Package 'simone'

February 4, 2019

Version 1.0-4

Date 2019-02-06

Title Statistical Inference for MOdular NEtworks (SIMoNe)

Maintainer Julien Chiquet <julien.chiquet@inra.fr>

Depends R (>= 3.1.1), blockmodels

Description Implements the inference of
co-expression networks based on partial correlation
coefficients from either steady-state or time-course
transcriptomic data. Note that with both type of data this
package can deal with samples collected in different
experimental conditions and therefore not identically
distributed. In this particular case, multiple but related
networks are inferred on one simone run.

License GPL (>= 2)

URL http://julien.cremeriefamily.info/simone.html

Encoding UTF-8

Repository CRAN

Date/Publication 2019-02-03 23:10:03 UTC

RoxygenNote 5.0.1

NeedsCompilation yes

Author Julien Chiquet [aut, cre] (<https://orcid.org/0000-0002-3629-3429>),
Gilles Grasseau [aut],
Christophe Ambroise [aut],
Camille Charbonnier [ctb],
Alexander Smith [ctb],
Catherine Matias [ctb]

R topics documented:

`simone-package` *Statistical Inference for MOdular NEtworks (SIMoNe)*

Description

The `R` package **simone** implements the inference of co-expression networks based on partial correlation coefficients from either steady-state or time-course transcriptomic data. Note that with both type of data this package can deal with samples collected in different experimental conditions and therefore not identically distributed. In this particular case, multiple but related graphs are inferred at once.

The underlying statistical tools enter the framework of Gaussian graphical models (GGM). Basically, the algorithm searches for a latent clustering of the network to drive the selection of edges through an adaptive ℓ_1-penalization of the model likelihood.

The available inference methods for edges selection and/or estimation include

neighborhood selection as in Meinshausen and Buhlman (2006), steady-state data only;

graphical Lasso as in Banerjee *et al*, 2008 and Friedman *et al* (2008), steady-state data only;

VAR(1) inference as in Charbonnier, Chiquet and Ambroise (2010), time-course data only;

multitask learning as in Chiquet, Grandvalet and Ambroise (preprint), both time-course and steady-state data.

All the listed methods are ℓ_1-norm based penalization, with an additional grouping effect for multitask learning (including three variants: "intertwined", "group-Lasso" and "cooperative-Lasso").

The penalization of each individual edge may be weighted according to a latent clustering of the network, thus adapting the inference of the network to a particular topology. The clustering algorithm is performed by the `mixer` package, based upon Daudin, Picard and Robin (2008)'s Mixture Model for Random Graphs.

Since the choice of the network sparsity level remains a current issue in the framework of sparse Gaussian network inference, the algorithm provides a full path of estimates starting from an empty network and adding edges as the penalty level progressively decreases. *Bayesian Information Criteria* (BIC) and *Akaike Information Criteria* (AIC) are adapted to the GGM context in order to help to choose one particular network among this path of solutions.

Graphical tools are provided to summarize the results of a `simone` run and offer various representations for network plotting.

Details

Index:

```
cancer                Microarray data set for breast cancer
coNetwork             Random perturbations of a reference network
getNetwork            Network extraction from a SIMoNe run
plot.simone           Graphical representation of SIMoNe outputs
plot.simone.network   Graphical representation of a network
rNetwork              Simulation of (clustered) Gaussian networks
rTranscriptData       Simulation of artificial transcriptomic data
setOptions            Low-level options of the 'simone' function
simone                SIMoNe algorithm for network inference
```

Demos available

Beyond the examples of this manual, a good starting point is to have a look at the scripts available via `demo(package="simone")`. They make use of `simone`, main function in the package, in various contexts (steady-state or time-course data, multiple sample learning). All these scripts also illustrate the use of the different plot functions.

`demo(cancer_multitask)` example on the `cancer` data set of the multitask approach with a cooperative-Lasso grouping effect across tasks. Patient responses to the chemiotherapy (pCR or not-pCR) split the data set into two distinct samples. Network inference is performed jointly on these samples and graphical comparison is made between the two networks.

`demo(cancer_pooled)` example on the `cancer` data set which is designed to compare network inference when a clustering prior is used or not. Graphical comparison between the two inferred networks (with/without clustering prior) illustrates how inference is driven to a particular network topology when clustering is relevant (here, an affiliation structure).

`demo(check_glasso, echo=FALSE)` example that basically checks the consistency between the **glasso** package of Friedman *et al* and the **simone** package to solve the ℓ_1-penalized Gaussian likelihood criterion suggested by Banerjee *et al* in the $n > p$ settings. In the $n < p$ settings, **simone** provides sparser solutions than the **glasso** package since the underlying Lasso problems are solved with an active set algorithm instead of the shooting/pathwise coordinate algorithm.

`demo(simone_multitask)` example of multitask learning on simulated, steady-state data: two networks are generated by randomly perturbing a common ancestor with the `coNetwork` function. These two networks are then used to generate two multivariate Gaussian samples. Multitask learning is applied and a simple illustration of the use of the `setOptions` function is given.

`demo(simone_steadyState)` example of how to learn a single network from steady-state data. A sample is first generated with the `rNetwork` and `rTranscriptData` functions. Then the path of solutions of the *neighborhood selection* method (default for single task steady-state data) is computed.

`demo(simone_timeCourse)` example of how to learn a single network from time-course data. A sample is first generated with the `rNetwork` and `rTranscriptData` functions and the path of solutions of the *VAR(1) inference* method is computed, with and without clustering prior.

Author(s)

- Julien Chiquet <julien.chiquet@genopole.cnrs.fr>,
- Gilles Grasseau <gilles.grasseau@genopole.cnrs.fr>,
- Camille Charbonnier <camille.charbonnier@genopole.cnrs.fr>,
- Christophe Ambroise <christophe.ambroise@genopole.cnrs.fr>.

References

J. Chiquet, Y. Grandvalet, and C. Ambroise (preprint). Inferring multiple graphical structures. *preprint available on ArXiv*. http://arxiv.org/abs/0912.4434.

C. Charbonnier, J. Chiquet, and C. Ambroise (2010). Weighted-Lasso for Structured Network Inference from Time Course Data. *Statistical Applications in Genetics and Molecular Biology*, vol. 9, iss. 1, article 15. http://www.bepress.com/sagmb/vol9/iss1/art15/

C. Ambroise, J. Chiquet, and C. Matias (2009). Inferring sparse Gaussian graphical models with latent structure. *Electronic Journal of Statistics*, vol. 3, pp. 205–238. http://dx.doi.org/10.1214/08-EJS314

O. Banerjee, L. El Ghaoui, A. d'Aspremont (2008). Model Selection Through Sparse Maximum Likelihood Estimation. *Journal of Machine Learning Research*, vol. 9, pp. 485–516. http://www.jmlr.org/papers/volume9/banerjee08a/banerjee08a.pdf

J. Friedman, T. Hastie and R. Tibshirani (2008). Sparse inverse covariance estimation with the graphical Lasso. *Biostatistics*, vol. 9(3), pp. 432–441. http://www-stat.stanford.edu/~tibs/ftp/graph.pdf

N. Meinshausen and P. Buhlmann (2006). High-dimensional graphs and variable selection with the Lasso. *The Annals of Statistics*, vol. 34(3), pp. 1436–1462. http://projecteuclid.org/DPubS/Repository/1.0/Disseminate?view=body&id=pdfview_1&handle=euclid.aos/1152540754

J.-J. Daudin, F.Picard and S. Robin, S. (2008). Mixture model for random graphs. *Statistics and Computing*, vol. 18(2), pp. 173–183. http://www.springerlink.com/content/9v6846342mu82x42/fulltext.pdf

---

`cancer` *Microarray data set for breast cancer*

Description

This gene expression data set is freely available, coming from the Hess *et al*'s paper. It concerns one hundred thirty-three patients with stage I–III breast cancer. Patients were treated with chemotherapy prior to surgery. Patient response to the treatment can be classified as either a pathologic complete response (pCR) or residual disease (not-pCR). Hess *et al* developed and tested a reliable multigene predictor for treatment response on this data set, composed by a set of 26 genes having a high predictive value.

The dataset splits into 2 parts (pCR and not pCR), on which network inference algorithms should be applied independently or in the multitask framework: only individuals from the same classes should be consider as independent and identically distributed.

Usage

```
data(cancer)
```

Format

A list named `cancer` comprising two objects:

`expr` a `data.frame` with 26 columns and 133 rows. The nth row gives the expression levels of the 26 identified genes for the nth patient. The columns are named according to the genes.

`status` a factor of size 133 with 2 levels (`"pcr"` and `"not"`), describing the status of the patient.

References

K.R. Hess, K. Anderson, W.F. Symmans, V. Valero, N. Ibrahim, J.A. Mejia, D. Booser, R.L. Theriault, U. Buzdar, P.J. Dempsey, R. Rouzier, N. Sneige, J.S. Ross, T. Vidaurre, H.L. Gomez, G.N. Hortobagyi, and L. Pustzai (2006). Pharmacogenomic predictor of sensitivity to preoperative chemotherapy with Paclitaxel and Fluorouracil, Doxorubicin, and Cyclophosphamide in breast cancer, *Journal of Clinical Oncology*, vol. 24(26), pp. 4236–4244.

Examples

```
## load the breast cancer data set
data(cancer)
attach(cancer)

## histogram of gene expression levels
par(mfrow=c(1,2))
hist(as.matrix(expr[status == "pcr",]), main="pCR")
hist(as.matrix(expr[status == "not",]), main="not pCR")

## mean of gene expression levels for pCR and not-pCR
colMeans( expr[ which( status=="not"), ] )
colMeans( expr[ which( status=="pcr"), ] )
detach(cancer)
```

---

coNetwork *Random perturbations of a reference network*

Description

Simulates a network from another network object by randomly perturbing a given number of edges.

Usage

```
coNetwork(graph,
          delta,
          name = "a co-network")
```

Arguments

graph	an object of class simone.network (typically generated by the rNetwork function).
delta	an integer giving the number of edges to randomly remove AND add to graph in order to obtain a randomly perturbed network.
name	a character string indicating the name of the perturbed network.

Value

Returns an object of class simone.network, see rNetwork for further details.

Author(s)

J. Chiquet

See Also

rNetwork, plot.simone.network.

Examples

```
## ancestor and child network generation
ancestor <- rNetwork(p = 20, pi = 20, name = "ancestor")
child    <- coNetwork(ancestor, delta = 1, name = "child")

# network comparison
plot(ancestor, child)
```

---

getNetwork *Network extraction from a SIMoNe run*

Description

When running simone, a family of networks is generated and one may want to pick one of them according to a given criterion. This is the job handled by the getNetwork function.

Usage

```
getNetwork(object,
           selection = length(object$clusters),
           nodes     = NULL)
```

Arguments

object	output of a SIMoNe run (must be an object of class simone)
selection	either a character string ("BIC" or "AIC") or an integer (a number of edges) that specifies how the network is selected from the list generated by simone. When a number num of edges is specified, the network is extracted by picking from the list the network with at most num edges. The effective number num will be displayed. Default is to extract a number of edges at most equal to the number of variables.
nodes	a vector of character string or integers used to extract a sub-part of the selected network, which can be more readable. When NULL (the default), all the variables are kept.

Value

Returns an object of class simone.network, see rNetwork for further details.

Author(s)

J. Chiquet

See Also

simone, rNetwork, plot.simone.network.

Examples

```
## load the breast cancer data set
data(cancer)
attach(cancer)

## launch SIMoNe on the full data set
res <- simone(expr)

## the default selected network (at most p edges)
plot(getNetwork(res))

## a sub network on some 10 randomly selected genes
plot(getNetwork(res,"BIC", nodes = sample(colnames(expr),10)))

## a network with a penalty corresponding to at most 40 edges
plot(getNetwork(res, 40))

detach(cancer)
```

plot.simone *Graphical representation of SIMoNe outputs*

Description

Plots various outputs associated to a SIMoNe run.

Usage

```
## S3 method for class 'simone'
plot(x,
     output    = c("BIC", "AIC", "ROC", "PR", "path.edges",
                   "path.penalty", "sequence"),
     ref.graph = NULL,
     ask       = TRUE,  ...)
```

Arguments

x	output of a simone run (must be an object of class simone)
output	a vector of character string indicating which outputs must be plotted (picken from "BIC", "AIC", "ROC", "PR", "path.edges", "path.penalty" or "sequence"). Default is to plot everything possible.
ref.graph	a network of reference provided through an adjacency matrix that is used to compute the ROC and PR curves. Only required if "ROC" and "PR" belongs to the output argument.
ask	a logical indicating if the graphics device should be interactive. Default is TRUE.
...	Additonal arguments for generic plot (such as main = "my title").

Details

Here are some details about the plots possibly produced:

- If "BIC" belongs to the output argument, a plot representing the Bayesian Information Criterion as a function of each network inferred by simone is displayed.
- If "AIC" belongs to the output argument, a ploy representing the Akaike Information Criterion as a function of each network inferred by simone is displayed.
- If "ROC" belongs to the output argument and ref.graph is specified, the ROC curve (Receiver Operating Characteristic) is plotted by representing true positive rate vs. false positive rate.
- If "PR" belongs to the output argument and ref.graph is provided by the user, the PR curve (Precision/Recall) is plotted by representing positive predicted values vs. true positive rate.
- If "path.penalty" belongs to the output argument, a regularization path is plotted by representing the value of each entry of the Theta matrix (that is, of each edge) vs. the penalty level λ: there are as many values for the penalty as networks stocked in the simone object x.

- If "path.edges" belongs to the output argument, a regularization path is plotted by representing the value of each entry of the Theta matrix (that is, of each edge) vs. the degree of freedom in Theta (that is, the number of edges in the current network). This is done for all the network stocked in the simone object x.
- If "sequence" belongs to the output argument, an interactive plot is provided by starting from the empty network and adding the edges by successively covering the networks stocked in the simone object x.

Note

If the user asked for "PR" and "ROC" curves yet did not specify a network of reference, these curves will not be plotted (no warning or error will be specified).

Author(s)

J. Chiquet

See Also

simone.

Examples

```
## data set and network generation
g    <- rNetwork(p=50, pi=50)
data <- rTranscriptData(300,g)
attach(data)

## running simone
res <- simone(X, type="steady-state")

## plotting the results: just the ROC curve
plot(res, output=c("ROC"), ref.graph=g$A)

## plotting the results: just the path as a function of the penalty
plot(res, output=c("path.penalty"), main="I want to put my own title")

## plotting the results: everything possible (the default)
plot(res)

detach(data)
```

---

plot.simone.network *Graphical representation of a network*

Description

Displays the network contained in an object of class simone.network.

Usage

```
## S3 method for class 'simone.network'
plot(x,
     y    = NULL,
     type = "default", last.coord=FALSE, ...)
```

Arguments

x	an object of class simone.network to display.
y	an optional simone.network object to compare x with.
type	network display types (see also details) are
	"circle" displays the network nodes on a circle shape.
	"circles" displays the network nodes on circle shapes. The different circles correspond to node classes.
	"cluster" (default) displays network nodes (no underlying shape is used)
	"overlap" display a unique graph in which 2 graphs are overlaid
	"4graphs" (default) displays the two networks, the intersection and the symmetric difference between the two networks.
last.coord	use last node coordinates if TRUE.
...	additionnal parameters

Details

This function plots a graph representation from a simone.network object. When available, the classification vector describing a partition of nodes is represented.

Different node layouts (see type option) can be chosen to represent networks:

1. if a single simone.network object is provided, the available layouts are cluster (the default), circle (nodes are laid on one circle) and circles (nodes are laid on several circles, one circle for a node class);
2. if two simone.network objects are provide, the available layouts are 4graphs (the default, which displays both networks as well as the intersection and the difference between them) and overlap (which overlay two networks, representing common edges in gray, edges present in the first network in blue and edges present in the second network in red).

Note

When comparing two networks, the network with the more numerous edges should be passed as the first argument of plot.simone.network since the node positions for both networks will be computed so as the first graph is as readable as possible.

Author(s)

G. Grasseau

See Also

plot.simone, simone

Examples

```
## data set and graph generation
lambda  <- 0.125
epsilon <- 0.00125
alpha <- c(1/3,1/3,1/3)

pi.affi <- matrix(epsilon,3,3)
diag(pi.affi) <- lambda

g1 <- rNetwork(p=200, pi=pi.affi, alpha=alpha)
g2 <- coNetwork(g1, delta=10)

plot(g1, type="cluster") # the default
plot(g1, type="circle" ) # one circle
plot(g1, type="circles" ) # one circle per cluster
plot(g1, g2, type="4graphs") # the default for multiple inputs
plot(g1, g2, type="overlap") # comparison of 2 networks on an unique graph
```

---

rNetwork *Simulation of (clustered) Gaussian networks*

Description

Simulates a network with various structures.

Usage

```
rNetwork(p,
         pi,
         alpha    = c(1),
         directed = FALSE,
         name     = "a network",
         signed   = TRUE)
```

Arguments

p	the number of nodes of the simulated network.
pi	a matrix of cluster connectivity (see details).
alpha	a vector of cluster proportions.
directed	a logical indicating the directedness of the network.
name	a character string indicating the name of the network.
signed	a logical indicating whether partial correlations should be signed or all kept positive.

Details

Matrix `pi` should be a square matrix of the same size as vector `alpha`. When the network is not directed, `pi` should be symmetric. When the graph is directed, entry $\pi_{q\ell}$ corresponds to edges heading from class q to class ℓ.

Entries of `pi` can be either integers or real numbers. If they are integers, they are considered as the exact number of edges required from one class to another. Otherwise, they are considered as connectivity probabilities between classes. They should therefore sum up to at most 1. If they do not sum up to one excatly, the remaining value is considered as the probability for a node to belong to the dust class (connected to no other node).

Value

Returns an object of class `simone.network`, that is, a list comprising

`A`	the $p \times p$ adjacency matrix of the network, filled with 0 and 1's, which is symmetric if `directed` is `FALSE`.
`Theta`	a $p \times p$ matrix of parameters of the associated Gaussian model, which depends on the directedness of the network: if directed, `Theta` contains the parameters of a VAR(1) model; if undirected, `Theta` is the concentration matrix (inverse of the covariance matrix) of a Gaussian vector
`directed`	a logicial indicating the directedness of the network.
`clusters`	a size-p factor indicating the node class. The number of levels is determined by the number of columns of the matrix of connectivity `pi`: the levels are labeled $1, \ldots, Q$ where Q is the number of clusters.
`name`	a character string containing the name of the network.

Author(s)

J. Chiquet, C. Charbonnier

See Also

`coNetwork`, `plot.simone.network`.

Examples

```
## generate an Erdos-Renyi network with 50 nodes and Pr of edges = 0.1
plot(rNetwork(p = 50, pi = 0.1, name = "an Erdos-Renyi network"))

## generate an network with 15 nodes and 25 randomly selected edges
plot(rNetwork(p = 15, pi = 25, name = "a 25 edges network"))

## generate an undirected network with an affiliation structure
PI <- matrix(c(15,2,2,50),2,2)
alpha <- c(1/3,2/3)
plot(rNetwork(p = 20, pi = PI, alpha = alpha,
                      name = "Affiliation, fixed num of edges"))

## generate a directed network with hubs
```

```
PI <- t(matrix(c(0.2,0.1,0.4,0,0.05,0.15,0,0.4,rep(0,8)),4,4))
alpha <- c(1/20,1/20,9/20,9/20)
plot(rNetwork(p = 55, pi = PI, alpha = alpha, directed = TRUE,
                      name = "Hubs structured network"))
```

`rTranscriptData` *Simulation of artificial transcriptomic data*

Description

Simulates a Gaussian sample that mimics transcriptomic data, according to a given network, either steady-state or time-course data. When several networks are given, multiple samples are generated.

Usage

```
rTranscriptData(n,
                graph,
                ...,
                mu    = rep(0, p),
                sigma = 0.1)
```

Arguments

`n`	integer or vector of integer indicating the sample sizes of each task
`graph`	a `simone.network` object typically generated either by `rNetwork` or `coNetwork`
`...`	additional `simone.network` objects in case of multiple sample generation
`mu`	if the network(s) is(are) directed, `mu` is the offset of the VAR(1) model that is used to generate the time-course data; if undirected, `mu` is the offset of the Gaussian vector.
`sigma`	standard deviation of the noise term used in the simulation process

Details

If the network is directed, time-course data are simulated according to a VAR(1) model. If the network is undirected, steady-state data are generated by simulating an independent, identically distributed sample of a Gaussian vector.

In both cases, samples are generated on the basis of $\mathbf{\Theta}$, as provided by `graph$Theta`.

If the network is directed, samples are generated according to the following VAR(1) process:

$$\begin{cases} X_0 \sim \mathcal{N}(0, \sigma) \\ X_t \sim \mu + \mathbf{\Theta} X_{t-1} + \varepsilon_t \quad \text{for all } t \in 1, \ldots, n \\ \varepsilon_t \sim \mathcal{N}(0, \sigma) \end{cases}$$

If the network is undirected, samples are generated according to the following Gaussian vector:

$$\begin{cases} X_i \sim \mu + (\mathbf{\Theta}^{-1/2})^t U_i + \varepsilon_i \quad \text{for all } i \in 1, \ldots, n \\ U_i \sim \mathcal{N}(0, 1) \\ \varepsilon_i \sim \mathcal{N}(0, \sigma) \end{cases}$$

Numerically, $\mathbf{\Theta}^{-1/2}$ is computed with the Cholesky decomposition of the pseudo-inverse of $\mathbf{\Theta}$.

Value

Returns a list comprising :

`X`	matrix of simulated gene expression data, `n` observations in rows, genes in columns
`tasks`	factor indicating the tasks corresponding to the simulated gene expression data in case of multiple networks.

Author(s)

J. Chiquet, C. Charbonnier

See Also

`rNetwork`, `coNetwork`.

Examples

```
## time-Course data generation
##----------------------------
# generate a directed network
n <- 20
p <- 5
g <- rNetwork(p, pi=5, directed=TRUE)
# Generate the data, data2 noisier than data1
data1  <- rTranscriptData(n,g)
data2  <- rTranscriptData(n,g,sigma=1)
matplot(1:n, data1$X,type= "l", xlab = "time points",
        ylab = "level of expression", col=rainbow(n,start=2/6,end = 3/6),
        ylim = range(c(data1$X,data2$X)),
        main="data2 (blue) generated with more noise than data1 (green)")
matlines(1:n,data2$X,type= "l",col = rainbow(n,start=4/6,end=5/6))

## steady-state data generation
##----------------------------
# generate an undirected network
p <- 10
g <- rNetwork(p, pi=10)
data <- rTranscriptData(n=1000,g, sigma=0)
attach(data)
# Inference of Theta (here without dimension problems since p << n)
b <- sapply(1:p,function(x){
   tmp <- -solve(t(X[,-x]) %*% X[,-x]) %*% t(X[,-x]) %*% X[,x]
   res <- rep(NA,10)
   res[-x] <- tmp
   res[x] <- 1
   return(res)
  }
)
```

```
detach(data)
# comparison of theoretical Theta and inferred Theta
par(mfrow=c(1,2))
image(g$Theta, main = "Theoretical Theta")
image(b, main = "Inferred Theta")

## time-course multitask data generation
##--------------------------------------
# start by generating the networks
ancestor <- rNetwork(p=5, pi=5, name="ancestor", directed=TRUE)
child1   <- coNetwork(ancestor, 1, name = "child 1")
child2   <- coNetwork(ancestor, 1, name = "child 2")
# generate the data
n <- c(20,20)
data  <- rTranscriptData(n,child1,child2)
attach(data)
par(mfrow=c(2,1))
matplot(1:(n[1]),X[tasks ==1,],type= "l", main="Dataset from child 1",
        xlab = "time points", ylab = "level of expression")
matplot(1:(n[2]),X[tasks == 2,], type= "l", main="Dataset from child 2",
        xlab = "time points", ylab = "level of expression")
detach(data)
```

---

setOptions *Low-level options of a SIMoNe run*

Description

This function is intended to design low-level uses of SIMoNe by specifying various parameters of the underlying algorithms.

Usage

```
setOptions(normalize      = TRUE,
           verbose        = TRUE,
           penalties      = NULL,
           penalty.min    = NULL,
           penalty.max    = NULL,
           n.penalties    = 100,
           edges.max      = Inf,
           edges.sym.rule = NULL,
           edges.steady   = "neighborhood.selection",
           edges.coupling = "coopLasso",
           clusters.crit  = "BIC",
           clusters.meth  = "bayesian",
           clusters.qmin  = 2,
           clusters.qmax  = 4)
```

Arguments

normalize	logical specifying wether the data should be normalized to unit variance. The normalization is made task-wisely in the multiple sample setting. Default is TRUE.
verbose	a logical that indicates verbose mode to display progression. Default is TRUE.
penalties	vector of *decreasing* penalty levels for the network estimation. If NULL (the default), an appropriate vector will be generated in simone with n.penalties entries, starting from penalty.max and shrinked to penalty.min.
penalty.min	The minimal value of the penalty that will be tried for network inference. If NULL (the default), it will be set in simone to 1e-5 for the monotask framework and to 1e-2 for the multitask framework.
penalty.max	The maximal value of the penalty that will be tried for network inference. If NULL (the default), it will be set to a value that provoques an empty granph. Default is NULL.
n.penalties	integer that indicates the number of penalties to put in the penalties vector. Default is 100.
edges.max	integer giving an upper bound for the number of edges to select: if a network is inferred along the algorithm with a number of edges overstepping edges.max, it will stop there. Default is Inf.
edges.steady	a character string indicating the method to use for the network inference associated to steady-state data, one task framework. Either "graphical.lasso" or "neighborhood.selection". Default is the later.
edges.coupling	character string (either "coopLasso", "groupLasso" or "intertwined") that indicates the coupling method across task in the multiple sample setup. Defautl is "coopLasso".
edges.sym.rule	character string ("AND", "OR", "NO") for post-symmetrization of the infered networks. Enforced to "NO" for time-course data (directed network) and set to "AND" as default for steady-state data (undirected network).
clusters.crit	criterion to select the network that is used to find an underlying clustering. Either "BIC", "AIC" or an integer for the number of edges. Default is "BIC".
clusters.qmin	minimum number of classes for clustering. Default is 2.
clusters.qmax	maximum number of classes for clustering. Default is 4.
clusters.meth	character string indicating the strategy used for the estimation: "variational", "classification", or "bayesian". See the **mixer** package for further details. Default is "bayesian".

Value

A list that contains all the specified parameters.

Note

If the user specifies its own penalties vector, all the networks inferred during the algorithm will be kept, even if they share the very same number of edges.

On the other hand, if you only specify `penalty.max` and/or `penalty.min` and/or `n.penalties`, the algorithm will only kept the networks who show different numbers of edges. That is to say, the number of networks stocked in the output of `simone` generally does not have a length equal to `n.penalties`.

Author(s)

J. Chiquet

See Also

`simone`.

Examples

```
## generate an object (list) with the default parameters
setOptions()
```

`simone` *SIMoNe algorithm for network inference*

Description

The `simone` function offers an interface to infer networks based on partial correlation coefficients in various contexts and methods (steady-state data, time-course data, multiple sample setup, clustering prior)

Usage

```
simone(X,
       type       = "steady-state",
       clustering = FALSE,
       tasks      = factor(rep(1, nrow(X))),
       control    = setOptions())
```

Arguments

`X`	a $n \times p$ matrix of data, typically n expression levels associated to the same p genes. Can also be a `data.frame` with n entries, each column corresponding to a variable (a gene). Specifying `colnames` to `X` may be convenient in view of results analysis, since it will be used to annotate the plots. Note that this is the only required argument.
`type`	a character string indicating the data specification (either `"steady-state"` or `"time-course"` data). Default is `"steady-state"`.
`clustering`	a logical indicating if the network inference should be perfomed by penalizing the edges according to a latent clustering discovered during the network structure recovery. Default is `FALSE`.

`tasks` A factor with n entries indicating the task belonging for each observation in the multiple sample framework. Default is `factor(rep(1, nrow(X)))`, that is, all observations come from a unique homogeneous sample.

`control` A list that is used to specify low-level options for the algorithm, defined through the `setOptions` function.

Details

Any inference method available ("neighborhood selection", "graphical-Lasso", "VAR(1) inference" and "multitask learning" - see `simone-package`) relies on an optimization problem under the general form

$$\hat{\mathbf{\Theta}}_\lambda = \arg\max_{\mathbf{\Theta}} \mathcal{L}(\mathbf{\Theta}; \mathbf{X}) - \lambda \, \mathrm{pen}_{\ell_1}(\mathbf{\Theta}, \mathbf{Z}),$$

where $\mathcal{L}$ is the log-likelihood of the model (pseudo log-likelihood for "neighborhood selection") and λ is a penalty parameter which controls the sparsity level of the network. The $p \times p$ matrix $\mathbf{\Theta}$ describes the parameters (basically, the edges) of the model, while $\mathbf{Z}$ represents a latent clustering which is also estimated when the argument `clustering` is set to `TRUE`.

The model and the penalty function pen_{ℓ_1} differ according to the context (steady-state/time-course data, multitask learning and its associated coupling effect). For further details on the models, please check the papers listed in the reference section of `simone-package`.

The criterion displayed during a SIMoNe run is the value of the penalized likelihood for the current values of the estimor $\hat{\mathbf{\Theta}}_\lambda$ corresponding to a given value of the overall penalty level λ.

The following information criteria are also computed for any value of λ and part of the output of `simone`. The BIC (*Bayesian Information Criterion*)

$$\mathrm{BIC}(\lambda) = \mathcal{L}(\hat{\mathbf{\Theta}}_\lambda; \mathbf{X}) - \mathrm{df}(\hat{\mathbf{\Theta}}_\lambda) \frac{\log(n)}{2},$$

and the AIC (*Akaike Information Criterion*)

$$\mathrm{AIC}(\lambda) = \mathcal{L}(\hat{\mathbf{\Theta}}_\lambda; \mathbf{X}) - \mathrm{df}(\hat{\mathbf{\Theta}}_\lambda).$$

Value

Returns an object of class `simone`, which is list-like and contains the following:

`networks` a list with all the inferred networks stocked as adjacency matrices (the successive values of $\hat{\mathbf{\Theta}}_\lambda$ controled by the penalty level λ). In the multiple sample setup, each element of the list is a list with as many entries as samples or levels in `tasks`.

`penalties` a vector of the same length as `networks`, containing the successive values of the penalty level.

`n.edges` a vector of the same length as `networks`, containing the successive numbers of edges in the inferred networks. In the multiple sample setup, `n.edges` is a matrix with as many columns as levels in `tasks`.

`BIC` a vector of the same length as `networks`, containing the value of the BIC for the successively estimated networks.

`AIC` a vector of the same length as `networks`, containing the value of the AIC for the successively estimated networks.

`clusters` a size-p factor indicating the class of each variable.

`weights` a $p \times p$ matrix of weigths used to adapt the penalty to each entry of the `Theta` matrix. It is inferred through the algorithm according to the latent clustering of the network. When `clustering` is set to `FALSE`, all the weights are equal to "1", which mean no adaptive penalization.

`control` a list describing all the posterior values of the parameters used by the algorithm, to compare with the one set by the `setOptions` function. As a matter of fact, many of the options are defined depending on the nature of the data and can be automatically corrected during internal checks of the coherence of desired options to the characteristics of the data.

Note

If nothing particular is specified about the penalty through the `control` list (see `setOptions`), the default is to start from a value of λ that ensures an empty network. Then λ is progressively shrinked, as close to zero as possible. Along the shrinkage of λ, only networks with different numbers of edges are kept in the final output.

Author(s)

J. Chiquet

See Also

`setOptions`, `plot.simone`, `cancer` and `demo(package="simone")`.

Examples

```
## load the breast cancer data set
data(cancer)
attach(cancer)

## launch simone with the default parameters and plot results
plot(simone(expr))

## Not run:
## try with clustering now (clustering is achieved on a 30-edges network)
plot(simone(expr, clustering=TRUE, control=setOptions(clusters.crit=30)))

## try the multiple sample
plot(simone(expr, tasks=status))

## End(Not run)

detach(cancer)
```

Index