin_single

*Grid - search burnin single*

---

### Description

Function that can be used to find a good length for the burn-in of the Markov chain for a given model and differents sets of transitions in the chain (the neighborhoods). For each neighborhood, it draws a chain and calculates the mean statistics for different burn-ins.

## Usage

```
gridsearch_burnin_single(
  partition,
  theta,
  nodes,
  effects,
  objects,
  num.steps,
  neighborhoods,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  parallel = FALSE,
  cpus = 1
)
```

## Arguments

| | |
|---|---|
| partition | A partition (vector) |
| theta | Initial model parameters |
| nodes | Node set (data frame) |
| effects | Effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | Objects used for statistics calculation (list with a vector "name", and a vector "object") |
| num.steps | Number of samples wanted |
| neighborhoods | List of probability vectors (proba actors swap, proba merge/division, proba single actor move) |
| numgroups.allowed | = NULL, # vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | Vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | Vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| parallel | False, to run different neighborhoods in parallel |
| cpus | Equal to 1 |

## Value

all simulations

gridsearch_thining_single *Grid - search thining single*

## Description

Function that can be used to find a good length for the thining of the Markov chain for a given model and differents sets of transitions in the chain (the neighborhoods). For each neighborhood, it draws a chain and calculates the autocorrelation of statistics for different thinings.

## Usage

```
gridsearch_thining_single(
  partition,
  theta,
  nodes,
  effects,
  objects,
  num.steps,
  neighborhoods,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  burnin,
  max.thining,
  parallel = FALSE,
  cpus = 1
)
```

## Arguments

| | |
|---|---|
| partition | A partition (vector) |
| theta | Initial model parameters |
| nodes | Node set (data frame) |
| effects | Effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | Objects used for statistics calculation (list with a vector "name", and a vector "object") |
| num.steps | Number of samples wanted |
| neighborhoods | List of probability vectors (proba actors swap, proba merge/division, proba single actor move) |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |

| | |
|---|---|
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | Vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | Vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| burnin | length of the burn-in period |
| max.thining | maximal value for the thining to be tested |
| parallel | False, to run different neighborhoods in parallel |
| cpus | Equal to 1 |

## Value

all simulations

---

# group_size *Statistics on the size of groups in a partition*

---

## Description

This function computes the average or the standard deviation of the size of groups in a partition.

## Usage

```
group_size(partition, stat)
```

## Arguments

| | |
|---|---|
| partition | A partition (vector) |
| stat | The statistic to compute : 'avg' for average and 'sd' for standard deviation |

## Value

A number corresponding to the correlation coefficient if the attribute is numerical or the correlation ratio if the attribute is categorical.

## Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
group_size(p,'avg')
group_size(p,'sd')
```

## icc *Intra class correlation*

### Description

This function computes the intra class correlation correlation of attributes for 2 randomly drawn individuals in the same group.

### Usage

```
icc(partition, attribute)
```

### Arguments

| | |
|---|---|
| partition | A partition |
| attribute | A vector containing the values of the attribute |

### Value

A number corresponding to the ICC

### Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
at <- c(3,5,23,2,1,0,3,9,2)
icc(p, at)
```

## number_categories *Number of individuals having an attribute*

### Description

This function computes the total number of individuals being in a category of an attribute in a partition. It also computes the sum of the proportion in each group of individuals being in a category.

### Usage

```
number_categories(partition, attribute, stat, category)
```

### Arguments

| | |
|---|---|
| partition | A partition (vector) |
| attribute | A vector containing the values of the attribute |
| stat | The statistic to compute : 'avg' for the sum of proportion per group and 'sum' for the total number |
| category | The category to consider or category = 'all' if all categories have to be considered |

## Value

The statisic chosen in stat depending on the value of category. If category = 'all', returns a vector.

## Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
at <- c(1,0,0,0,1,1,0,0,1)
number_categories(p,at,'avg','all')
```

---

`number_ties` *Same pairs of individuals in a partition*

---

## Description

This function computes the number of ties.

## Usage

```
number_ties(partition, dyadic_attribute, stat)
```

## Arguments

| | |
|---|---|
| `partition` | A partition (vector) |
| `dyadic_attribute` | A matrix containing the values of the attribute |
| `stat` | The statistic to compute : 'avg_pergroup' for the average per group , 'sum_pergroup' for the sum, 'sum_perind' and 'avg_perind' for the number of ties per individuals each individual has in its group. |

## Value

The statisic chosen in stat

## Examples

```
p <- c(1,2,2,3,3,4)
v <- c(0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0)
at <- matrix(v,6,6, byrow = TRUE)
number_ties(p,at,'avg_pergroup')
```

`order_groupids` *Function to replace the ids of the group without forgetting an id and put in the first appearance order for example:* `[2 1 1 4 2]` *becomes* `[1 2 2 3 1]`

## Description

Function to replace the ids of the group without forgetting an id and put in the first appearance order for example: `[2 1 1 4 2]` becomes `[1 2 2 3 1]`

## Usage

```
order_groupids(partition)
```

## Arguments

| | |
|---|---|
| `partition` | observed partition |

## Value

a vector (partition)

`outcomeObjects` *Exemplary outcome objects for the ERPM Package*

## Description

These are exemplary outcome objects for the ERPM package and can be used in order not to run all precedent functions and thus save time. The following products are provided:

## Format

`estimation` An results object created by the function `estimate_ERPM()`.

phase1 *Core function for Phase 1*

## Description

Core function for Phase 1

## Usage

```
phase1(
  startingestimates,
  inv.zcov,
  inv.scaling,
  z.phase1,
  z.obs,
  nodes,
  effects,
  objects,
  r.truncation.p1,
  length.p1,
  fixed.estimates,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| startingestimates | vector containing initial parameter values |
| inv.zcov | inverted covariance matrix |
| inv.scaling | scaling matrix |
| z.phase1 | statistics retrieved from phase 1 |
| z.obs | observed statistics |
| nodes | node set (data frame) |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| r.truncation.p1 | numeric used to limit extreme values in the covariance matrix (for stability) |
| length.p1 | number of samples in phase 1 |
| fixed.estimates | if some parameters are fixed, list with as many elements as effects, these elements equal a fixed value if needed, or NULL if they should be estimated |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

### Value

estimated parameters after phase 1

---

`plot_averagesizes` *Plot average sizes*

---

### Description

Function to plot the average size of a random partition depending on the number of nodes

### Usage

```
plot_averagesizes(nmin, nmax, ninc)
```

### Arguments

| | |
|---|---|
| `nmin` | minimum number of nodes |
| `nmax` | maximum number of nodes |
| `ninc` | increment between the different number of nodes |

### Value

a vector

---

`plot_numgroups_likelihood`
*Plot likelihood of number groups*

---

### Description

Function to plot the log-likelihood of the model with a single statistic (number of groups) depending on the parameter value for this statistic

### Usage

```
plot_numgroups_likelihood(m.obs, num.nodes, pmin, pmax, pinc)
```

### Arguments

| | |
|---|---|
| `m.obs` | observed number of groups |
| `num.nodes` | number of nodes |
| `pmin` | lowest parameter value |
| `pmax` | highest parameter value |
| `pinc` | increment between different parameter values |

## Value

a vector

---

`plot_partition` *Visualization of partition*

---

## Description

This function plot the groups of a partition

## Usage

```
plot_partition(
  partition,
  title = NULL,
  group.color = NULL,
  attribute.color = NULL,
  attribute.shape = NULL
)
```

## Arguments

| | |
|---|---|
| `partition` | A partition (vector) |
| `title` | Character, the title of the plot (default=NULL) |
| `group.color` | A vector with the colors of the groups (default=NULL) |
| `attribute.color` | A vector, attribute to represent with colors (default=NULL) |
| `attribute.shape` | A vector, attribute to represent with shapes (default=NULL) |

## Value

A plot of the partition

## Examples

```
p <- c(1,1,1,2,2,2,2,3,3,3,4,4,4,4,4,4)
attr1 <- c(1,0,0,1,0,0,1,0,1,0,1,1,1,1,1,2)
attr2 <- c(1,1,1,1,0,0,3,0,1,0,1,1,1,1,1,2)
plot_partition(p,attribute.color = attr1, attribute.shape = attr2)
```

## print.results.bayesian.erpm

*Print results of bayesian estimation (beta version)*

### Description

Print results of bayesian estimation (beta version)

### Usage

```
## S3 method for class 'results.bayesian.erpm'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | output of the bayesian estimate function |
| ... | For internal use only. |

### Value

a data frame

## print.results.list.erpm

*Print estimation results*

### Description

Print estimation results

### Usage

```
## S3 method for class 'results.list.erpm'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | output of the estimate function |
| ... | For internal use only. |

### Value

a data frame

## print.results.p3.erpm *Print results of estimation of phase 3*

### Description

Print results of estimation of phase 3

### Usage

```
## S3 method for class 'results.p3.erpm'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | output of the estimate function |
| ... | For internal use only. |

### Value

a data frame

## proportion_isolate *Proportion of isolates*

### Description

This function computes the proportion of individuals not joining others.

### Usage

```
proportion_isolate(partition)
```

### Arguments

| | |
|---|---|
| partition | A partition (vector) |

### Value

A number corresponding to proportion of individuals alone.

### Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
proportion_isolate(p)
```

## range_attribute *Range of attribute in groups*

### Description

This function computes the sum or the average range of an attribute for groups in a partition.

### Usage

```
range_attribute(partition, attribute, stat)
```

### Arguments

| | |
|---|---|
| partition | A partition (vector) |
| attribute | A vector containing the values of the attribute |
| stat | The statistic to compute : 'avg_pergroup' for the average per group and 'sum_pergroup' for the sum of the ranges |

### Value

The statisic chosen in stat

### Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
at <- c(3,5,23,2,1,0,3,9,2)
range_attribute(p,at,'avg_pergroup')
```

## run_phase1_multiple *Phase 1 wrapper for multiple observations*

### Description

Phase 1 wrapper for multiple observations

### Usage

```
run_phase1_multiple(
  partitions,
  startingestimates,
  z.obs,
  presence.tables,
  nodes,
  effects,
  objects,
```

```
  burnin,
  thining,
  gainfactor,
  a.scaling,
  r.truncation.p1,
  length.p1,
  neighborhood,
  fixed.estimates,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  parallel = FALSE,
  cpus = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| partitions | observed partitions |
| startingestimates | vector containing initial parameter values |
| z.obs | observed statistics |
| presence.tables | data frame to indicate which times nodes are present in the partition |
| nodes | node set (data frame) |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |
| gainfactor | gain factor (useless now) |
| a.scaling | scaling factor |
| r.truncation.p1 | truncation factor (for stability) |
| length.p1 | number of samples for phase 1 |
| neighborhood | vector for the probability of choosing a particular transition in the chain |
| fixed.estimates | if some parameters are fixed, list with as many elements as effects, these elements equal a fixed value if needed, or NULL if they should be estimated |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |

| | |
|---|---|
| sizes.allowed | vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | vector of group sizes allowed in the Markov chain but not necessarily sampled (now, it only works for vectors like size_min:size_max) |
| parallel | boolean to indicate whether the code should be run in parallel |
| cpus | number of cpus if parallel = TRUE |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

a list

---

`run_phase1_single` *Phase 1 wrapper for single observation*

---

## Description

Phase 1 wrapper for single observation

## Usage

```
run_phase1_single(
  partition,
  startingestimates,
  z.obs,
  nodes,
  effects,
  objects,
  burnin,
  thining,
  gainfactor,
  a.scaling,
  r.truncation.p1,
  length.p1,
  neighborhood,
  fixed.estimates,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  parallel = TRUE,
  cpus = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| partition | observed partition |
| startingestimates | vector containing initial parameter values |
| z.obs | observed statistics |
| nodes | node set (data frame) |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |
| gainfactor | gain factor (useless now) |
| a.scaling | scaling factor |
| r.truncation.p1 | truncation factor (for stability) |
| length.p1 | number of samples for phase 1 |
| neighborhood | vector for the probability of choosing a particular transition in the chain |
| fixed.estimates | if some parameters are fixed, list with as many elements as effects, these elements equal a fixed value if needed, or NULL if they should be estimated |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | vector of group sizes allowed in the Markov chain but not necessarily sampled (now, it only works for vectors like size_min:size_max) |
| parallel | boolean to indicate whether the code should be run in parallel |
| cpus | number of cpus if parallel = TRUE |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

a list

`run_phase2_multiple` *Phase 2 wrapper for multiple observation*

## Description

Phase 2 wrapper for multiple observation

## Usage

```
run_phase2_multiple(
  partitions,
  estimates.phase1,
  inv.zcov,
  inv.scaling,
  z.obs,
  presence.tables,
  nodes,
  effects,
  objects,
  burnin,
  thining,
  num.steps,
  gainfactors,
  r.truncation.p2,
  min.iter,
  max.iter,
  multiplication.iter,
  neighborhood,
  fixed.estimates,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  double.averaging,
  parallel = FALSE,
  cpus = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `partitions` | observed partitions |
| `estimates.phase1` | vector containing parameter values after phase 1 |
| `inv.zcov` | inverted covariance matrix |
| `inv.scaling` | scaling matrix |

| | |
|---|---|
| z.obs | observed statistics |
| presence.tables | data frame to indicate which times nodes are present in the partition |
| nodes | node set (data frame) |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |
| num.steps | number of sub-phases in phase 2 |
| gainfactors | vector of gain factors |
| r.truncation.p2 | truncation factor |
| min.iter | minimum numbers of steps in each subphase |
| max.iter | maximum numbers of steps in each subphase |
| multiplication.iter | used to calculate min.iter and max.iter if not specified |
| neighborhood | vector for the probability of choosing a particular transition in the chain |
| fixed.estimates | if some parameters are fixed, list with as many elements as effects, these elements equal a fixed value if needed, or NULL if they should be estimated |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| double.averaging | boolean to indicate whether we follow the double-averaging procedure (often leads to better convergence) |
| parallel | boolean to indicate whether the code should be run in parallel |
| cpus | number of cpus if parallel = TRUE |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

a list

`run_phase2_single` *Phase 2 wrapper for single observation*

## Description

Phase 2 wrapper for single observation

## Usage

```
run_phase2_single(
  partition,
  estimates.phase1,
  inv.zcov,
  inv.scaling,
  z.obs,
  nodes,
  effects,
  objects,
  burnin,
  thining,
  num.steps,
  gainfactors,
  r.truncation.p2,
  min.iter,
  max.iter,
  multiplication.iter,
  neighborhood,
  fixed.estimates,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  double.averaging,
  parallel = FALSE,
  cpus = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `partition` | observed partition |
| `estimates.phase1` | vector containing parameter values after phase 1 |
| `inv.zcov` | inverted covariance matrix |
| `inv.scaling` | scaling matrix |
| `z.obs` | observed statistics |

| | |
|---|---|
| nodes | node set (data frame) |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |
| num.steps | number of sub-phases in phase 2 |
| gainfactors | vector of gain factors |
| r.truncation.p2 | truncation factor |
| min.iter | minimum numbers of steps in each subphase |
| max.iter | maximum numbers of steps in each subphase |
| multiplication.iter | used to calculate min.iter and max.iter if not specified |
| neighborhood | vector for the probability of choosing a particular transition in the chain |
| fixed.estimates | if some parameters are fixed, list with as many elements as effects, these elements equal a fixed value if needed, or NULL if they should be estimated |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| double.averaging | boolean to indicate whether we follow the double-averaging procedure (often leads to better convergence) |
| parallel | boolean to indicate whether the code should be run in parallel |
| cpus | number of cpus if parallel = TRUE |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

a list

`run_phase3_multiple` *Phase 3 wrapper for multiple observation*

## Description

Phase 3 wrapper for multiple observation

## Usage

```
run_phase3_multiple(
  partitions,
  estimates.phase2,
  z.obs,
  presence.tables,
  nodes,
  effects,
  objects,
  burnin,
  thining,
  a.scaling,
  length.p3,
  neighborhood,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  fixed.estimates,
  parallel = FALSE,
  cpus = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `partitions` | observed partitions |
| `estimates.phase2` | vector containing parameter values after phase 2 |
| `z.obs` | observed statistics |
| `presence.tables` | data frame to indicate which times nodes are present in the partition |
| `nodes` | node set (data frame) |
| `effects` | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| `objects` | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| `burnin` | integer for the number of burn-in steps before sampling |

| | |
|---|---|
| thining | integer for the number of thining steps between sampling |
| a.scaling | multiplicative factor for out-of-diagonal elements of the covariance matrix |
| length.p3 | number of samples in phase 3 |
| neighborhood | vector for the probability of choosing a particular transition in the chain |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| fixed.estimates | if some parameters are fixed, list with as many elements as effects, these elements equal a fixed value if needed, or NULL if they should be estimated |
| parallel | boolean to indicate whether the code should be run in parallel |
| cpus | number of cpus if parallel = TRUE |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

a list

---

`run_phase3_single` *Phase 3 wrapper for single observation*

---

## Description

Phase 3 wrapper for single observation

## Usage

```
run_phase3_single(
  partition,
  estimates.phase2,
  z.obs,
  nodes,
  effects,
  objects,
  burnin,
  thining,
```

```
  a.scaling,
  length.p3,
  neighborhood,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  fixed.estimates,
  parallel = FALSE,
  cpus = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| partition | observed partition |
| estimates.phase2 | vector containing parameter values after phase 2 |
| z.obs | observed statistics |
| nodes | node set (data frame) |
| effects | effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | objects used for statistics calculation (list with a vector "name", and a vector "object") |
| burnin | integer for the number of burn-in steps before sampling |
| thining | integer for the number of thining steps between sampling |
| a.scaling | multiplicative factor for out-of-diagonal elements of the covariance matrix |
| length.p3 | number of sampled partitions in phase 3 |
| neighborhood | vector for the probability of choosing a particular transition in the chain |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| fixed.estimates | if some parameters are fixed, list with as many elements as effects, these elements equal a fixed value if needed, or NULL if they should be estimated |
| parallel | boolean to indicate whether the code should be run in parallel |
| cpus | number of cpus if parallel = TRUE |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

a list

---

`same_pairs` *Same pairs of individuals in a partition*

---

## Description

This function computes the total number, the average number having the same value of a categorical variable and the number of individuals a partition.

## Usage

```
same_pairs(partition, attribute, stat)
```

## Arguments

| | |
|---|---|
| `partition` | A partition (vector) |
| `attribute` | A vector containing the values of the attribute |
| `stat` | The statistic to compute : 'avg_pergroup' for the average, 'sum_pergroup' for the sum, 'sum_perind' and 'avg_perind' for the number of ties per individual each individual has in its group. |

## Value

The statistic chosen in stat

## Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
at <- c(0,1,1,1,1,0,0,0,0)
same_pairs(p,at,'avg_pergroup')
```

---

`similar_pairs` *Similar pairs of individuals in a partition*

---

## Description

This function computes the total number, the average number having the close values of a numerical variable and the number of individuals a partition.

## Usage

```
similar_pairs(partition, attribute, stat, threshold)
```

### Arguments

| | |
|---|---|
| partition | A partition (vector) |
| attribute | A vector containing the values of the attribute |
| stat | The statistic to compute : 'avg_pergroup' for the average, 'sum_pergroup' for the sum, 'sum_perind' and 'avg_perind' for individuals |
| threshold | Threshold to determine if 2 individuals attributes values are close |

### Value

The statisic chosen in stat

### Examples

```
p <- c(1,2,2,3,3,4,4,4,5)
at <- c(3,5,23,2,1,0,3,9,2)
similar_pairs(p,at,1,'avg_pergroup')
```

---

## simulate_burninthining_multiple

*Simulate burnin thining multiple*

---

### Description

Function that simulates the Markov chain for a given model and a set of transitions (the neighborhood), for multiple partitions. It calculates the autocorrelation of statistics for different thinings and the average statistics for different burn-ins.

### Usage

```
simulate_burninthining_multiple(
  partitions,
  presence.tables,
  theta,
  nodes,
  effects,
  objects,
  num.steps,
  neighborhood,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  max.thining,
  verbose = FALSE
)
```

### Arguments

| | |
|---|---|
| partitions | Observed partitions |
| presence.tables | to indicate which nodes were present when |
| theta | Initial model parameters |
| nodes | Node set (data frame) |
| effects | Effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | Objects used for statistics calculation (list with a vector "name", and a vector "object") |
| num.steps | Number of samples wanted |
| neighborhood | Way of choosing partitions: probability vector (proba actors swap, proba merge/division, proba single actor move) |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | Vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | Vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| max.thining | maximal number of simulated steps in the thining |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

### Value

A list

---

## simulate_burninthining_single

*Simulate burnin thining single*

---

### Description

Function that simulates the Markov chain for a given model and a set of transitions (the neighborhood), for a single partition. It calculates the autocorrelation of statistics for different thinings and the average statistics for different burn-ins.

## Usage

```
simulate_burninthining_single(
  partition,
  theta,
  nodes,
  effects,
  objects,
  num.steps,
  neighborhood,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  max.thining,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| partition | Observed partition (vector) |
| theta | Initial model parameters |
| nodes | Node set (data frame) |
| effects | Effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | Objects used for statistics calculation (list with a vector "name", and a vector "object") |
| num.steps | Number of samples wanted |
| neighborhood | Way of choosing partitions: probability vector (proba actors swap, proba merge/division, proba single actor move) |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | Vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | Vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| max.thining | maximal number of simulated steps in the thining |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

A list

simulate_burnin_single *Simulate burn in single*

## Description

Function that can be used to find a good length for the burn-in of the Markov chain for a given model and a given set of transitions in the chain (the neighborhood). It draws a chain and calculates the mean statistics for different burn-ins.

## Usage

```
simulate_burnin_single(
  partition,
  theta,
  nodes,
  effects,
  objects,
  num.steps,
  neighborhood,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated
)
```

## Arguments

| | |
|---|---|
| partition | A partition (vector) |
| theta | Initial model parameters |
| nodes | Node set (data frame) |
| effects | Effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| objects | Objects used for statistics calculation (list with a vector "name", and a vector "object") |
| num.steps | Number of samples wanted |
| neighborhood | Way of choosing partitions: probability vector (proba actors swap, proba merge/division, proba single actor move) |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | Vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |

`sizes.simulated`
: Vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max)

## Value

A list with list the draws, the moving.means and the moving means smoothed

---

`simulate_thining_single` *Simulate thining single*

---

## Description

Function that can be used to find a good length for the thining of the Markov chain for a given model and a set of transitions in the chain (the neighborhood). It draws a chain and calculates the autocorrelation of statistics for different thinings.

## Usage

```
simulate_thining_single(
  partition,
  theta,
  nodes,
  effects,
  objects,
  num.steps,
  neighborhood,
  numgroups.allowed,
  numgroups.simulated,
  sizes.allowed,
  sizes.simulated,
  burnin,
  max.thining,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `partition` | A partition (vector) |
| `theta` | Initial model parameters |
| `nodes` | Node set (data frame) |
| `effects` | Effects/sufficient statistics (list with a vector "names", and a vector "objects") |
| `objects` | Objects used for statistics calculation (list with a vector "name", and a vector "object") |
| `num.steps` | Number of samples wanted |

| | |
|---|---|
| neighborhood | Way of choosing partitions: probability vector (proba actors swap, proba merge/division, proba single actor move) |
| numgroups.allowed | vector containing the number of groups allowed in the partition (now, it only works with vectors like num_min:num_max) |
| numgroups.simulated | vector containing the number of groups simulated |
| sizes.allowed | Vector of group sizes allowed in sampling (now, it only works for vectors like size_min:size_max) |
| sizes.simulated | Vector of group sizes allowed in the Markov chain but not necessraily sampled (now, it only works for vectors like size_min:size_max) |
| burnin | number of simulated steps for the burn-in |
| max.thining | maximal number of simulated steps in the thining |
| verbose | logical: should intermediate results during the estimation be printed or not? Defaults to FALSE. |

## Value

A list

---

# Stirling2_constraints *Function to calculate the number of partitions with k groups of sizes between smin and smax*

---

## Description

Function to calculate the number of partitions with k groups of sizes between smin and smax

## Usage

```
Stirling2_constraints(n, k, smin, smax)
```

## Arguments

| | |
|---|---|
| n | number of nodes |
| k | number of groups |
| smin | minimum group size possible in the partition |
| smax | maximum group size possible in the partition |

## Value

a numeric

## Examples

```
n <- 6
k <- 2
size_min <- 2
size_max <- 4
Stirling2_constraints(n,k,size_min,size_max)
```

# Index