Title Network Analysis and Community Detection

Description Features tools for the network data analysis and community detection.

Provides multiple methods for fitting, model selection and goodness-of-fit testing in degree-corrected stochastic blocks models.

Most of the computations are fast and scalable for sparse networks, esp. for Poisson versions of the models.

Implements the following:

Lei (2016) <doi:10.1214/15-AOS1370>

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

Suggests testthat, knitr, rmarkdown, igraph, ggplot2, dplyr, tidyr, tibble, mixtools, EnvStats, purrr, RSpectra

Depends R (>= 2.10)

Imports magrittr, Rcpp, Matrix, foreach, methods, stats, grDevices, graphics

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

URL https://github.com/aaamini/nett

BugReports https://github.com/aaamini/nett/issues

NeedsCompilation yes

Author Arash A. Amini [aut, cre] (<https://orcid.org/0000-0002-2808-8310>), Linfan Zhang [aut]
Maintainer  Arash A. Amini <aaamini@ucla.edu>
Repository  CRAN
Date/Publication  2022-11-09 10:50:05 UTC

**R topics documented:**

- adj_spec_test .......................... 3
- bethe_hessian_select .................. 4
- compute_block_sums ........................ 5
- compute_confusion_matrix ................. 5
- compute_mutual_info ................... 6
- estim_dcsbm .......................... 6
- eval_dcsbm_bic ........................ 6
- eval_dcsbm_like ....................... 8
- eval_dcsbm_loglr ...................... 9
- extract_largest_cc .................... 10
- extract_low_deg_comp .................. 10
- fast_cpl ............................. 11
- fast_sbm ............................. 12
- gen_rand_conn .......................... 12
- get_dcsbm_exav_deg .................... 13
- label_mat2vec ........................ 14
- label_vec2mat ........................ 14
- nac_test .............................. 15
- plot_deg_dist ........................ 16
- plot_net .............................. 16
- plot_roc .............................. 17
- plot_smooth_profile ................... 18
- polblogs .............................. 18
- pp_conn ............................... 19
- printf ................................ 20
- rsymperm .............................. 20
- sample_deer .......................... 21
- sample_dclvm .......................... 21
- sample_dcpp ........................... 22
- sample_dcsbm ........................ 23
- sample_tdcsm ................................ 23
- simulate_roc .......................... 24
- sinkhorn_knopp ........................ 25
- snac_resample ........................ 26
- snac_select .......................... 26
- snac_test ............................ 27
- spec_elust ........................... 29
- spec_repr ............................ 30

Index ................................ 31
adj_spec_test

**Description**

The adjusted spectral goodness-of-fit test based on Poisson DCSBM.

The test is a natural extension on Lei’s work of testing goodness-of-fit for SBM. The residual matrix $\hat{A}$ is computed from the DCSBM estimation expectation of $A$. To speed up computation, the residual matrix uses Poisson variance instead. Specifically,

$$
\hat{A}_{ij} = (A_{ij} - \hat{P}_{ij})/(n\hat{P}_{ij})^{1/2}, \quad \hat{P}_{ij} = \hat{\theta}_i\hat{\theta}_j\hat{B}_{\hat{z}_i,\hat{z}_j} \cdot 1\{i \neq j\}
$$

where $\hat{\theta}$ and $\hat{B}$ are computed using `estim_dcsbm` if not provided.

Adjusted spectral test

**Usage**

```
adj_spec_test(
  A, 
  K, 
  z = NULL, 
  DC = TRUE, 
  theta = NULL, 
  B = NULL, 
  cluster_fct = spec_clust, 
  ... 
)
```

**Arguments**

- `A` : adjacency matrix.
- `K` : number of communities.
- `z` : label vector for rows of adjacency matrix. If not given, will be calculated by the spectral clustering.
- `DC` : whether or not include degree correction in the parameter estimation.
- `theta` : give the propensity parameter directly.
- `B` : give the connectivity matrix directly.
- `cluster_fct` : community detection function to get $z$, by default using `spec_clust`.
- `...` : additional arguments for `cluster_fct`.

**Value**

Adjusted spectral test statistics.
bethe_hessian_select

References


bethe_hessian_select  Beth-Hessian model selection

Description

Estimate the number of communities under block models by using the spectral properties of network Beth-Hessian matrix with moment correction.

Usage

bethe_hessian_select(A, Kmax)

Arguments

A  adjacency matrix.
Kmax  the maximum number of communities to check.

Value

A list of result

K  estimated the number of communities
rho  eigenvalues of the Beth-Hessian matrix

References

**compute_block_sums**

*Block sum of an adjacency matrix*

**Description**

Compute the block sum of an adjacency matrix given a label vector.

**Usage**

```r
compute_block_sums(A, z)
```

**Arguments**

- `A`: adjacency matrix.
- `z`: label vector.

**Value**

A $K \times L$ matrix with $(k,l)$-th element as $\sum_{i,j} A_{i,j} 1_{z_i = k, z_j = l}$

---

**compute_confusion_matrix**

*Compute confusion matrix*

**Description**

Compute confusion matrix

**Usage**

```r
compute_confusion_matrix(z, y, K = NULL)
```

**Arguments**

- `z`: a label vector
- `y`: a label vector
- `K`: number of labels in both `z` and `y`

**Value**

A $K \times K$ confusion matrix between `z` and `y`
compute_mutual_info  \(\text{Compute normalized mutual information (NMI)}\)

**Description**

Compute the NMI between two label vectors with the same cluster number

**Usage**

\[
\text{compute_mutual_info}(z, y)
\]

**Arguments**

- \(z\) a label vector
- \(y\) a label vector

**Value**

NMI between \(z\) and \(y\)

estim_dcsbm  \(\text{Estimate model parameters of a DCSBM}\)

**Description**

Compute the block sum of an adjacency matrix given a label vector.

**Usage**

\[
\text{estim_dcsbm}(A, z)
\]

**Arguments**

- \(A\) adjacency matrix.
- \(z\) label vector.

**Details**

\[
\hat{B}_{k\ell} = \frac{N_{k\ell}(\hat{z})}{m_{k\ell}(\hat{z})}, \quad \hat{\theta}_i = \frac{n_{z_i}(\hat{z})d_i}{\sum_{j: z_j = \hat{z}_i} a_{ij}}
\]

where \(N_{k\ell}(\hat{z})\) is the sum of the elements of \(A\) in block \((k, \ell)\) specified by labels \(\hat{z}\), \(n_k(\hat{z})\) is the number of nodes in community \(k\) according to \(\hat{z}\) and \(m_{k\ell}(\hat{z}) = n_k(\hat{z})(n_{\ell}(\hat{z}) - 1\{k = \ell\})\)
eval_dcsbm_bic

Value

A list of result

B estimated connectivity matrix.

theta estimated node propensity parameter.

---

eval_dcsbm_bic Compute BIC score

Description

compute BIC score when fitting a DCSBM to network data

Usage

eval_dcsbm_bic(A, z, K, poi)

Arguments

A adjacency matrix

z label vector

K number of community in z

poi whether to use Poisson version of likelihood

Details

the BIC score is calculated by $-2 \times \log \text{likelihood} - K \times (K + 1) \times \log(n)$

Value

BIC score

References


See Also

eval_dcsbm_like, eval_dcsbm_loglr
eval_dcsbm_like  

*Log likelihood of a DCSBM (fast with poi = TRUE)*

Description

Compute the log likelihood of a DCSBM, using estimated parameters B, theta based on the given label vector.

Usage

```r
eval_dcsbm_like(A, z, poi = TRUE, eps = 1e-06)
```

Arguments

- `A` adjacency matrix
- `z` label vector
- `poi` whether to use Poisson version of likelihood
- `eps` truncation threshold for the Bernoulli likelihood, used when parameter phat is close to 1 or 0.

Details

The log likelihood is calculated by

\[
\ell(\hat{B}, \hat{\theta}, \hat{\pi}, \hat{z} | A) = \sum_i \log \hat{\pi}_{z_i} + \sum_{i<j} \phi(A_{ij}; \hat{\theta}, \hat{\theta}, \hat{B}_{\hat{z}_i,\hat{z}_j})
\]

where \(\hat{B}, \hat{\theta}\) is calculated by `estim_dcsbm`, \(\hat{\pi}_k\) is the proportion of nodes in community \(k\).

Value

log likelihood of a DCSBM

See Also

- `eval_dcsbm_loglr`, `eval_dcsbm_bic`
**Description**

Computes the log-likelihood ratio of one DCSBM relative to another, using estimated parameters B and theta based on the given label vectors.

**Usage**

```r
eval_dcsbm_loglr(A, labels, poi = TRUE, eps = 1e-06)
```

**Arguments**

- `A`: adjacency matrix
- `labels`: a matrix with two columns representing two different label vectors
- `poi`: whether to use Poisson version of likelihood (instead of Bernoulli)
- `eps`: truncation threshold for the Bernoulli likelihood, used when parameter phat is close to 1 or 0.

**Details**

The log-likelihood ratio is computed between two DCSBMs specified by the columns of `labels`. The function computes the log-likelihood ratio of the model with `labels[, 2]` w.r.t. the model with `labels[, 1]`. This is often used with two label vectors fitted using different number of communities (say K and K+1).

When `poi` is set to `TRUE`, the function uses fast sparse matrix computations and is scalable to large sparse networks.

**Value**

log-likelihood ratio

**See Also**

`eval_dcsbm_like`, `eval_dcsbm_bic`
extract_largest_cc  Extract largest component

Description
Extract the largest connected component of a network

Usage
extract_largest_cc(gr, mode = "weak")

Arguments
  gr  The network as an igraph object
  mode Type of connected component ("weak","strong")

Value
An igraph object

extract_low_deg_comp  Extract low-degree component

Description
Extract a low-degree connected component of a network

Usage
extract_low_deg_comp(g, deg_prec = 0.75, verb = FALSE)

Arguments
  g  The network as an igraph object
  deg_prec The cut-off degree percentile
  verb Whether to be verbose (TRUE|FALSE)

Value
An igraph object
Description

The Conditional Pseudo-Likelihood (CPL) algorithm for fitting degree-corrected block models

Usage

```r
fast_cpl(Amat, K, ilabels = NULL, niter = 10)
```

Arguments

- `Amat`: adjacency matrix of the network
- `K`: desired number of communities
- `ilabels`: initial label vector (if not provided, initial labels are estimated using `spec_clust`)
- `niter`: number of iterations

Details

The function implements the CPL algorithm as described in the paper below. It relies on the `mixtools` package for fitting a mixture of multinomials to a block compression of the adjacency matrix based on the estimated labels and then reiterates.

Technically, `fast_cpl` fits a stochastic block model (SBM) conditional on the observed node degrees, to account for the degree heterogeneity within communities that is not modeled well in SBM. CPL can also be used to effectively estimate the parameters of the degree-corrected block model (DCSBM).

The code is an adaptation of the original R code by Aiyou Chen with slight simplifications.

Value

Estimated community label vector.

References


Examples

```r
head(fast_cpl(igraph::as_adj(polblogs), 2), 50)
```
Description

Samples an adjacency matrix from a stochastic block model (SBM)

Usage

```
fast_sbm(z, B)
```

Arguments

- `z`: Node labels (n * 1)
- `B`: Connectivity matrix (K * K)

Details

The function implements a fast algorithm for sampling sparse SBMs, by only sampling the necessary nonzero entries. This function is adapted almost verbatim from the original code by Aiyou Chen.

Value

An adjacency matrix following SBM

Examples

```
B = pp_conn(n = 10^4, oir = 0.1, lambda = 7, pri = rep(1,3))$B
head(fast_sbm(sample(1:3, 10^4, replace = TRUE), B))
```

Description

Generate randomly permuted connectivity matrix with a given average expected degree

Usage

```
gen_rand_conn(n, K, lambda, gamma = 0.3, pri = rep(1, K)/K, theta = rep(1, n))
```
**get_dcsbm_exav_deg**

**Arguments**

- `n`: number of nodes
- `K`: number of communities
- `lambda`: expected average degree
- `gamma`: a measure of out-in-ratio (convex combination parameter)
- `pri`: the prior on community labels
- `theta`: node connection propensity parameter of DCSBM, by default E(theta) = 1

**Details**

The connectivity matrix is a convex combination of a random symmetric permutation matrix and the matrix of all ones, with weights gamma and 1-gamma.

**Value**

connectivity matrix B of the desired DCSBM.

---

**get_dcsbm_exav_deg**  
*Calculate the expected average degree of a DCSBM*

---

**Description**

Calculate the expected average degree of a DCSBM

**Usage**

```r
get_dcsbm_exav_deg(n, pri, B, ex_theta = 1)
```

**Arguments**

- `n`: number of nodes
- `pri`: distribution of node labels (K x 1)
- `B`: connectivity matrix (K x K)
- `ex_theta`: expected value of theta

**Value**

expected average degree of a DCSBM
label_mat2vec

Convert label matrix to vector

Description
Convert label matrix to vector

Usage
label_mat2vec(Z)

Arguments
Z a cluster assignment matrix

Value
A label vector that follows the assignment matrix

label_vec2mat

Convert label vector to matrix

Description
Convert label vector to matrix

Usage
label_vec2mat(z, K = NULL, sparse = FALSE)

Arguments
z a label vector
K number of labels in z
sparse whether the output should be sparse matrix

Value
A cluster assignment matrix that follows from the label vector z
nac_test

Description

The NAC test to measure the goodness-of-fit of the DCSBM to network data. The function computes the NAC+ or NAC statistics in the paper below. Label vectors, if not provided, are estimated using `spec_clust` by default but one can also use any other community detection algorithms through `cluster_fct`. Note that the function has to have \( A \) and \( K \) as its first two arguments, and additional arguments could be provided through `...`.

Usage

```r
nac_test(A, K, z = NULL, y = NULL, plus = TRUE, cluster_fct = spec_clust, ...)
```

Arguments

- \( A \) : adjacency matrix.
- \( K \) : number of communities.
- \( z \) : label vector for rows of \( A \). If not provided, will be estimated from `cluster_fct`.
- \( y \) : label vector for columns of \( A \). If not provided, will be estimated from `cluster_fct`.
- `plus` : whether or not use column label vector with \( (K+1) \) communities, default is `TRUE`.
- `cluster_fct` : community detection function to get \( z \) and \( y \), by default using `spec_clust`. The first two arguments have to be \( A \) and \( K \).
- `...` : additional arguments for `cluster_fct`.

Value

A list of result

- `stat` : NAC or NAC+ test statistic.
- `z` : row label vector.
- `y` : column label vector.

References


See Also

- `snac_test`
Examples

A <- sample_dcpp(500, 10, 4, 0.1)$adj
nac_test(A, K = 4)$stat
nac_test(A, K = 4, cluster_fct = fast_cpl)$stat

---

plot_deg_dist  

**Plot degree distribution**

Description

Plot the degree distribution of a network on log scale

Usage

plot_deg_dist(gr, logx = TRUE)

Arguments

- **gr**: the network as an igraph object
- **logx**: whether the degree is in log scale.

Value

Histogram of the degree of 'gr'.

---

plot_net  

**Plot a network**

Description

Plot a network using degree-modulated node sizes, community colors and other enhancements

Usage

plot_net(
  gr,
  community = NULL,
  color_map = NULL,
  extract_lcc = TRUE,
  heavy_edge_deg_perc = 0.97,
  coord = NULL,
  vsize_func = function(deg) log(deg + 3) * 1,
  vertex_border = FALSE,
  niter = 1000,
  vertex_alpha = 0.4,
)
plot_roc

remove_loops = TRUE,
make_simple = FALSE,
...
)

Arguments

gr the network as an igraph object
community community assignment; vector of node labels
color_map color palette for clusters in 'gr'
extact_lcc Extract largest connected component or not
heavy_edge_deg_perc Degree percentile threshold for determining heavy edges
coord Optional starting positions for the vertices. If this argument is not NULL then it should be an appropriate matrix of starting coordinates.
vsize_func function to determine the size of node size
vertex_border whether to show the border of vertex or not
niter number of iteration for FR layout computation
vertex_alpha factor modifying the opacity alpha of vertex; typically in [0,1]
remove_loops whether to remove loops in the network
make_simple whether to simplify edge weight calculation
... other settings

Value

A network plot

plot_roc  
Plot ROC curves

Description

Plot ROC curves given results from simulate_roc.

Usage

plot_roc(roc_results, method_names = NULL, font_size = 16)

Arguments

roc_results data frame roc from the output list of simulate_roc
method_names a list of method names
font_size font size of the plot

Value

Roc plot based on results from simulate_roc
plot_smooth_profile  
*Plot community profiles*

**Description**

Plot the smooth community profiles based on a resampled statistic.

**Usage**

```r
plot_smooth_profile(
  tstat,
  net_name = "",
  trunc_type = "none",
  spar = 0.3,
  plot_null_spar = TRUE,
  alpha = 0.3,
  base_font_size = 12
)
```

**Arguments**

- **tstat**: dataframe that has a column `value` as statistic in the plot and a column `K` as its corresponding community number.
- **net_name**: name of network.
- **trunc_type**: method to round the dip/elbow point as the estimated community number.
- **spar**: the sparsity level of fitting spline to the value of `tstat`.
- **plot_null_spar**: whether to plot the spline with zero sparsity.
- **alpha**: transparency of the points in the plot.
- **base_font_size**: font size of the plot.

**Value**

smooth profile plot of a network.

---

**polblogs**  
*Political blogs network*

**Description**

This is a directed network of hyperlinks between political blogs about politics in the United States of America.
**pp_conn**

**Usage**

```r
data(polblogs)
```

**Format**

An igraph data with 1490 nodes and 19090 edges

**References**


---

**pp_conn**  Generate planted partition (PP) connectivity matrix

---

**Description**

Create a degree-corrected planted partition connectivity matrix with a given average expected degree.

**Usage**

```r
pp_conn(
  n,
  oir,
  lambda,
  pri,
  theta = rep(1, n),
  normalize_theta = FALSE,
  d = rep(1, length(pri))
)
```

**Arguments**

- `n` the number of nodes
- `oir` out-in-ratio
- `lambda` the expected average degree
- `pri` the prior on community labels
- `theta` node connection propensity parameter of DCSBM
- `normalize_theta` whether to normalize theta so that max(theta) == 1
- `d` diagonal of the connectivity matrix. An all-one vector by default.

**Value**

The connectivity matrix B of the desired DCSBM.
**printf**  
*The usual "printf" function*

---

**Description**

The usual "printf" function

**Usage**

`printf(...)`

**Arguments**

...  
printing object

**Value**

the value of the printing object

---

**rsymperm**  
*Generate random symmetric permutation matrix*

---

**Description**

Generate a random symmetric permutation matrix (recursively)

**Usage**

`rsymperm(K)`

**Arguments**

K  
size of the matrix

**Value**

A random K x K symmetric permutation matrix
**sample_dcer**  

Sample from a DCER

---

**Description**

Sample an adjacency matrix from a degree-corrected Erdős–Rényi model (DCER).

**Usage**

```r
sample_dcer(theta)
```

**Arguments**

- `theta`: Node connectivity propensity vector (n * 1)

**Value**

An adjacency matrix following DCSBM

---

**sample_dclvm**  

Sample from a DCLVM

---

**Description**

A DCLVM with K clusters has edges generated as

$$E[A_{ij} | x, \theta] \propto \theta_i \theta_j e^{-\|x_i - x_j\|^2}$$

where $x_i = 2e_{z_i} + w_i$, $e_k$ is the kth basis vector of $\mathbb{R}^d$, $w_i \sim N(0, I_d)$, and $\{z_i\} \subset [K]^n$. The proportionality constant is chosen such that the overall network has expected average degree $\lambda$.

To calculate the scaling constant, we approximate $E[e^{-\|x_i - x_j\|^2}]$ for $i \neq j$ by generating random npairs $\{z_i, z_j\}$ and average over them.

**Usage**

```r
sample_dclvm(z, lambda, theta, npairs = NULL)
```

**Arguments**

- `z`: a vector of cluster labels
- `lambda`: desired average degree of the network
- `theta`: degree parameter
- `npairs`: number of pairs of $\{z_i, z_j\}$
sample_dcpp

Details
Sample form a degree-corrected latent variable model with Gaussian kernel

Value
Adjacency matrix of DCLVM

Description
Sample from a degree-corrected planted partition model

Usage
sample_dcpp(
  n, lambda, K, oir,
  theta = NULL, pri = rep(1, K)/K,
  normalize_theta = FALSE
)

Arguments
n number of nodes
lambda average degree
K number of communities
oir out-in ratio
theta propensity parameter, if not given will be samples from a Pareto distribution with scale parameter 2/3 and shape parameter 3
pri prior distribution of node labels
normalize_theta whether to normalize theta so that max(theta) == 1

Value
an adjacency matrix following a degree-corrected planted partition model

See Also
sample_dcsbm, sample_tdcsvm
**sample_dcsbm**  
*Sample from a DCSBM*

**Description**  
Sample an adjacency matrix from a degree-corrected block model (DCSBM)

**Usage**  
```r
sample_dcsbm(z, B, theta = 1)
```

**Arguments**
- `z`: Node labels ($n \times 1$)
- `B`: Connectivity matrix ($K \times K$)
- `theta`: Node connectivity propensity vector ($n \times 1$)

**Value**  
An adjacency matrix following DCSBM

**See Also**
- `sample_dcpp`, `fast_sbm`, `sample_tdcsbm`

**Examples**
```r
B = pp_conn(n = 10^3, oir = 0.1, lambda = 7, pri = rep(1,3))$B
def(sample_dcsbm(sample(1:3, 10^3, replace = TRUE), B, theta = rexp(10^3)))
```

---

**sample_tdcsbm**  
*Sample truncated DCSBM (fast)*

**Description**  
Sample an adjacency matrix from a truncated degree-corrected block model (DCSBM) using a fast algorithm.

**Usage**  
```r
sample_tdcsbm(z, B, theta = 1)
```

**Arguments**
- `z`: Node labels ($n \times 1$)
- `B`: Connectivity matrix ($K \times K$)
- `theta`: Node connectivity propensity vector ($n \times 1$)
Details

The function samples an adjacency matrix from a truncated DCSBM, with entries having Bernoulli distributions with mean

\[ E[A_{ij}|z] = B_{z_i, z_j} \min(1, \theta_i \theta_j). \]

The approach uses the masking idea of Aiyou Chen, leading to fast sampling for sparse networks. The masking, however, truncates \( \theta_i \theta_j \) to at most 1, hence we refer to it as the truncated DCSBM.

Value

An adjacency matrix following DCSBM

Examples

```r
B = pp_conn(n = 10^4, oir = 0.1, lambda = 7, pri = rep(1,3))$B
head(sample_tdcsbm(sample(1:3, 10^4, replace = TRUE), B, theta = rexp(10^4)))
```

simulate_roc  
Simulate data to estimate ROC curves

Description

Simulate data from the null and alternative distributions to estimate ROC curves for a collection of methods.

Usage

```r
simulate_roc(
  apply_methods,
  gen_null_data,
  gen_alt_data,
  nruns = 100,
  core_count = parallel::detectCores() - 1,
  seed = NULL
)
```

Arguments

- `apply_methods`: a function that returns a data.frame with columns "method", "tstat" and "twosided"
- `gen_null_data`: a function that generate data under the null model
- `gen_alt_data`: a function that generate data under the alternative model
- `nruns`: number of simulated data from the null/alternative model
- `core_count`: number of cores used in parallel computing
- `seed`: seed for random simulation
sinkhorn_knopp

Description

Implements the Sinkhorn–Knopp algorithm for transforming a square matrix with positive entries to a stochastic matrix with given common row and column sums (e.g., a doubly stochastic matrix).

Usage

```r
sinkhorn_knopp(
  A,
  sums = rep(1, nrow(A)),
  niter = 100,
  tol = 1e-08,
  sym = FALSE,
  verb = FALSE
)
```

Arguments

- `A`: input matrix
- `sums`: desired row/column sums
- `niter`: number of iterations
- `tol`: convergence tolerance
- `sym`: whether to compute symmetric scaling \( D A D \)
- `verb`: whether to print the current change

Details

Computes diagonal matrices \( D_1 \) and \( D_2 \) to make \( D_1 A D_2 \) into a matrix with given row/column sums. For a symmetric matrix \( A \), one can set `sym = TRUE` to compute a symmetric scaling \( D A D \).

Value

Diagonal matrices \( D_1 \) and \( D_2 \) to make \( D_1 A D_2 \) into a matrix with given row/column sums.

Value

- `roc`: A data frame used to plot ROC curves with columns: method, whether a two sided test, false positive rate (FPR), and true positive rate (TPR)
- `raw`: A data frame containing raw output from null and alternative models with columns: method, statistics value, whether a two sided test, and the type of hypothesis
- `elapsed_time`: system elapsed time for generating ROC data
**snac_resample**  
*Resampled SNAC*

### Description
Compute SNAC+ with resampling

### Usage
```
snac_resample(
  A,
  nrep = 20,
  Kmin = 1,
  Kmax = 13,
  ncores = parallel::detectCores() - 1,
  seed = 1234
)
```

### Arguments
- **A**  
  adjacency matrix
- **nrep**  
  number of times SNAC+ is computed
- **Kmin**  
  minimum community number to use in SNAC+
- **Kmax**  
  maximum community number to use in SNAC+
- **ncores**  
  number of cores to use in the parallel computing
- **seed**  
  seed for random sampling

### Value
A data frame with columns specifying repetition cycles, number of community numbers and the value of SNAC+ statistics

---

**snac_select**  
*Estimate community number with SNAC*

### Description
Applying SNAC+ test sequentially to estimate community number of a network fit to DCSBM
Usage

```r
snac_select(
  A,
  Kmin = 1,
  Kmax,
  alpha = 1e-05,
  labels = NULL,
  cluster_fct = spec_clust,
  ...
)
```

Arguments

- **A**: adjacency matrix.
- **Kmin**: minimum candidate community number.
- **Kmax**: maximum candidate community number.
- **alpha**: significance level for rejecting the null hypothesis.
- **labels**: a matrix with each column being a row label vector for a candidate community number. If not provided, will be computed by `cluster_fct`.
- **cluster_fct**: community detection function to get label vectors to compute SNAC+ statistics (in `snac_test`), by default using `spec_clust`.
- **...**: additional arguments for `cluster_fct`.

Value

estimated community number.

See Also

- `snac_test`

Examples

```r
A <- sample_dcpp(500, 10, 3, 0.1)$adj
snac_select(A, Kmax = 6)
```

Description

The SNAC test to measure the goodness-of-fit of the DCSBM to network data. The function computes the SNAC+ or SNAC statistics in the paper below. The row label vector of the adjacency matrix could be given through `z` otherwise will be estimated by `cluster_fct`. One can specify the ratio of nodes used to estimate column label vector. If `plus = TRUE`, the column labels will be estimated by `spec_clust` with `(K+1)` clusters, i.e. performing SNAC+ test, otherwise with `K` clusters SNAC test. One can also get multiple test statistics with repeated random subsampling on nodes.
Usage

`snac_test`

```r
snac_test(
    A,
    K,
    z = NULL,
    ratio = 0.5,
    fromEachCommunity = TRUE,
    plus = TRUE,
    cluster_fct = spec_clust,
    nrep = 1,
    ...
)
```

Arguments

- `A` : adjacency matrix.
- `K` : desired number of communities.
- `z` : label vector for rows of adjacency matrix. If not provided, will be estimated by `cluster_fct`.
- `ratio` : ratio of subsampled nodes from the network.
- `fromEachCommunity` : whether subsample from each estimated community or the full network, default is TRUE.
- `plus` : whether or not use column label vector with (K+1) communities to compute the statistics, default is TRUE.
- `cluster_fct` : community detection function to estimate label vectors, by default using `spec_clust`. The first two arguments have to be A and K.
- `nrep` : number of times the statistics are computed.
- `...` : additional arguments for `cluster_fct`.

Value

A list of result

- `stat` : SNAC or SNAC+ test statistic.
- `z` : row label vector.

References


See Also

`nac_test`
spec_clust

Examples

A <- sample_dcpp(500, 10, 4, 0.1)$adj
snac_test(A, K = 4, niter = 3)$stat

---

**spec_clust**

*Spectral clustering (fast)*

**Description**

Perform spectral clustering (with regularization) to estimate communities

**Usage**

```r
spec_clust(
  A,
  K,
  type = "lap",
  tau = 0.25,
  nstart = 20,
  niter = 10,
  ignore_first_col = FALSE
)
```

**Arguments**

- **A**: Adjacency matrix (n x n)
- **K**: Number of communities
- **type**: ("lap" | "adj" | "adj2") Whether to use Laplacian or adjacency-based spectral clustering
- **tau**: Regularization parameter for the Laplacian
- **nstart**: argument from function 'kmeans'
- **niter**: argument from function 'kmeans'
- **ignore_first_col**: whether to ignore the first eigen vector when doing spectral clustering

**Value**

A label vector of size n x 1 with elements in 1,2,...,K
Spectral Representation

Description

Provides a spectral representation of the network (with regularization) based on the adjacency or Laplacian matrices

Usage

```
spec_repr(A, K, type = "lap", tau = 0.25, ignore_first_col = FALSE)
```

Arguments

- **A**: Adjacency matrix (n x n)
- **K**: Number of communities
- **type**: ("lap" | "adj" | "adj2") Whether to use Laplacian or adjacency-based spectral clustering
- **tau**: Regularization parameter for the Laplacian
- **ignore_first_col**: whether to ignore the first eigen vector

Value

The n x K matrix resulting from a spectral embedding of the network into R^K
Index

* comm_detect
  fast_cpl, 11
  spec_clust, 29
* datasets
  polblogs, 18
* estimation
  compute_block_sums, 5
  estim_dcsbm, 6
  eval_dcsbm_like, 8
* evaluation
  compute_confusion_matrix, 5
  compute_mutual_info, 6
  simulateroc, 24
* mod_sel
  bethe_hessian_select, 4
  eval_dcsbm_bic, 7
  eval_dcsbm_loglr, 9
  snac_select, 26
* models
  fast_sbm, 12
  gen_rand_conn, 12
  get_dcsbm_exav_deg, 13
  pp_conn, 19
* net_repr
  spec_repr, 30
* plotting
  plot_roc, 17
* utils
  extract_largest_cc, 10
  extract_low_deg_comp, 10
  label_mat2vec, 14
  label_vec2mat, 14
  printf, 20
  rsymperr, 20
  sinkhorn_knopp, 25
  adj_spec_test, 3
  bethe_hessian_select, 4
  compute_block_sums, 5
  compute_confusion_matrix, 5
  compute_mutual_info, 6
  estim_dcsbm, 3, 6, 8
  eval_dcsbm_bic, 7, 8, 9
  eval_dcsbm_like, 7, 8, 9
  eval_dcsbm_loglr, 7, 8, 9
  extract_largest_cc, 10
  extract_low_deg_comp, 10
  fast_cpl, 11
  fast_sbm, 12, 23
  gen_rand_conn, 12
  get_dcsbm_exavg_deg, 13
  label_mat2vec, 14
  label_vec2mat, 14
  nac_test, 15, 28
  plot_deg_dist, 16
  plot_net, 16
  plot_roc, 17
  plot_smooth_profile, 18
  polblogs, 18
  pp_conn, 19
  printf, 20
  rsymperr, 20
  sample_dcer, 21
  sample_dclvm, 21
  sample_dcpp, 22, 23
  sample_dcsbm, 22, 23
  sample_tdcsbm, 22, 23, 23
  simulate_roc, 17, 24
  sinkhorn_knopp, 25
  snac_resample, 26
  snac_select, 26
snac_test, 15, 27, 27
spec_clust, 3, 11, 15, 27, 28, 29
spec_repr, 30