Package 'lpNet'

November 22, 2024

Type Package

Title Linear Programming Model for Network Inference

Version 2.39.0

Date 2020-10199

Author Bettina Knapp, Marta R. A. Matos, Johanna Mazur, Lars Kaderali

Maintainer Lars Kaderali <lars.kaderali@uni-greifswald.de>

Depends lpSolve, KEGGgraph

Description lpNet aims at infering biological networks, in particular signaling and gene networks. For that it takes perturbation data, either steady-state or time-series, as input and generates an LP model which allows the inference of signaling networks. For parameter identification either leave-one-out cross-validation or stratified n-fold cross-validation can be used.

License Artistic License 2.0

biocViews NetworkInference

git_url https://git.bioconductor.org/packages/lpNet

git_branch devel

git_last_commit b90bc0c

git_last_commit_date 2024-10-29

Repository Bioconductor 3.21

Date/Publication 2024-11-21

Contents

`lpNet-package` *Network Inference Of Perturbation Data Using a Linear Programming Approach.*

Description

lpNet aims at infering biological networks, in particular signaling and gene networks. For that it takes perturbation data, either steady-state or time-series, as input and generates an LP model which allows the inference of signaling networks. For parameter identification either leave-one-out cross-validation or stratified n-fold cross-validation can be used.

Details

Package:	lpNet
Type:	Package
Version:	1.99.2
Date:	2013-01-11
License:	Artistic License 2.0

Author(s)

B. Knapp, M. R. A. Matos, J. Mazur, L. Kaderali

Maintainer: bettina.knapp@helmholtz-muenchen.de

References

Bettina Knapp and Lars Kaderali, Reconstruction of cellular signal transduction networks using perturbation assays and linear programming, PLoS ONE, 2013.

Marta R. A. Matos, Network inference : extension of linear programming model for time-series data, Master's thesis, Department of Informatics, University of Minho.

calcActivation *Calculate Activation Matrix*

Description

Calculate the activation matrix assuming that the signaling is deterministically propagated along the network. For a given network and perturbation experiment the theoretical states of the genes are computed. So, if a gene has been silenced in an experiment, then the state of this gene is assumed to be inactive, otherwise if its inflow (coming from parent nodes) is activating, it is active. Cycles within a network are not resolved, therefore this function can be used only for networks without cycles. This function is also used to generate the network states for time-series data (by generateTimeSeriesNetStates), in which case flag_gen_data is set to true, and the activation matrix is calculated without taking the edges sign into account.

Usage

```
calcActivation(T_nw, b, n, K, flag_gen_data = FALSE)
```

Arguments

T_nw	Adjacency matrix: the network which is used to compute the activities and inactivities.
b	Vector of 0/1 values describing the experiments (entry is 0 if gene is inactivated in the respetive experiment and 1 otherwise). The measurements of the genes of each experiment are appended as a long vector.
n	Integer: number of genes.
K	Integer: number of perturbation experiments.
flag_gen_data	Logical: if set to TRUE the edges sign will not be taken into account. It should be TRUE if the function is used to generate the network states for time-series data.

Value

Matrix of 0/1 values; rows corresponding to genes, columns to experiments. If an entry is 1, it means that the corresponding gene is active in the corresponing experiment and inactive otherwise.

Examples

```
n <- 5 # number of genes
K <- 7 # number of perturbations experiments

# perturbation vector, entry is 0 if gene is inactivated and 1 otherwise
b <- c(0,1,1,1,1, # perturbation exp1: gene 1 perturbed, gene 2-5 unperturbed
       1,0,1,1,1, # perturbation exp2: gene 2 perturbed, gene 1,3,4,5 unperturbed
       1,1,0,1,1, # perturbation exp3....
       1,1,1,0,1,
       1,1,1,1,0,
```

```
       1,0,0,1,1,
       1,1,1,1,1)

# example network
T_nw <- matrix(c(0,1,1,0,0,
                 0,0,0,-1,0,
                 0,0,0,1,0,
                 0,0,0,0,1,
                 0,0,0,0,0), nrow=n,ncol=n,byrow=TRUE)

# compute theoretical activation of genes from example network with given perturbations
act_mat <- calcActivation(T_nw, b, n, K)
```

`calcPrediction` *Calculate Predicted Observation.*

Description

Calculate the predicted observation of a perturbation experiment. If observations of an experiment are missing this function can be used to determine for a given network the predicted outcome. The missing measurement is predicted from two normal distributions, one for observations coming from active and one coming from inactive genes. The state of the gene is predicted based on the states of its parents.

Usage

```
calcPredictionLOOCV(obs, delta, b, n ,K, adja, baseline, rem_gene,
                    rem_k, rem_t=NULL, active_mu, active_sd, inactive_mu,
                    inactive_sd, mu_type, flag_time_series=FALSE)
calcPredictionKfoldCV(obs, delta, b, n, K, adja, baseline, rem_entries=NULL,
                      rem_entries_vec=NULL, active_mu, active_sd, inactive_mu,
                      inactive_sd, mu_type, flag_time_series=FALSE)
```

Arguments

`obs`	Numeric matrix/array: the observation matrix/array. It can have up to 3 dimensions, where dimension 1 are the network nodes, dimension 2 are the perturbation experiments, and dimension 3 are the time points (if considered).
`delta`	Numeric vector defining the thresholds for each gene to determine its observation to be active or inactive.
`b`	Binary vector representing the perturbation experiments (entry is 0 if gene is inactive in the respective experiment and 1 otherwise).
`n`	Number of genes in the observation matrix.
`K`	Number of perturbation experiments.
`adja`	Numeric matrix: the adjacency matrix of the given network.
`baseline`	Vector containing the inferred baseline vectors of each gene.

`rem_gene` Integer: the index of the gene that is missing.

`rem_k` Integer: the index of the perturbation experiment that is missing.

`rem_t` Integer: the index of the time point that is missing.

`rem_entries` Numeric matrix: each row represents an entry that was removed from the observation matrix, while the 3 columns represent the gene, perturbation experiment and time point, respectively.

`rem_entries_vec` Numeric vector: contains the entries that were removed in an "absolute form", i.e., if entry (2,1,2) was removed, it will appear in this vector as simply 5.

`active_mu` Numeric: the average value assumed for observations coming from active nodes. The parameter active_mu and active_sd are used for predicting the observations of the normal distribution of activate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.

`active_sd` Numeric: the variation assumed for observations coming from active nodes. The parameter active_mu and active_sd are used for predicting the observations of the normal distribution of activate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.

`inactive_mu` Numeric: the average value assumed for observations coming from inactive nodes. The parameter inactive_mu and inactive_sd are used for predicting the observations of the normal distribution of inactivate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.

`inactive_sd` Numeric: the variation assumed for observations coming from inactive nodes. The parameter inactive_mu and inactive_sd are used for predicting the observations of the normal distribution of inactivate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.

`mu_type` Character: can have the following values and meanings:

- "simple" - the value of active_mu/sd and inactive_mu/sd is independent of the gene/perturbation experiment/time point;
- "perGene" - the value of active_mu/sd and inactive_mu/sd depends on the gene;
- perGeneExp" - the value of active_mu/sd and inactive_mu/sd depends on the gene and perturbation experiment;
- perGeneTime" - the value of active_mu/sd and inactive_mu/sd depends on the gene and time point;
- "perGeneExpTime" - the value of active_mu/sd and inactive_mu/sd depends on the gene, perturbation experiment, and time point;

`flag_time_series` Logical: specifies whether steady-state (FALSE) or time series data (TRUE) is used.

See Also

`loocv`, `kfoldCV`

Examples

```
n <- 3 # number of genes
K <- 4 # number of experiments
T_ <- 4 # number of time points

# perturbation vector, entry is 0 if gene is inactivated and 1 otherwise
b <- c(0,1,1, # perturbation exp1: gene 1 perturbed, gene 2,3 unperturbed
       1,0,1, # perturbation exp2: gene 2 perturbed, gene 1,3 unperturbed
       1,1,0, # perturbation exp3....
       1,1,1)

# adjacency matrix
adja <- matrix(c(0,1,0,
                 0,0,1,
                 0,0,0), nrow=n, ncol=n, byrow=TRUE)

# define node baseline values
baseline <- c(0.75, 0, 0)

# define delta value
delta <- rep(0.75, n)

# define the parameters for the observation generated from the normal distributions
mu_type <- "single"
active_mu <- 0.9
inactive_mu <- 0.5
active_sd <- inactive_sd <- 0.01

#### kfoldCV

# generate random observation matrix
obs <- array(rnorm(n*K*T_), c(n,K,T_))

# define the observationd to be removed, whose values will be predicted
obs[2,4,2] <- NA
obs[3,4,3] <- NA

rem_entries <- which(is.na(obs), arr.ind=TRUE)
rem_entries_vec <- which(is.na(obs))

# compute the predicted observation matrix for the "kfoldCV"
calcPredictionKfoldCV(obs=obs, delta=delta, b=b, n=n, K=K, adja=adja, baseline=baseline,
                      rem_entries=rem_entries, rem_entries_vec=rem_entries_vec,
                      active_mu=active_mu, active_sd=active_sd, inactive_mu=inactive_mu,
                      inactive_sd=inactive_sd, mu_type=mu_type, flag_time_series=TRUE)


#### LOOCV
# generate random observation matrix
obs <- matrix(rnorm(n*K), nrow=n, ncol=K)

# define the observationd to be removed
```

```
rem_k <- 3
rem_gene <- 2
obs[rem_gene, rem_k] <- NA

# compute the predicted value
calcPredictionLOOCV(obs=obs, delta=delta,b=b, n=n ,K=K, adja=adja, baseline=baseline,
                rem_gene=rem_gene, rem_k=rem_k, active_mu=active_mu, active_sd=active_sd,
                     inactive_mu=inactive_mu, inactive_sd=inactive_sd, mu_type=mu_type)
```

`calcRangeLambda` *Compute Range Of Penalty Parameter Lambda.*

Description

The penalty parameter lambda can range from zero to infinity and it controls the introduction of slack variables in the network inference lp model. To limit the introduction of slack variables we restrict lambda to be not larger than lambdaMax (=the number of slack variables times the variance of all measurements given). This function computes the range from zero to lambdaMax with a given stepsize that increases exponentially.

Usage

```
calcRangeLambda(obs, delta, delta_type, flag_time_series=FALSE)
```

Arguments

`obs`	Numeric matrix/array: the observation matrix/array. It can have up to 3 dimensions, where dimension 1 are the network nodes, dimension 2 are the perturbation experiments, and dimension 3 are the time points (if considered).
`delta`	Numeric: defines the thresholds for each gene to determine its observation to be active or inactive. This parameter can be either a numeric vector, a matrix, or a 3D array, depending on the specified delta_type.
`delta_type`	Character: can have the following values and meanings: • "perGene" - the value of delta depends on the gene; • "perGeneExp" - the value of delta depends on the gene and perturbation experiment; • "perGeneTime" - the value of delta depends on the gene and time point; • "perGeneExpTime" - the value of delta depends on the gene, perturbation experiment, and time point;
`flag_time_series`	Logical: specifies whether steady-state (FALSE) or time series data (TRUE) is used.

Value

Numeric vector of possible values for lambda.

Examples

```
# generate random observation matrix with 5 experiments and 5 genes
obs <- matrix(rnorm(5*5, 1, 0.1), nrow=5, ncol=5)

# define delta to be 1 for each gene
delta <- rep(1, 5)
delta_type <- "perGene"

lambda_values <- calcRangeLambda(obs, delta, delta_type)
```

CV *Cross-validation*

Description

Performs a stratified k-fold cross-validation or a Leave-One-Out cross-validation.

Usage

```
loocv(kfold=NULL, times, obs, delta, lambda, b, n, K, T_=NULL,
      annot, annot_node, active_mu, active_sd, inactive_mu,
      inactive_sd, mu_type, delta_type, prior=NULL, sourceNode=NULL,
      sinkNode=NULL, allint=FALSE, allpos=FALSE, flag_time_series=FALSE)
kfoldCV(kfold, times, obs, delta, lambda, b, n, K, T_=NULL,
        annot, annot_node, active_mu, active_sd, inactive_mu,
        inactive_sd, mu_type, delta_type, prior=NULL,
    sourceNode=NULL, sinkNode=NULL, allint=FALSE,
        allpos=FALSE, flag_time_series=FALSE)
```

Arguments

kfold	Integer value of the number "k" in the k-fold cross-calidation.
times	Integer: the number of times the cross-validation shall be performed.
obs	Numeric matrix/array: the measured observation matrix/array. It can have up to 3 dimensions, where dimension 1 are the network nodes, dimension 2 are the perturbation experiments, and dimension 3 are the time points (if considered).
delta	Numeric vector, matrix, or array defining the thresholds to determine an observation active or inactive.
lambda	Numeric value defining the penalty parameter lambda. It can range from zero to infinity and it controls the introduction of slack variables in the network inference lp model.
n	Integer: number of genes.
b	Vector of 0/1 values describing the experiments (entry is 0 if gene is inactivated in the respetive experiment and 1 otherwise). The measurements of the genes of each experiment are appended as a long vector.

K — Integer: number of perturbation experiments.

T_ — Integer: number of time points in time-series data.

annot — Vector of character strings: the annotation of the edges as returned by "getEdgeAnnot".

annot_node — Vector of character strings: the annoation of the nodes.

active_mu — Numeric: the average value assumed for observations coming from activated nodes. The parameter active_mu and active_sd are used for predicting the observations of the normal distribution of activate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.

active_sd — Numeric: the variation assumed for observations coming from activated nodes. The parameter active_mu and active_sd are used for predicting the observations of the normal distribution of activate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.

inactive_mu — Numeric: the average value assumed for observations coming from inactivated nodes. The parameter inactive_mu and inactive_sd are used for predicting the observations of the normal distribution of inactivate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.

inactive_sd — Numeric: the variation assumed for observations coming from inactivated nodes. The parameter inactive_mu and inactive_sd are used for predicting the observations of the normal distribution of inactivate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.

mu_type — Character: can have the following values and meanings:

- "simple" - the value of active_mu/sd and inactive_mu/sd is independent of the gene/perturbation experiment/time point;
- "perGene" - the value of active_mu/sd and inactive_mu/sd depends on the gene;
- perGeneExp" - the value of active_mu/sd and inactive_mu/sd depends on the gene and perturbation experiment;
- perGeneTime" - the value of active_mu/sd and inactive_mu/sd depends on the gene and time point;
- "perGeneExpTime" - the value of active_mu/sd and inactive_mu/sd depends on the gene, perturbation experiment, and time point;

delta_type — Character: can have the following values and meanings:

- "perGene" - the value of delta depends on the gene;
- "perGeneExp" - the value of delta depends on the gene and perturbation experiment;
- "perGeneTime" - the value of delta depends on the gene and time point;
- "perGeneExpTime" - the value of delta depends on the gene, perturbation experiment, and time point;

prior — Prior knowledge, given as a list of constraints. Each constraint consists of a vector with four entries describing the prior knowledge of one edge. For example the edge between node 1 and 2, called w+_1_2, is defined to be bigger than 1

with constraint c("w+_1_2",1,">",2). The first entry specifies the annotation of the edge (see function "getEdgeAnnot") and the second defines the coefficient of the objective function (see parameter "objective.in" in the "lp" function of the package "lpSolve"). Furthermore, the third, respectively the fourth elements give the direction, respectively the right-hand side of the constraint (see the parameters "const.dir", respectively "const.rhs" in the "lp" function of the package "lpSolve").

`sourceNode` Integer vector: indices of the known source nodes.

`sinkNode` Integer vector: indices of the known sink nodes.

`allint` Logical: should all variables be integer? Corresponds to an Integer Linear Program (see "lp" function in package "lpSolve"). Default: FALSE.

`allpos` Logical: should all variables be positive? Corresponds to learning only activating edges. Default: FALSE.

`flag_time_series` Logical: specifies whether steady-state (FALSE) or time series data (TRUE) is used.

Value

A list of

`MSE` The mean squared error (MSE) of predicted and observed measurements of the corresponding cross-validation step.

`edges_all` The learned edge weights for each cross-validation step.

`baseline_all` The learned baseline weights for each cross-validation step.

Examples

```
n <- 3 # number of genes
K <- 4 # number of experiments
T_ <- 4 # number of time points

annot_node <- seq(1, n)
annot <- getEdgeAnnot(n)

# generate random observation matrix
obs <- array(rnorm(n*K*T_), c(n,K,T_))

baseline <- c(0.75, 0, 0)

# define delta
delta <- apply(obs, 1, mean, na.rm=TRUE)

# perturbation vector, entry is 0 if gene is inactivated and 1 otherwise
b <- c(0,1,1, # perturbation exp1: gene 1 perturbed, gene 2,3 unperturbed
       1,0,1, # perturbation exp2: gene 2 perturbed, gene 1,3 unperturbed
       1,1,0, # perturbation exp3....
       1,1,1)
```

```
T_nw <- matrix(c(0,1,0,
                 0,0,1,
                 0,0,0), nrow=n, ncol=n, byrow=TRUE)
colnames(T_nw) <- rownames(T_nw) <- annot_node

## calculate observation matrix with given parameters for the
# Gaussian distributions for activation and deactivation
active_mu <- 0.95
inactive_mu <- 0.56
active_sd <- inactive_sd <- 0.1

times <- kfold <- 10   # can be increased i.e. to 1000 to produce stable results

mu_type <- "single"
delta_type <- "perGene"

lambda <- 1/10


#### LOOCV
loocv(kfold=NULL, times=times, obs=obs, delta=delta, lambda=lambda, b=b, n=n, K=K, T_=T_, annot=annot,
    annot_node=annot_node, active_mu=active_mu, active_sd=active_sd, inactive_mu=inactive_mu,
    inactive_sd=inactive_sd, mu_type=mu_type, delta_type=delta_type, prior=NULL, sourceNode=NULL,
      sinkNode=NULL, allint=FALSE, allpos=FALSE, flag_time_series=TRUE)

#### K-fold CV
kfoldCV(kfold=kfold, times=times, obs=obs, delta=delta, lambda=lambda, b=b, n=n, K=K, T_=T_, annot=annot,
     annot_node=annot_node, active_mu=active_mu, active_sd=active_sd, inactive_mu=inactive_mu,
     inactive_sd=inactive_sd, mu_type=mu_type, delta_type=delta_type, prior=NULL, sourceNode=NULL,
        sinkNode=NULL, allint=FALSE, allpos=FALSE, flag_time_series=TRUE)
```

---

doILP *Do The Network Inference With The Linear Programming Approach.*

Description

This function converts observation data into a linear programming problem.

Usage

```
doILP(obs, delta, lambda, b, n, K, T_=NULL, annot, delta_type,
      prior=NULL,sourceNode=NULL, sinkNode=NULL, all.int=FALSE,
      all.pos=FALSE, flag_time_series=FALSE)
```

Arguments

obs	Numeric matrix/array: the given observation matrix/array. It can have up to 3 dimensions, where dimension 1 are the network nodes, dimension 2 are the perturbation experiments, and dimension 3 are the time points (if considered).

delta	Numeric: defining the thresholds for each gene to determine its observation to be active or inactive. This parameter can be either a numeric vector, a matrix, or a 3D array, depending on the specified delta_type.
lambda	Numeric value defining the penalty parameter lambda. It can range from zero to infinity and it controls the introduction of slack variables in the network inference lp model.
b	Vector of 0/1 values describing the experiments (entry is 0 if gene is inactivated in the respetive experiment and 1 otherwise). The measurements of the genes of each experiment are appended as a long vector.
n	Integer: number of genes.
K	Integer: number of perturbation experiments.
T_	Integer: number of time points.
annot	Vector of character strings: the annotation of the edges as returned by "getEdgeAnnot".
delta_type	Character: can have the following values and meanings: - "perGene" - the value of delta depends on the gene; - "perGeneExp" - the value of delta depends on the gene and perturbation experiment; - "perGeneTime" - the value of delta depends on the gene and time point; - "perGeneExpTime" - the value of delta depends on the gene, perturbation experiment, and time point;
prior	Prior knowledge, given as a list of constraints. Each constraint consists of a vector with four entries describing the prior knowledge of one edge. For example the edge between node 1 and 2, called w+_1_2, is defined to be bigger than 1 with constraint c("w+_1_2",1,">",2). The first entry specifies the annotation of the edge (see function "getEdgeAnnot") and the second defines the coefficient of the objective function (see parameter "objective.in" in the "lp" function of the package "lpSolve"). Furthermore, the third, respectively the fourth elements give the direction, respectively the right-hand side of the constraint (see the parameters "const.dir", respectively "const.rhs" in the "lp" function of the package "lpSolve").
sourceNode	Integer vector: indices of the known source nodes.
sinkNode	Integer vector: indices of the known sink nodes.
all.int	Logical: should all variables be integer? Corresponds to an Integer Linear Program (see "lp" function in package "lpSolve"). Default: FALSE.
all.pos	Logical: should all variables be positive? Corresponds to learning only activating edges. Default: FALSE.
flag_time_series	Logical: specifies whether steady-state (FALSE) or time series data (TRUE) is used.

Value

An lp object. See "lp.object" in package "lpSolve" for details.

Examples

```
n <- 3 # number of genes
K <- 4 # number of experiments
T_ <- 4 # number of time points

# generate random observation matrix
obs <- array(rnorm(n*K*T_), c(n,K,T_))

baseline <- c(0.75, 0, 0)

delta <- rep(0.75, n)

# perturbation vector, entry is 0 if gene is inactivated and 1 otherwise
b <- c(0,1,1, # perturbation exp1: gene 1 perturbed, gene 2,3 unperturbed
       1,0,1, # perturbation exp2: gene 2 perturbed, gene 1,3 unperturbed
       1,1,0, # perturbation exp3....
       1,1,1)

delta_type <- "perGene"
lambda <- 1/10
annot <- getEdgeAnnot(n)

res <- doILP(obs, delta, lambda, b, n, K, T_, annot, delta_type, prior=NULL,
         sourceNode=NULL, sinkNode=NULL, all.int=FALSE, all.pos=FALSE, flag_time_series=TRUE)
```

`generateTimeSeriesNetStates`

Generate Time Series Network States

Description

The function returns all gene states for each network state in time-series data. The signalling propagates downstream one edge per time-point. The stopping criteria is when all edges have been active at least once, so that infinite loops are avoided. The number of time points for the data can be defined by the user or not, if not the number of time points will be the same as the number of different network states. If the number of time points is defined by the user, network states will be either repeated or removed, so that there are as many network states as time points.

Usage

```
generateTimeSeriesNetStates(nw_und, b, n, K, T_user=NULL)
```

Arguments

`nw_und`	Numeric matrix: the adjacency matrix representing the underlying network.
`b`	Vector of 0/1 values describing the experiments (entry is 0 if gene is inactivated in the respetive experiment and 1 otherwise). The measurements of the genes of each experiment are appended as a long vector.

n	Integer: number of genes.
K	Integer: number of perturbation experiments.
T_user	Integer definining the number of time points in the network.

Value

List containing an array with all nodes states and the number of time points.

Examples

```
n <- 3 # number of genes
K <- 4 # number of experiments

# perturbation vector, entry is 0 if gene is inactivated and 1 otherwise
b <- c(0,1,1, # perturbation exp1: gene 1 perturbed, gene 2,3 unperturbed
       1,0,1, # perturbation exp2: gene 2 perturbed, gene 1,3 unperturbed
       1,1,0, # perturbation exp3....
       1,1,1)

# adjacency matrix
nw_und <- matrix(c(0,1,0,
                   0,0,1,
                   0,0,0), nrow=n, ncol=n, byrow=TRUE)

generateTimeSeriesNetStates(nw_und,b, n, K, T_user=5)
```

---

getAdja *Get Adjacency Matrix.*

Description

The function returns the adjacency matrix of the network computed with the "doILP" function.

Usage

```
getAdja(res, n, annot=NULL)
```

Arguments

res	Result returned by the "doILP" function.
n	Integer: the number of nodes of the inferred network.
annot	Vector of character strings: the annotation of the edges as returned by "getEdgeAnnot".

Value

Numeric matrix: the adjacency matrix of the network.

See Also

doILP

Examples

```
n <- 3 # number of genes
K <- 4 # number of experiments
T_ <- 4 # number of time points

# generate random observation matrix
obs <- array(rnorm(n*K*T_), c(n,K,T_))

baseline <- c(0.75, 0, 0)

delta <- rep(0.75, n)

# perturbation vector, entry is 0 if gene is inactivated and 1 otherwise
b <- c(0,1,1, # perturbation exp1: gene 1 perturbed, gene 2,3 unperturbed
       1,0,1, # perturbation exp2: gene 2 perturbed, gene 1,3 unperturbed
       1,1,0, # perturbation exp3....
       1,1,1)

delta_type <- "perGene"
lambda <- 1/10
annot <- getEdgeAnnot(n)

#infer the network
res <- doILP(obs, delta, lambda, b, n, K, T_, annot, delta_type, prior=NULL,
         sourceNode=NULL, sinkNode=NULL, all.int=FALSE, all.pos=FALSE, flag_time_series=TRUE)

# make the adjacency matrix
adja <- getAdja(res, n)
```

---

getBaseline *Get Baseline Vector.*

Description

The function returns a vector with the baseline values of each node in the network computed with the "doILP" function.

Usage

```
getBaseline(res, n, allpos=FALSE)
```

Arguments

res	Result returned by the "doILP" or "doILP_timeSeries" function.
n	Integer: the number of nodes of the inferred network.
allpos	Logical: should all variables be positive? Corresponds to learning only activating edges. Default: FALSE.

Value

Numeric matrix: the adjacency matrix of the network.

See Also

doILP

Examples

```
n <- 3 # number of genes
K <- 4 # number of experiments
T_ <- 4 # number of time points

# generate random observation matrix
obs <- array(rnorm(n*K*T_), c(n,K,T_))

baseline <- c(0.75, 0, 0)

delta <- rep(0.75, n)

# perturbation vector, entry is 0 if gene is inactivated and 1 otherwise
b <- c(0,1,1, # perturbation exp1: gene 1 perturbed, gene 2,3 unperturbed
       1,0,1, # perturbation exp2: gene 2 perturbed, gene 1,3 unperturbed
       1,1,0, # perturbation exp3....
       1,1,1)

delta_type <- "perGene"
lambda <- 1/10
annot <- getEdgeAnnot(n)

#infer the network
res <- doILP(obs, delta, lambda, b, n, K, T_, annot, delta_type, prior=NULL, sourceNode=NULL,
             sinkNode=NULL, all.int=FALSE, all.pos=FALSE, flag_time_series=TRUE)

# make the adjacency matrix
adja <- getBaseline(res, n)
```

getEdgeAnnot *Get the annotation of the edges.*

Description

The function returns the annotation of the edges needed for the LP. Positive edges are annotated with "w+" and negative with "w-". The given nodes are just enumerated from 1 to n and the edge between node i and j is given by "w+_i_j" for the positive, respectively by "w-_i_j" for the negative edges. The annotation "w_i_^_0" defines the baseline activity of gene i.

Usage

```
getEdgeAnnot(n, allpos)
```

Arguments

n	Integer: number of genes.
allpos	Logical: should all edges be positive? Corresponds to learning only activating edges. Default: FALSE.

Examples

```
n <- 5
annot <- getEdgeAnnot(n)
```

getObsMat *Get Observation Matrix.*

Description

The function generates the observation matrix where active/inactive observations are generated from a normal distribution with the average and variation as given in the parameters. This matrix can either be generated from the activation matrix calculated with calcActivation or from the network states caculated with generateTimeSeriesNetStates.

Usage

```
getObsMat(act_mat=NULL, net_states=NULL, active_mu, active_sd,
          inactive_mu, inactive_sd, mu_type)
```

Arguments

act_mat	Matrix of 0/1 values called the activation matrix. Rows correspond to genes, columns to experiments. If an entry is 1, it means that the corresponding gene is active in the corresponing experiment and inactive otherwise.
net_states	Array of 0/1 values called the network states. Rows correspond to genes, columns to experiments, and the third dimension corresponds to time points. If an entry is 1, it means that the corresponding gene is active in the corresponing experiment and inactive otherwise.
active_mu	Numeric: the average value assumed for observations coming from activated nodes. The parameter active_mu and active_sd are used for predicting the observations of the normal distribution of activate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.
active_sd	Numeric: the variation assumed for observations coming from activated nodes. The parameter active_mu and active_sd are used for predicting the observations of the normal distribution of activate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.
inactive_mu	Numeric: the average value assumed for observations coming from inactivated nodes. The parameter inactive_mu and inactive_sd are used for predicting the observations of the normal distribution of inactivate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.
inactive_sd	Numeric: the variation assumed for observations coming from inactivated nodes. The parameter inactive_mu and inactive_sd are used for predicting the observations of the normal distribution of inactivate genes. This parameter can be either a numeric, a vector, a matrix, or a 3D array, depending on the specified mu_type.
mu_type	Character: can have the following values and meanings: • "simple" - the value of active_mu/sd and inactive_mu/sd is independent of the gene/perturbation experiment/time point; • "perGene" - the value of active_mu/sd and inactive_mu/sd depends on the gene; • perGeneExp" - the value of active_mu/sd and inactive_mu/sd depends on the gene and perturbation experiment; • perGeneTime" - the value of active_mu/sd and inactive_mu/sd depends on the gene and time point; • "perGeneExpTime" - the value of active_mu/sd and inactive_mu/sd depends on the gene, perturbation experiment, and time point;

Value

Numeric matrix/array: the observation matrix/array. It can have up to 3 dimensions, where dimension 1 are the network nodes, dimension 2 are the perturbation experiments, and dimension 3 are the time points (if considered).

See Also

calcActivation

Examples

```
n <- 5 # number of genes
K <- 7 # number of knockdowns

# perturbation vector, entry is 0 if gene is inactivated and 1 otherwise
b <- c(0,1,1,1,1, # perturbation exp1: gene 1 perturbed, gene 2-5 unperturbed
       1,0,1,1,1, # perturbation exp2: gene 2 perturbed, gene 1,3,4,5 unperturbed
       1,1,0,1,1, # perturbation exp3....
       1,1,1,0,1,
       1,1,1,1,0,
       1,0,0,1,1,
       1,1,1,1,1)

T_nw <- matrix(c(0,1,1,0,0,
                 0,0,0,-1,0,
                 0,0,0,1,0,
                 0,0,0,0,1,
                 0,0,0,0,0), nrow=n, ncol=n, byrow=TRUE)

act_mat <- calcActivation(T_nw, b, n, K)

# define the parameters for the observation generated from the normal distribution
active_mu <- 0.9
inactive_mu <- 0.5
active_sd <- inactive_sd <- 0.1
mu_type <- "single"

# compute the observations matrix
getObsMat(act_mat=act_mat, active_mu=active_mu, active_sd=active_sd, inactive_mu=inactive_mu, inactive_sd=inacti
```

`getSampleAdja` *Get The Sample Adjacency.*

Description

The function computes the adjacency of the edges computed in each step of the "loocv" or the "kfoldCV" function. If the variance of each edge shall be taken into account use "getSampleAdjaMAD", otherwise "getSampleAdja".

Usage

```
getSampleAdjaMAD(edges_all, n, annot_node, method = median,
                 method2 = mad, septype = "->")
getSampleAdja(edges_all, n, annot_node, method = median, septype = "->")
```

Arguments

`edges_all`	The inferred edges using the "loocv" or the "kfoldCV" function.
`n`	Integer: the number of nodes.

annot_node	Vector of character strings: the annoation of the nodes.
method	Character string: the method used to summarize the edges of the individual steps. Default: "median".
method2	Character string: the method used for the computation of the variation of the edges of the individual steps. Default: "mad".
septype	Character string: the type of separation of two nodes in the annot string vector. Default: "->".

Value

Numeric matrix: the adjacency matrix.

See Also

loocv, kfoldCV

Examples

```
# compute random edge weights
edges_all <- matrix(rnorm(5*6), nrow=5, ncol=6)

# annotation of the edges as returned by "loocv" and kfoldCV
colnames(edges_all) <- c("1->2", "1->3", "2->1", "2->3", "3->1", "3->2")

# annotation of the nodes
annot_node <- c(1,2,3)
getSampleAdjaMAD(edges_all, n=3, annot_node, method = "median", method2 = "mad", septype = "->")

getSampleAdja(edges_all, n=3, annot_node, method = "median", septype = "->")
```

---

summarizeRepl *Summarize Replicate Measurements*

Description

The function returns the the summarized replicate measuremenst.

Usage

```
summarizeRepl(data,type=median)
```

Arguments

data	The data matrix.
type	The summarization type which shall be used. Default: median.

Value

Numeric matrix: the summarized data.

Examples

```
data("SahinRNAi2008")
## process data
dataStim <- dat.normalized[dat.normalized[ ,17] == 1, -17]

# summarize replicates
dataSt <- t(summarizeRepl(dataStim, type=mean))
```

Index