Package ‘mgm’

July 7, 2022

Type Package

Title Estimating Time-Varying k-Order Mixed Graphical Models

Version 1.2-13

Date 2022-07-07

Author Jonas Haslbeck

Maintainer Jonas Haslbeck <jonashaslbeck@gmail.com>

Depends R (>= 3.5.0)

Description Estimation of k-Order time-varying Mixed Graphical Models and mixed VAR(p) models via elastic-net regularized neighborhood regression. For details see Haslbeck & Waldorp (2020) <doi:10.18637/jss.v093.i08>.

URL https://www.jstatsoft.org/article/view/v093i08

BugReports https://github.com/jmbh/mgm/issues

Imports glmnet, stringr, Hmisc, qgraph, gtools

LazyData true

Suggests testthat (>= 2.0.0)

Config/testthat/edition 2

NeedsCompilation no

Repository CRAN

Date/Publication 2022-07-07 12:00:05 UTC

R topics documented:

`mgm-package` *Estimating Time-Varying k-order Mixed Graphical Models*

Description

Estimation of time-varying Mixed Graphical models and mixed VAR models via elastic-net regularized neighborhood regression.

Details

Package:	mgm
Type:	Package
Version:	1.2-10
Date:	2020-06-27
License:	GPL-2

Author(s)

Jonas Haslbeck

Maintainer: <jonashaslbeck@gmail.com>

References

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Loh, P. L., & Wainwright, M. J. (2013). Structure estimation for discrete graphical models: Generalized covariance matrices and their inverses. The Annals of Statistics, 41(6), 3022-3049.

Yang, E., Baker, Y., Ravikumar, P., Allen, G., & Liu, Z. (2014). Mixed graphical models via exponential families. In Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (pp. 1042-1050).

`bwSelect` *Select optimal bandwidth for time-varying MGMs and mVAR Models*

Description

Selects the bandwidth parameter with lowest out of sample prediction error for MGMs and mVAR Models.

Usage

```
bwSelect(data, type, level, bwSeq, bwFolds,
         bwFoldsize, modeltype, pbar, ...)
```

Arguments

`data`	A n x p data matrix.
`type`	p vector indicating the type of variable for each column in `data`. "g" for Gaussian, "p" for Poisson, "c" for categorical.
`level`	p vector indicating the number of categories of each variable. For continuous variables set to 1.
`bwSeq`	A sequence with candidate bandwidth values (0, s] with s < Inf. Note that the bandwidth is applied relative to the unit time interval [0,1] and hence a banwidth of > 2 corresponds roughly to equal weights for all time points and hence gives similar estimates as the stationary model estimated via `mvar()`.
`bwFolds`	The number of folds (see details below).
`bwFoldsize`	The size of each fold (see details below).
`modeltype`	If `modeltype = "mvar"` model, the optimal bandwidth parameter for a `tvmvar()` model is selected. If `modeltype = "mgm"` model, the optimal bandwidth parameter for a `tvmgm()` model is selected. Additional arguments to `tvmvar()` or `tvmgm()` can be passed via the `...` argument.
`pbar`	If TRUE a progress bar is shown. Defaults to `pbar = "TRUE"`.
`...`	Arguments passed to `tvmgm` or `tvmvar`.

Details

Performs a cross-validation scheme that is specified by `bwFolds` and `bwFoldsize`. In the first fold, the test set is defined by an equally spaced sequence between [1, `n - bwFolds`] of length `bwFoldsize`. In the second fold, the test set is defined by an equally spaced sequence between [2, `n - bwFolds + 1`] of length `bwFoldsize`, etc. . Note that if `bwFoldsize = n / bwFolds`, this procedure is equal to `bwFolds`-fold cross valildation. However, full cross validation is computationally very expensive and a single split in test/training set by setting `bwFolds = 1` is sufficient in many situations. The procedure selects the bandwidth with the lowest prediction error, averaged over variables and time points in the test set.

bwSelect computes the absolute error (continuous) or 0/1-loss (categorical) for each time point in the test set defined by bwFoldsize as described in the previous paragraph for every fold specified in bwFolds, separately for each variable. The computed errors are returned in different levels of aggregation in the output list (see below). Note that continuous variables are scaled (centered and divided by their standard deviation), hence the absolute error and 0/1-loss are roughly on the scale scale.

Note that selecting the bandwidth with the EBIC is no alternative. This is because the EBIC always selects the intercept model with the lowest bandwidth. The reason is that the unregularized intercept closely models the noise in the data and hence the penalty sets all other parameters to zero. This problem is solved by using out of sample prediction error in the cross validation scheme.

Value

The function returns a list with the following entries:

- `call` Contains all provided input arguments. If `saveData = TRUE`, it also contains the data.
- `bwModels` Contains the models estimated at the time points in the tests set. For details see `tvmvar` or `tvmgm`.
- `fullErrorFolds` List with number of entries equal to the length of `bwSeq` entries. Each entry contains a list with `bwFolds` entries. Each of those entries contains a contains a `bwFoldsize` times p matrix of out of sample prediction errors.
- `fullError` The same as `fullErrorFolds` but pooled over folds.
- `meanError` List with number of entries equal to the length of `bwSeq` entries. Each entry contains the average prediction error over variables and time points in the test set.
- `testsets` List with `bwFolds` entries, which contain the rows of the test sample for each fold.
- `zeroweights` List with `bwFolds` entries, which contains the observation weights used to fit the model at the `bwFoldsize` time points.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Barber, R. F., & Drton, M. (2015). High-dimensional Ising model selection with Bayesian information criteria. Electronic Journal of Statistics, 9(1), 567-607.

Foygel, R., & Drton, M. (2010). Extended Bayesian information criteria for Gaussian graphical models. In Advances in neural information processing systems (pp. 604-612).

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Examples

```
## Not run:


## A) bwSelect for tvmgm()

# A.1) Generate noise data set
p <- 5
n <- 100
data_n <- matrix(rnorm(p*n), nrow=100)
head(data_n)

type <- c("c", "c", rep("g", 3))
level <- c(2, 2, 1, 1, 1)
x1 <- data_n[,1]
x2 <- data_n[,2]
data_n[x1>0,1] <- 1
data_n[x1<0,1] <- 0
data_n[x2>0,2] <- 1
data_n[x2<0,2] <- 0

head(data_n)

# A.2) Estimate optimal bandwidth parameter

bwobj_mgm <- bwSelect(data = data_n,
                      type = type,
                      level = level,
                      bwSeq = seq(0.05, 1, length=3),
                      bwFolds = 1,
                      bwFoldsize = 3,
                      modeltype = "mgm",
                      k = 3,
                      pbar = TRUE,
                      overparameterize = TRUE)


print.mgm(bwobj_mgm)



## B) bwSelect for tvmVar()

# B.1) Generate noise data set

p <- 5
n <- 100
data_n <- matrix(rnorm(p*n), nrow=100)
head(data_n)

type <- c("c", "c", rep("g", 3))
```

```
level <- c(2, 2, 1, 1, 1)
x1 <- data_n[,1]
x2 <- data_n[,2]
data_n[x1>0,1] <- 1
data_n[x1<0,1] <- 0
data_n[x2>0,2] <- 1
data_n[x2<0,2] <- 0

head(data_n)

# B.2) Estimate optimal bandwidth parameter

bwobj_mvar <- bwSelect(data = data_n,
                       type = type,
                       level = level,
                       bwSeq = seq(0.05, 1, length=3),
                       bwFolds = 1,
                       bwFoldsize = 3,
                       modeltype = "mvar",
                       lags = 1:3,
                       pbar = TRUE,
                       overparameterize = TRUE)


print.mgm(bwobj_mvar)

# For more examples see https://github.com/jmbh/mgmDocumentation



## End(Not run)
```

`condition` *Computes mgm object conditional on a set of variables*

Description

The function takes an mgm object and a set of variables fixed to given values as input and returns the conditional mgm object.

Usage

```
condition(object, values)
```

Arguments

`object`	An mgm object, the output of the `mgm()` function.
`values`	A list, where the entry name indicates the column number of the variable that should be fixed, and the entry value indicates the value to which the corresponding variable should be fixed. See below for an example.

Details

The new conditional object still contains the variables that were fixed, however, they are not related to any of the random variables anymore. We kept the variables in the object to avoid confusion with variable labels and plotting. Also note that `mgm()` by default scales all Gaussian variables to mean=0, sd=1. Thus, fixed values should be selected based on the scaled version of variables.

Value

The function returns an mgm object that is conditional on the provided values. The new mgm object can again be used as input in `predict()`, `print()`, `showInteraction()`, etc..

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Haslbeck, J., & Waldorp, L. J. (2019). mgm: Estimating time-varying mixed graphical models in high-dimensional data. arXiv preprint arXiv:1510.06871.

See Also

`mgm`

Examples

```
## Not run:

# --- Create Mixture of two Gaussians ---

set.seed(1)
n <- 500
library(MASS)

# Component A
Sigma_a <- diag(2)
Sigma_a[1, 2] <- Sigma_a[2, 1] <- .5
Xa <- mvrnorm(n = n, mu = rep(0, 2), Sigma = Sigma_a)

# Component B
Sigma_b <- diag(2)
Sigma_b[1, 2] <- Sigma_b[2, 1] <- 0
Xb <- mvrnorm(n = n, mu = rep(0, 2), Sigma = Sigma_b)

data <- as.data.frame(cbind(rbind(Xa, Xb), c(rep(0, n), rep(1, n))))
colnames(data) <- c("x1", "x2", "x3")


# --- Fit MGM ---
```

```
# with mgm
mgm_obj <- mgm(data = data,
               type = c("g","g","c"),
               level = c(1, 1, 2),
               moderator = c(3),
               lambdaSel = "EBIC")

# --- Condition on / fix values of variable 3 ---

# Fix x3=0
mgm_obj_x3.0 <- condition(object = mgm_obj,
                          values = list("3"=0))
mgm_obj_x3.0$pairwise$wadj

# Fix x3=1
mgm_obj_x3.1 <- condition(object = mgm_obj,
                          values = list("3"=1))
mgm_obj_x3.1$pairwise$wadj


## End(Not run)
```

---

datasets *Example Datasets in the mgm Package*

Description

The autism dataset (and its short version) are taken from Deserno et al. (2016).

The restingstate fMRI data are taken from Schmittmann et al. (2015).

The gene expression data across the life span of the fruit fly are taken from Gibberd & Nelson (2017), who took a subset of the data first presented by Arbeitman et al. (2002).

The symptom data of the single individual diagnosed with major depression is described in Kossakowski et al. (2017).

The PTSD data is taken from McNally et al. (2015).

The dataset `mgm_data` is generated by example code shown in ?mgmsampler, and `mvar_data` is generated by example code shown in ?mvarsampler.

The dataset `Fried2015` contains 515 cases of the 11 depression symptoms measured by the CES-D and is taken from Fried et al. 2015.

The dataset `B5MS` contains the mean scores across subscales (48 items each) for the Big Five personality traits. The dataset is taken from the `qgraph` package (Epskamp, et al., 2012) and was first used in Dolan et al. (2009).

The dataset `dataGD` contains 4 continuous variables and 3 categorical variables that are generated from a mixed DAG. This dataset is useful to illustrate estimating group differences in MGMs using moderation.

All datasets are loaded automatically. All real data sets come as a list including the data and additional information (names of variables, types of variables, time stamps for time series data, etc.)

References

Deserno, M. K., Borsboom, D., Begeer, S., & Geurts, H. M. (2016). Multicausal systems ask for multicausal approaches: A network perspective on subjective well-being in individuals with autism spectrum disorder. Autism.

Dolan, C. V., Oort, F. J., Stoel, R. D., & Wicherts, J. M. (2009). Testing measurement invariance in the target rotated multigroup exploratory factor model. Structural Equation Modeling, 16(2), 295-314.

Epskamp, S., Cramer, A. O., Waldorp, L. J., Schmittmann, V. D., & Borsboom, D. (2012). qgraph: Network visualizations of relationships in psychometric data. Journal of Statistical Software, 48(4), 1-18.

Schmittmann, V. D., Jahfari, S., Borsboom, D., Savi, A. O., & Waldorp, L. J. (2015). Making large-scale networks from fMRI data. PloS one, 10(9), e0129074.

Gibberd, A. J., & Nelson, J. D. (2017). Regularized Estimation of Piecewise Constant Gaussian Graphical Models: The Group-Fused Graphical Lasso. Journal of Computational and Graphical Statistics, (just-accepted).

Arbeitman, M. N., Furlong, E. E., Imam, F., Johnson, E., Null, B. H., Baker, B. S., ... & White, K. P. (2002). Gene expression during the life cycle of Drosophila melanogaster. Science, 297(5590), 2270-2275.

Kossakowski, J., Groot, P., Haslbeck, J., Borsboom, D., & Whichers, M. (2017). Data from "Critical Slowing Down as a Personalized Early Warning Signal for Depression". Journal of Open Psychology Data, 5(1).

McNally, R. J., Robinaugh, D. J., Wu, G. W., Wang, L., Deserno, M. K., & Borsboom, D. (2015). Mental disorders as causal systems a network approach to posttraumatic stress disorder. Clinical Psychological Science, 3(6), 836-849.

Fried, E. I., Bockting, C., Arjadi, R., Borsboom, D., Amshoff, M., Cramer, A. O., ... & Stroebe, M. (2015). From loss to loneliness: The relationship between bereavement and depressive symptoms. Journal of abnormal psychology, 124(2), 256.

`FactorGraph` *Draws a factor graph of a (time-varying) MGM*

Description

Wrapper function around qgraph() that draws factor graphs for (time-varying) MGMs

Usage

```
FactorGraph(object, labels, PairwiseAsEdge = FALSE,
            Nodewise = FALSE, DoNotPlot = FALSE,
            FactorLabels = TRUE, colors, shapes,
            shapeSizes = c(8, 4), estpoint = NULL,
            negDashed = FALSE, ...)
```

Arguments

`object`	The output object of `mgm()` or `tvmgm()`.
`labels`	A character vector of (variable) node labels.
`PairwiseAsEdge`	If `TRUE`, pairwise interactions are not displayed as factors but as simple edges between nodes. Defaults to `PairwiseAsEdge = FALSE`.
`Nodewise`	If `TRUE`, the estimates from the individual nodewise regressions are displayed as a directed edge towards the node on which the respective nodewise regression was performed. This is useful to identify model misspecification (e.g. moderation effects / interaction parameters with largely different values across nodewise regressions). Defaults to `Nodewise = FALSE`.
`DoNotPlot`	If `DoNotPlot = TRUE` no factorgraph is plotted. This way the computed factor graph can be obtained without plotting. Defaults to `DoNotPlot = FALSE`.
`FactorLabels`	If `FactorLabels = TRUE` the factors are labeled by their order. If `FactorLabels = FALSE` no label is shown. Defaults to `FactorLabels = TRUE`.
`colors`	A character vector of colors for nodes and factors. The first color is for variable-nodes, the second for 2-way interactions, the third for 3-way interactions, etc. Defaults to `colors = c("white", "tomato", "lightblue", "orange")`.
`shapes`	A character vector of shapes for for nodes and factors. The first shape is for variable-nodes, the second for 2-way interactions, the third for 3-way interactions, etc. Defaults to `shapes = c("circle", "square", "triangle", "diamond")`.
`shapeSizes`	A numeric vector of length two indicating the size of shapes for nodes and factors. Defaults to `shapeSizes = c(8, 4)`.
`estpoint`	An integer indicating the estimation point to display if the output object of a time-varying MGM is provided.
`negDashed`	If `negDashed = TRUE`, edges with negative sign are dashed.
`...`	Arguments passed to qgraph.

Details

`FactorGraph()` is a wrapper around `qgraph()` from the qgraph package. Therefore all arguments of `qgraph()` are available and can be provided as additional arguments.

To make time-varying factor graphs comparable across estimation points, the factor graph of each estimation point includes all factors that are estimated nonzero at least at one estimation point.

Value

Plots the factor graph and returns a list including the arguments used to plot the factor graph using qgraph().

Specifically, a list is returned including: `graph` contains a weighted adjacency matrix of a (bipartide) factor graph. If p is the number of variables and E the number of interactions (factors) in the model, this matrix has dimensions (p+E) x (p+E). The factor graph is furter specified by the following objects: `signs` is a matrix of the same dimensions as `graph` that indicates the sign of each interaction, if defined (see `pairwise` above). `edgecolor` is a matrix with the same dimension as `graph` that provides edge colors depending on the sign as above. `order` is a (p+E) vector indicating the order of interaction. The first p entries are set to zero. `qgraph` contains the qgraph object created while plotting.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

See Also

mgm(), tvmgm(), qgraph()

Examples

```
## Not run:

# Fit MGM with pairwise & threeway interactions to Autism Dataset
fit_k3 <- mgm(data = autism_data$data,
              type = autism_data$type,
              level = autism_data$lev,
              k = 3,
              overparameterize = TRUE,
              lambdaSel = "EBIC",
              lambdaGam = .5)

# List of estimated interactions
fit_k3$interactions$indicator

FactorGraph(object = fit_k3,
            PairwiseAsEdge = FALSE,
            DoNotPlot = FALSE,
            labels = 1:7,
            layout="circle")

# For more examples see https://github.com/jmbh/mgmDocumentation

## End(Not run)
```

---

mgm *Estimating Mixed Graphical Models*

Description

Function to estimate k-degree Mixed Graphical Models via nodewise regression.

Usage

```
mgm(data, type, level, lambdaSeq, lambdaSel, lambdaFolds,
    lambdaGam, alphaSeq, alphaSel, alphaFolds, alphaGam,
    k, moderators, ruleReg, weights, threshold, method,
    binarySign, scale, verbatim, pbar, warnings, saveModels,
    saveData, overparameterize, thresholdCat, signInfo, ...)
```

Arguments

- `data` — n x p data matrix
- `type` — p vector indicating the type of variable for each column in `data`. "g" for Gaussian, "p" for Poisson, "c" for categorical.
- `level` — p vector indicating the number of categories of each variable. For continuous variables set to 1.
- `lambdaSeq` — A sequence of lambdas that should be searched (see also `lambdaSel`). Defaults to `NULL`, which uses the `glmnet` default to select a lambda candidate sequence (recommended). See `?glmnet` for details.
- `lambdaSel` — Specifies the procedure for selecting the tuning parameter controlling the Lq-penalization. The two options are cross validation "CV" and the Extended Bayesian Information Criterion (EBIC) "EBIC". The EBIC performs well in selecting sparse graphs (see Barber and Drton, 2010 and Foygel and Drton, 2014). Note that when also searching the alpha parameter in the elastic net penalty, cross validation should be preferred, as the parameter vector will not necessarily be sparse anymore. The EBIC tends to be a bit more conservative than CV (see Haslbeck and Waldorp, 2016). CV can sometimes not be performed with categorical variables, because `glmnet` requires at least 2 events of each category of each categorical variable in each training-fold. Defaults to `lambdaSel = "CV"`.
- `lambdaFolds` — Number of folds in cross validation if `lambdaSel = "CV"`.
- `lambdaGam` — Hyperparameter gamma in the EBIC if `lambdaSel = "EBIC"`. Defaults to `lambdaGam = .25`.
- `alphaSeq` — A sequence of alpha parameters for the elastic net penality in [0,1] that should be searched (see also `alphaSel`). Defaults to `alphaSeq = 1`, which means that the lasso is being used. `alphaSeq = 0` corresponds to an L2-penalty (Ridge regression). For details see Friedman, Hastie and Tibshirani (2010).
- `alphaSel` — Specifies the procedure for selecting the alpha parameter in the elastic net penalty. The two options are cross validation "CV" and the Extended Bayesian Information Criterion (EBIC) "EBIC". The EBIC performs well in selecting sparse graphs (see Barber and Drton, 2010 and Foygel and Drton, 2014). Note that when also searching the alpha parameter in the elastic net penalty, cross validation should be preferred, as the parameter vector will not necessarily be sparse anymore. The EBIC tends to be a bit more conservative than CV (see Haslbeck and Waldorp, 2016). CV can sometimes not be performed with categorical variables, because `glmnet` requires at least 2 events of each category of each categorical variable in each training-fold. Defaults to `alphaSel = "CV"`.
- `alphaFolds` — Number of folds in cross validation if `alphaSel = "CV"`.
- `alphaGam` — Hyperparameter gamma in the EBIC if `alphaSel = "EBIC"`. Defaults to `alphaGam = .25`.
- `k` — Order up until including which interactions are included in the model. `k = 2` means that all pairwise interactions are included, `k = 3` means that all pairwise and all three-way interactions are included, etc. In previous versions of mgm the order of interactions was specified by the parameter d, the largest size or a neighborhood. Note that k = d + 1.

moderators
: Integer vector with elements in `1:p`, specifying moderation effects to be included in the model. For instance, `moderators = c(4)` includes all linear moderation effects of variable 4. This is equivalent to including all 3-way interactions that include variable 4. Note that `moderators = 1:p` gives the same model as setting `k = 3` (see previous argument). Alternatively, a specific set of moderators can be specified via a M x 3 matrix, where M is the number of moderation effects. For example, `moderators = matrix(1:3, nrow=1)` adds (only) the 3-way interaction 1-2-3 to the model.

ruleReg
: Rule used to combine estimates from neighborhood regression. E.g. for pairwise interactions, two estimates (one from regressing A on B and one from B on A) have to combined in one edge parameter. `ruleReg = "AND"` requires all estimates to be nonzero in order to set the edge to be present. `ruleReg = "OR"` requires at least one estiamte to be nonzero in order to set the edge to be present. For higher order interactions, `k` estimate have to be combined with this rule.

weights
: A n vector with weights for observations.

threshold
: A threshold below which edge-weights are put to zero. This is done in order to guarantee a lower bound on the false-positive rate. `threshold = "LW"` refers to the threshold in Loh and Wainwright (2013), which was used in all previous versions of mgm. `threshold = "HW"` refers to the threshold in Haslbeck and Waldorp (2016). If `threshold = "none"` no thresholding is applied. Defaults to `threshold = "LW"`.

method
: Estimation method, currently only `method = "glm"`.

binarySign
: If `binarySign = TRUE`, a sign for the interaction within binary nodes and between binary and continuous nodes is provided in the output. Note that in this case the two categories of the binary variables have to be coded in 0,1. This is to ensure that the interpretation of the sign is unambigous: a positive sign of a parameter means that increasing the associated predictor results in a higher probability for category 1.

scale
: If `scale = TRUE`, all Gaussian nodes (specified by `"g"` in `type`) are centered and divided by their standard deviation. Scaling is recommended, because otherwise the penalization of a parameter depends on the variance of the associated predictor. Defaults to `scale = TRUE`.

verbatim
: If `verbatim = TRUE`, no warnings and no progress bar is shown. Defaults to `verbatim = FALSE`.

pbar
: If `pbar = TRUE`, a progress bar is shown. Defaults to `pbar = TRUE`.

warnings
: If `warnings = TRUE`, no warnigns are returned. Defaults to `warnings = FALSE`.

saveModels
: If `saveModels = FALSE`, only information about the weighted adjacency matrix, and if k > 2 about the factor graph is provided in the output list. If `saveModels = TRUE`, all fitted parameters are additionally returned.

saveData
: If `saveData = TRUE`, the data is saved in the output list. Defaults to `saveData = FALSE`.

overparameterize
: If `overparameterize = TRUE`, `mgm()` uses over-parameterized design-matrices for each neighborhood regression; this means that an interaction between two categorical variables with m and s categories is parameterized by m*s parameters. If `overparameterize = FALSE` the standard parameterization (in `glmnet`)

with m*(s-1) parameters is used, where the first category of the predicting variable serves as reference category. If all variables are continuous both parameterizations are the same. Note that the default is set to `overparameterize = FALSE`, to be consistent with the previous mgm versions. However when the goal is to seperate pairwise interactions from 3-way (or higher) interactions, then the overparameterized version is advantageous. See the examples below for an illustration. Note that we can estimate the model despite the colinear columns in the design matrix because we use penalized regression.

`thresholdCat` If `thresholdCat = FALSE`, the thresholds of categorical variables are set to zero. Defaults to `thresholdCat = TRUE` for which the thresholds are esimated.

`signInfo` If `signInfo = TRUE`, a message is shown in the console, indicating that the sign of estimates is stored separately. Defaults to `signInfo = TRUE`.

`...` Additional arguments.

Details

`mgm()` estimates an exponential mixed graphical model as introduced in Yang and colleagies (2014). Note that MGMs are not normalizable for all parameter values. See Chen, Witten & Shojaie (2015) for an overview of when pairwise MGMs are normalizable. To our best knowledge, for MGMs with interactions of order > 2 that include non-categorical variables, the conditions for normalizablity are unknown.

Value

The function returns a list with the following entries:

`call` Contains all provided input arguments. If `saveData = TRUE`, it also contains the data

`pairwise` Contains a list with all information about estimated pairwise interactions. `wadj` contains the p x p weighted adjacency matrix, if p is the number of variables in the network. `signs` has the same dimensions as `wadj` and contains the signs for the entries of `wadj`: 1 indicates a positive sign, -1 a negative sign and 0 an undefined sign. A sign is undefined if an edge is a function of more than one parameter. This is the case for interactions involving a categorical variable with more than 2 categories. `edgecolor` also has the same dimensions as `wadj` contains a color for each edge, depending on `signs`. It is provided for more convenient plotting. If only pairwise interactions are modeled (`d = 1`), `wadj` contains all conditional independence relations. The matrices `edgecolor_cb` contain a color blind friendly color scheme. `edge_lty` contains a matrix with 1s for positive/undefined signs and 2s for negative signes, which can be used as input to the `lty` argument in `qgraph()` in order to plot edges with negative sign as dashed lines.

`interactions` A list with three entries that relate each interaction in the model to all its parameters. This is different to the output provided in `factorgraph`, where one value is assigned to each interaction. `indicator` contains a list with k-1 entries, one for each order of modeled interaction, which contain the estimated (nonzero) interactions. `weightsAgg` contains a list with k-1 entries, which in turn contain R lists, where R is the number of interactions (and rows in the corresponding list entry `inindicator`) that were estimated (nonzero) in the given entry. Each of these entries contains the mean of the absolute values of all parameters involved in this interaction. `weights` has the same structure as `weightsAgg`, but does contain all parameters involved in the interaction instead of the mean of their absolute values. `signs` has the same structure as `weightsAgg`/`weights` and provides the sign of the interaction, if defined.

`intercepts` A list with p entries, which contain the intercept/thresholds for each node in the network. In case a given node is categorical with m categories, there are m thresholds for this variable.

`nodemodels` A list with p `glmnet()` models, from which all above output is computed. Also contains the coefficients `models` for the selected lambda and the applied tau threshold `tau`.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Barber, R. F., & Drton, M. (2015). High-dimensional Ising model selection with Bayesian information criteria. Electronic Journal of Statistics, 9(1), 567-607.

Chen S, Witten DM & Shojaie (2015). Selection and estimation for mixed graphical models. Biometrika, 102(1), 47.

Foygel, R., & Drton, M. (2010). Extended Bayesian information criteria for Gaussian graphical models. In Advances in neural information processing systems (pp. 604-612).

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33(1), 1.

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Loh, P. L., & Wainwright, M. J. (2012, December). Structure estimation for discrete graphical models: Generalized covariance matrices and their inverses. In NIPS (pp. 2096-2104).

Yang, E., Baker, Y., Ravikumar, P., Allen, G. I., & Liu, Z. (2014, April). Mixed Graphical Models via Exponential Families. In AISTATS (Vol. 2012, pp. 1042-1050).

Examples

```
## Not run:


## We fit a pairwise and 3-order MGM to the mixed Autism dataset (?autism_data)

# 1) Fit Pairwise MGM

# Call mgm()
fit_k2 <- mgm(data = autism_data$data,
              type = autism_data$type,
```

```
              level = autism_data$lev,
              k = 2) # ad most pairwise interacitons

# Weighted adjacency matrix
fit_k2$pairwise$wadj

# Visualize using qgraph()
library(qgraph)
qgraph(fit_k2$pairwise$wadj,
       edge.color = fit_k2$pairwise$edgecolor,
       layout = "spring",
       labels =  autism_data$colnames)


# 2) Fit MGM with pairwise & three-way interactions
fit_k3 <- mgm(data = autism_data$data,
              type = autism_data$type,
              level = autism_data$lev,
              k = 3) # include all interactions up to including order 3

# List of estimated interactions
fit_k3$interactions$indicator

# 3) Plot Factor Graph
FactorGraph(object = fit_k3,
            PairwiseAsEdge = FALSE,
            labels = autism_data$colnames)

# 4) Predict values
pred_obj <- predict(fit_k3, autism_data$data)

head(pred_obj$predicted) # first six rows of predicted values
pred_obj$errors # Nodewise errors


## Here we illustrate why we need to overparameterize the design matrix to
## recover higher order interactions including categorical variables

# 1) Define Graph (one 3-way interaction between 3 binary variables)

# a) General Graph Info
type = c("c", "c", "c")
level = c(2, 2, 2)
# b) Define Interaction
factors <- list()
factors[[1]] <- NULL # no pairwise interactions
factors[[2]] <- matrix(c(1,2,3), ncol=3, byrow = T) # one 3-way interaction
interactions <- list()
interactions[[1]] <- NULL
interactions[[2]] <- vector("list", length = 1)
# threeway interaction no1
interactions[[2]][[1]] <- array(0, dim = c(level[1], level[2], level[3]))
theta <- .7
```

```
interactions[[2]][[1]][1, 1, 1] <- theta  #weight theta for conf (1,1,1), weight 0 for all others
# c) Define Thresholds
thresholds <- list()
thresholds[[1]] <- c(0, 0)
thresholds[[2]] <- c(0, 0)
thresholds[[3]] <- c(0, 0)


# 2) Sample from Graph
iter <- 1
set.seed(iter)
N <- 2000
d_iter <- mgmsampler(factors = factors,
                     interactions = interactions,
                     thresholds = thresholds,
                     type = type,
                     level = level,
                     N = N,
                     nIter = 50,
                     pbar = TRUE)


# 3.1) Estimate order 3 MGM using standard parameterization
d_est_stand <- mgm(data = d_iter$data,
                   type = type,
                   level = level,
                   k = 3,
                   lambdaSel = "CV",
                   ruleReg = "AND",
                   pbar = TRUE,
                   overparameterize = FALSE,
                   signInfo = FALSE)

# We look at the nodewise regression for node 1 (same for all)
coefs_stand <- d_est_stand$nodemodels[[1]]$model
coefs_stand
# We see that nonzero-zero pattern of parameter vector does not allow us to infer whether
# interactions are present or not


# 3.2) Estimate order 3 MGM using overparameterization
d_est_over <- mgm(data = d_iter$data,
                  type = type,
                  level = level,
                  k = 3,
                  lambdaSel = "CV",
                  ruleReg = "AND",
                  pbar = TRUE,
                  overparameterize = TRUE,
                  signInfo = FALSE)

# We look at the nodewise regression for node 1 (same for all)
coefs_over <- d_est_over$nodemodels[[1]]$model
```

```
coefs_over # recovers exactly the 3-way interaction

# For more examples see https://github.com/jmbh/mgmDocumentation



## End(Not run)
```

---

mgmsampler *Sample from k-order Mixed Graphical Model*

Description

Generates samples from a k-order Mixed Graphical Model

Usage

```
mgmsampler(factors, interactions, thresholds, sds, type,
           level, N, nIter = 250, pbar = TRUE,
           divWarning = 10^3, returnChains = FALSE)
```

Arguments

`factors` This object indicates which interactions are present in the model. It is a list in which the first entry corresponds to 2-way interactions, the second entry corresponds to 3-way interactions, etc. and the kth entry to the k+1-way interaction. Each entry contains a matrix with dimensions order x number of interaction of given order. Each row in the matrix indicates an interaction, e.g. (1, 3, 7, 9) in the matrix in list entry three indicates a 4-way interaction between the variables 1, 3, 7 and 9.

`interactions` This object specifies the parameters associated to the interactions specified in `factors`. Corresponding to the structure in `factors`, this object is a list, where the kth entry corresponds to k+1-way interactions. Each list entry contains another list, with entries equal to the number of rows in the corresponding matrix in `factors`. Each of these list entries (for a fixed k) contains a k-dimensional array that specifies the parameters of the given k-order interaction. For instance, if we have a 3-way interaction (1, 2, 3) and all variables are binary, we have a 2 x 2 x 2 array specifiying the parameters for each of the 2^3 = 8 possible configurations. If all variables are continuous, we have a 1 x 1 x 1 array, so the interaction is specified by a single parameter. See the examples below for an illustration.

thresholds	A list with p entries corresponding to p variables in the model. Each entry contains a vector indicating the threshold for each category (for categorical variables) or a numeric value indicating the threshold/intercept (for continuous variables).
sds	A numeric vector with p entries, specifying the variances of Gaussian variables. If variables 6 and 13 are Gaussians, then the corresponding entries of `sds` have to contain the corresponding variances. Other entries are ignored.
type	p character vector indicating the type of variable for each column in `data`. "g" for Gaussian, "p" for Poisson, "c" of each variable.
level	p integer vector indicating the number of categories of each variable. For continuous variables set to 1.
N	Number of samples that should be drawn from the distribution.
nIter	Number of iterations in the Gibbs sampler until a sample is drawn.
pbar	If `pbar = TRUE` a progress bar is shown. Defaults to `pbar = TRUE`.
divWarning	`mgmsampler()` returns a warning message if the absolute value of a continuous variable the chain of the gibbs sampler is larger than `divWarning`. To our best knowledge there is no theory yet defining a parameter space that ensures a proper probability density and hence a converging chain. Defaults to `divWarning = 10^3`.
returnChains	If `returnChains = TRUE`, the sampler provides the entire chain of the Gibbs sampler, for each sampled case. Can be used to check convergence of the Gibbs sampler. Defaults to `returnChains = FALSE`.

Details

We use a Gibbs sampler to sample from the join distribution introduced by Yang and colleageus (2014). Note that the contraints on the parameter space necessary to ensure that the joint distribution is normalizable are to our best knowledge unknown. Yang and colleagues (2014) give these constraints for a number of simple pairwise models. In practice, an "improper joint density" will lead to a sampling process that approaches infinity, and hence `mgmsampler()` will return `Inf / -Inf` values.

Value

A list containing:

call	Contains all provided input arguments.
data	The N x p data matrix of sampled values

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Haslbeck, J., & Waldorp, L. J. (2018). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. arXiv preprint arXiv:1510.06871.

Yang, E., Baker, Y., Ravikumar, P., Allen, G. I., & Liu, Z. (2014, April). Mixed Graphical Models via Exponential Families. In AISTATS (Vol. 2012, pp. 1042-1050).

Examples

```
## Not run:

# --------- Example 1: p = 10 dimensional Gaussian ---------

# ----- 1) Specify Model -----

# a) General Graph Info
p <- 10 # number of variables
type = rep("g", p) # type of variables
level = rep(1, 10) # number of categories for each variable (1 = convention for continuous)

# b) Define interactions
factors <- list()
factors[[1]] <- matrix(c(1,2,
                         1,3,
                         4,5,
                         7,8), ncol=2, byrow = T) # 4 pairwise interactions
interactions <- list()
interactions[[1]] <- vector("list", length = 4)

# all pairwise interactions have value .5
for(i in 1:4) interactions[[1]][[i]] <- array(.5, dim=c(1, 1))

# c) Define Thresholds
thresholds <- vector("list", length = p)
thresholds <- lapply(thresholds, function(x) 0 ) # all means are zero

# d) Define Variances
sds <- rep(1, p) # All variances equal to 1


# ----- 2) Sample cases -----

data <- mgmsampler(factors = factors,
                   interactions = interactions,
                   thresholds = thresholds,
                   sds = sds,
                   type = type,
                   level = level,
                   N = 500,
                   nIter = 100,
                   pbar = TRUE)
```

```
# ----- 3) Recover model from sampled cases -----

set.seed(1)
mgm_obj <- mgm(data = data$data,
               type = type,
               level = level,
               k = 2,
               lambdaSel = "EBIC",
               lambdaGam = 0.25)

mgm_obj$interactions$indicator # worked!



# --------- Example 2: p = 3 Binary model with one 3-way interaction ---------

# ----- 1) Specify Model -----

# a) General Graph Info
type = c("c", "c", "c")
level = c(2, 2, 2)

# b) Define Interaction
factors <- list()
factors[[1]] <- NULL # no pairwise interactions
factors[[2]] <- matrix(c(1,2,3), ncol=3, byrow = T) # one 3-way interaction

interactions <- list()
interactions[[1]] <- NULL
interactions[[2]] <- vector("list", length = 1)
# threeway interaction no1
interactions[[2]][[1]] <- array(0, dim = c(level[1], level[2], level[3]))
theta <- 2
interactions[[2]][[1]][1, 1, 1] <- theta # fill in nonzero entries
# thus: high probability for the case that x1 = x2 = x3 = 1

# c) Define Thresholds
thresholds <- list()
thresholds[[1]] <- rep(0, level[1])
thresholds[[2]] <- rep(0, level[2])
thresholds[[3]] <- rep(0, level[3])


# ----- 2) Sample cases -----

set.seed(1)
dlist <- mgmsampler(factors = factors,
                    interactions = interactions,
                    thresholds = thresholds,
                    type = type,
                    level = level,
```

```
                    N = 500,
                    nIter = 100,
                    pbar = TRUE)


# ----- 3) Check: Contingency Table -----

dat <- dlist$data
table(dat[,1], dat[,2], dat[,3]) # this is what we expected


# ----- 4) Recover model from sampled cases -----

mgm_obj <- mgm(data = dlist$data,
               type = type,
               level = level,
               k = 3,
               lambdaSel = "EBIC",
               lambdaGam = 0.25,
               overparameterize = TRUE)

mgm_obj$interactions$indicator # recovered, plus small spurious pairwise 1-2


# --------- Example 3: p = 5 Mixed Graphical Model with two 3-way interaction ---------

# ----- 1) Specify Model -----

# a) General Graph Info
type = c("g", "c", "c", "g")
level = c(1, 3, 5, 1)
# b) Define Interaction
factors <- list()
factors[[1]] <- NULL # no pairwise interactions
factors[[2]] <- matrix(c(1,2,3,
                         2,3,4), ncol=3, byrow = T) # no pairwise interactions
interactions <- list()
interactions[[1]] <- NULL
interactions[[2]] <- vector("list", length = 2)
# 3-way interaction no1
interactions[[2]][[1]] <- array(0, dim = c(level[1], level[2], level[3]))
interactions[[2]][[1]][,,1:3] <- rep(.8, 3) # fill in nonzero entries
# 3-way interaction no2
interactions[[2]][[2]] <- array(0, dim = c(level[2], level[3], level[4]))
interactions[[2]][[2]][1,1,] <- .3
interactions[[2]][[2]][2,2,] <- .3
interactions[[2]][[2]][3,3,] <- .3
# c) Define Thresholds
thresholds <- list()
thresholds[[1]] <- 0
thresholds[[2]] <- rep(0, level[2])
thresholds[[3]] <- rep(0, level[3])
thresholds[[4]] <- 0
```

```
# d) Define Variances
sds <- rep(.1, length(type))


# ----- 2) Sample cases -----

set.seed(1)
data <- mgmsampler(factors = factors,
                   interactions = interactions,
                   thresholds = thresholds,
                   sds = sds,
                   type = type,
                   level = level,
                   N = 500,
                   nIter = 100,
                   pbar = TRUE)


# ----- 3) Check: Conditional Means -----

# We condition on the categorical variables and check whether
# the conditional means match what we expect from the model:

dat <- data$data

# Check interaction 1
mean(dat[dat[,2] == 1 & dat[,3] == 1, 1]) # (compare with interactions[[2]][[1]])
mean(dat[dat[,2] == 1 & dat[,3] == 5, 1])
# first mean higher, ok!

# Check interaction 2
mean(dat[dat[,2] == 1 & dat[,3] == 1, 4]) # (compare with interactions[[2]][[2]])
mean(dat[dat[,2] == 1 & dat[,3] == 2, 4])
# first mean higher, ok!



## End(Not run)
```

mvar *Estimating mixed Vector Autoregressive Model (mVAR)*

Description

Estimates mixed Vector Autoregressive Model (mVAR) via elastic-net regularized Generalized Linear Models

Usage

```
mvar(data, type, level, lambdaSeq, lambdaSel, lambdaFolds,
     lambdaGam, alphaSeq, alphaSel, alphaFolds, alphaGam, lags,
     consec, beepvar, dayvar, weights, threshold, method, binarySign,
     scale, verbatim, pbar, warnings, saveModels, saveData,
     overparameterize, thresholdCat, signInfo, ...)
```

Arguments

`data`	n x p data matrix.
`type`	p vector indicating the type of variable for each column in `data`. "g" for Gaussian, "p" for Poisson, "c" for categorical.
`level`	p vector indicating the number of categories of each variable. For continuous variables set to 1.
`lambdaSeq`	A sequence of lambdas that should be searched (see also `lambdaSel`). Defaults to `NULL`, which uses the `glmnet` default to select a lambda candidate sequence (recommended). See `?glmnet` for details.
`lambdaSel`	Specifies the procedure for selecting the tuning parameter controlling the Lq-penalization. The two options are cross validation "CV" and the Extended Bayesian Information Criterion (EBIC) "EBIC". The EBIC performs well in selecting sparse graphs (see Barber and Drton, 2010 and Foygel and Drton, 2014). Note that when also searching the alpha parameter in the elastic net penalty, cross validation should be preferred, as the parameter vector will not necessarily be sparse anymore. The EBIC tends to be a bit more conservative than CV (see Haslbeck and Waldorp, 2016). CV can sometimes not be performed with categorical variables, because `glmnet` requires at least 2 events of each category of each categorical variable in each training-fold. Defaults to `lambdaSel = "CV"`.
`lambdaFolds`	Number of folds in cross validation if `lambdaSel = "CV"`.
`lambdaGam`	Hyperparameter gamma in the EBIC if `lambdaSel = "EBIC"`. Defaults to `lambdaGam = .25`.
`alphaSeq`	A sequence of alpha parameters for the elastic net penality in [0,1] that should be searched (see also `alphaSel`). Defaults to `alphaSeq = 1`, which means that the lasso is being used. `alphaSeq = 0` corresponds to an L2-penalty (Ridge regression). For details see Friedman, Hastie and Tibshirani (2010).
`alphaSel`	Specifies the procedure for selecting the alpha parameter in the elastic net penalty. The two options are cross validation "CV" and the Extended Bayesian Information Criterion (EBIC) "EBIC". The EBIC performs well in selecting sparse graphs (see Barber and Drton, 2010 and Foygel and Drton, 2014). Note that when also searching the alpha parameter in the elastic net penalty, cross validation should be preferred, as the parameter vector will not necessarily be sparse anymore. The EBIC tends to be a bit more conservative than CV (see Haslbeck and Waldorp, 2016). CV can sometimes not be performed with categorical variables, because `glmnet` requires at least 2 events of each category of each categorical variable in each training-fold. Defaults to `alphaSel = "CV"`.
`alphaFolds`	Number of folds in cross validation if `alphaSel = "CV"`

alphaGam
: Hyperparameter gamma in the EBIC if `alphaSel = "EBIC"`. Defaults to `alphaGam = .25`.

lags
: Vector of positive integers indicating the lags included in the mVAR model (e.g. 1:3 or c(1,3,5))

consec
: An integer vector of length n, indicating the consecutiveness of measurement points of the rows in `data`. This means that rows for which the necessary (defined by the specified VAR model) measurements at previous time points are not available are excluded from the analysis. For instance, for a VAR model with lag 1 a consec vector of `consec = c(1,2,3,5)` would mean that the fourth row is excluded from the analysis, since no measurement 5-1=4 is available (next to the first row, for which also no previous measurement can be available). This is useful in many applications in which measurements are missing randomly or due to the design of the data collection (for example, respondents only respond during the hours they are awake). The "trimmed" dataset is returned in `call$data_lagged` if `saveData = TRUE`. Defaults to `consec = NULL`, which assumes that all measurements are consecutive, i.e. `consec = 1:n`. In this case only the first `max(lags)` lags are excluded to obtain the VAR design matrix.

beepvar
: Together with the argument `dayvar`, this argument is an alternative to the `consec` argument (see above) to specify the consecutiveness of measurements. This is tailored to experience sampling method (ESM) studies, where the consectutiveness is defined by the number of notification on a given day (`beepvar`) and the given day (`dayvar`).

dayvar
: See `beepvar`.

weights
: A vector with n - max(lags) entries, indicating the weight for each observation. The mVAR design matrix has with n - max(lags) rows, because the first row must be predictable by the highest lag. The weights have to be on the scale [0, n - max(lags)].

threshold
: A threshold below which edge-weights are put to zero. This is done in order to guarantee a lower bound on the false-positive rate. `threshold = "LW"` refers to the threshold in Loh and Wainwright (2013), which was used in all previous versions of mgm. `threshold = "HW"` refers to the threshold in Haslbeck and Waldorp (2016). If `threshold = "none"` no thresholding is applied. Defaults to `threshold = "LW"`.

method
: Estimation method, currently only `method = "glm"`.

binarySign
: If `binarySign = TRUE`, a sign for the interaction within binary nodes and between binary and continuous nodes is provided in the output. Note that in this case the two categories of the binary variables have to be coded in 0,1. This is to ensure that the interpretation of the sign is unambigous: a positive sign of a parameter means that increasing the associated predictor results in a higher probability for category 1.

scale
: If `scale = TRUE`, all Gaussian nodes (specified by `"g"` in the `type` argument) are centered and divided by their standard deviation. Scaling is recommended, because otherwise the penalization of a parameter depends on the variance of the associated predictor.

verbatim
: If `verbatim = TRUE`, no warnings and no progress bar is shown. Defaults to `verbatim = FALSE`.

`pbar`	If `pbar = TRUE`, a progress bar is shown. Defaults to `pbar = TRUE`.
`warnings`	If `warnings = TRUE`, no warnigns are returned. Defaults to `warnings = FALSE`.
`saveModels`	If `saveModels = FALSE`, only information about the weighted adjacency matrix, and if d > 1 about the factor graph is provided in the output list. If `saveModels = TRUE`, all fitted parameters are additionally returned.
`saveData`	If `saveData = TRUE`, the data is saved in the output list. Defaults to `saveData = FALSE`.
`overparameterize`	If `overparameterize = TRUE`, `mgm()` uses over-parameterized design-matrices for each neighborhood regression; this means that a cross-lagged effect between two categorical variables with m and s categories is parameterized by m*s parameters. If `overparameterize = FALSE` the standard parameterization (in `glmnet`) with m*(s-1) parameters is used, where the first category of the predicting variable serves as reference category. If all variables are continuous both parameterizations are the same. The default is set to `overparameterize = FALSE`.
`thresholdCat`	If `thresholdCat = FALSE`, the thresholds of categorical variables are set to zero. Defaults to `thresholdCat = TRUE` for which the thresholds are esimated.
`signInfo`	If `signInfo = TRUE`, a message is shown in the console, indicating that the sign of estimates is stored separately. Defaults to `signInfo = TRUE`.
`...`	Additional arguments.

Details

See Haslbeck and Waldorp (2018) for details about how the mixed VAR model is estimated.

Value

The function returns a list with the following entries:

`call`	Contains all provided input arguments. If `saveData = TRUE`, it also contains the data.
`wadj`	A p x p x n_lags array, in which rows are predicted by columns, i.e. entry `wadj[1, 2, 4]` corresponds to the parameter(s) of variable 2 at time point t predicting variable 1 at time point t - z, where z is the fourth specified lag in `lags` and n_lags is the number of specified lags in `lags`. For interactions that involve more than two parameters (e.g. always for categorical variables with more than 2 categories), we take the arithmetic mean of the absolute value of all parameters. The full set of estimated parameters is saved in `rawlags` (see below).
`signs`	A p x p x n_lags array, specifying the signs corresponding to the entries of `wadj` (if defined), where n_lags is the number of specified lags in `lags`. 1/-1 indicate positive and negative relationships, respectively. 0 indicates that no sign is defined, which is the case for interactions that involve a categorical variable where an interaction can have more than one parameter. If `binarySign = TRUE`, a sign is calculated for interactions between binary variables and binary and continuous variables, where the interaction is still defined by one parameter and hence a sign can be specified. `NA` indicates that the corresponding parameter in `wadj` is zero.

edgecolor — A p x p x n_lags array of colors indicating the sign of each parameter. This array contains the same information is `signs` and is included for convenient plotting.

rawlags — List with entries equal to the number of specified lags in `lags`. Each entry is a nested list, with each p entries: the first level indicates the predicted variable, the second level the predictor variable. In case of categorical variables, interactions have more than one parameter.

intercepts — A list with p entries, which contain the intercept/thresholds for each node. In case a given node is categorical with m categories, there are m thresholds for this variable.

nodemodels — A list with p `glmnet()` models, from which all above output is computed. Also contains the coefficients `models` for the selected lambda and the applied tau threshold `tau`.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Barber, R. F., & Drton, M. (2015). High-dimensional Ising model selection with Bayesian information criteria. Electronic Journal of Statistics, 9(1), 567-607.

Foygel, R., & Drton, M. (2010). Extended Bayesian information criteria for Gaussian graphical models. In Advances in neural information processing systems (pp. 604-612).

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33(1), 1.

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Loh, P. L., & Wainwright, M. J. (2012, December). Structure estimation for discrete graphical models: Generalized covariance matrices and their inverses. In NIPS (pp. 2096-2104).

Yang, E., Baker, Y., Ravikumar, P., Allen, G. I., & Liu, Z. (2014, April). Mixed Graphical Models via Exponential Families. In AISTATS (Vol. 2012, pp. 1042-1050).

Examples

```
## Not run:


## We generate data from a mixed VAR model and then recover the model using mvar()

# 1) Define mVAR model
p <- 6 # Six variables
type <- c("c", "c", "c", "c", "g", "g") # 4 categorical, 2 gaussians
level <- c(2, 2, 4, 4, 1, 1) # 2 categoricals with m=2, 2 categoricals with m=4, two continuous
max_level <- max(level)

lags <- c(1, 3, 9) # include lagged effects of order 1, 3, 9
```

```
n_lags <- length(lags)

# Specify thresholds
thresholds <- list()
thresholds[[1]] <- rep(0, level[1])
thresholds[[2]] <- rep(0, level[2])
thresholds[[3]] <- rep(0, level[3])
thresholds[[4]] <- rep(0, level[4])
thresholds[[5]] <- rep(0, level[5])
thresholds[[6]] <- rep(0, level[6])

# Specify standard deviations for the Gaussians
sds <- rep(NULL, p)
sds[5:6] <- 1

# Create coefficient array
coefarray <- array(0, dim=c(p, p, max_level, max_level, n_lags))

# a.1) interaction between continuous 5<-6, lag=3
coefarray[5, 6, 1, 1, 2] <- .4
# a.2) interaction between 1<-3, lag=1
m1 <- matrix(0, nrow=level[2], ncol=level[4])
m1[1,1:2] <- 1
m1[2,3:4] <- 1
coefarray[1, 3, 1:level[2], 1:level[4], 1] <- m1
# a.3) interaction between 1<-5, lag=9
coefarray[1, 5, 1:level[1], 1:level[5], 3] <- c(0, 1)


# 2) Sample
set.seed(1)
dlist <- mvarsampler(coefarray = coefarray,
                     lags = lags,
                     thresholds = thresholds,
                     sds = sds,
                     type = type,
                     level = level,
                     N = 200,
                     pbar = TRUE)

# 3) Recover
set.seed(1)
mvar_obj <- mvar(data = dlist$data,
                 type = type,
                 level = level,
                 lambdaSel = "CV",
                 lags = c(1, 3, 9),
                 signInfo = FALSE,
                 overparameterize = F)

# Did we recover the true parameters?
mvar_obj$wadj[5, 6, 2] # cross-lagged effect of 6 on 2 over lag lags[2]
mvar_obj$wadj[1, 3, 1] # cross-lagged effect of 3 on 1 over lag lags[1]
```

```
mvar_obj$wadj[1, 5, 3] # cross-lagged effect of 1 on 5 over lag lags[3]

# How to get the exact parameter estimates?
# Example: the full parameters for the crossed-lagged interaction of 2 on 1 over lag lags[1]
mvar_obj$rawlags[[1]][[1]][[2]]

# 4) Predict / Compute nodewise Error
pred_mvar <- predict.mgm(mvar_obj, dlist$data)

head(pred_mvar$predicted) # first 6 rows of predicted values
pred_mvar$errors # Nodewise errors

# For more examples see https://github.com/jmbh/mgmDocumentation


## End(Not run)
```

---

mvarsampler *Sampling from a mixed VAR model*

Description

Function to sample from a mixed VAR (mVAR) model

Usage

```
mvarsampler(coefarray, lags, thresholds,
            sds, type, level, N, pbar)
```

Arguments

coefarray
: A p x p x max(level) x max(level) x n_lags array, where p are the number of variables, level is the input argument `level` and n_lags is the number of specified lags in `lags`, so n_lags = length(n_lags). The first four dimensions specify the parameters involved in the cross-lagged effects of the lag specified in the 5th dimension. I.e. coefarray[5, 6, 1, 1, 3] indicates the cross-lagged effect of variable 6 on variable 5 (if both are continuous), for the third lag specified in `lags`. If variable 1 and 3 are categorical with m = 2 and = 4 categories, respectively, then coefarray[1, 3, 1:2, 1:4, 1] indicates the m*s=8 parameters specifying this interaction for the first lag specified in `lags`. See the examples below for an illustration.

lags
: A vector indicating the lags in the mVAR model. E.g. `lags = c(1, 4, 9)` specifies lags of order 1, 3, 9. The number of specified lags has to match the 5th dimension in `coefarray`.

thresholds
: A list with p entries, each consisting of a vector indicating a threshold for each category of the given variable. For continuous variable, the vector has length 1.

sds	A vector of length p indicating the standard deviations of the included Gaussian nodes. If non-Gaussian variables are included in the mVAR model, the corresponding entries are ignored.
type	p vector indicating the type of variable for each column in data. "g" for Gaussian, "p" for Poisson, "c" for categorical.
level	p vector indicating the number of categories of each variable. For continuous variables set to 1.
N	The number of samples to be drawn from the specified mVAR model.
pbar	If pbar = TRUE, a progress bar is shown.

Details

We sample from the mVAR model by separately sampling from its corresponding p conditional distributions.

Value

A list with two entries:

call	The function call
data	The sampled n x p data matrix

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Examples

```
## Not run:


## Generate data from mixed VAR model using mvarsampler() and recover model using mvar()

# 1) Define mVAR model

p <- 6 # Six variables
type <- c("c", "c", "c", "c", "g", "g") # 4 categorical, 2 gaussians
level <- c(2, 2, 4, 4, 1, 1) # 2 categoricals with m=2, 2 categoricals with m=4, two continuous
max_level <- max(level)

lags <- c(1, 3, 9) # include lagged effects of order 1, 3, 9
n_lags <- length(lags)
```

```
# Specify thresholds
thresholds <- list()
thresholds[[1]] <- rep(0, level[1])
thresholds[[2]] <- rep(0, level[2])
thresholds[[3]] <- rep(0, level[3])
thresholds[[4]] <- rep(0, level[4])
thresholds[[5]] <- rep(0, level[5])
thresholds[[6]] <- rep(0, level[6])

# Specify standard deviations for the Gaussians
sds <- rep(NULL, p)
sds[5:6] <- 1

# Create coefficient array
coefarray <- array(0, dim=c(p, p, max_level, max_level, n_lags))

# a.1) interaction between continuous 5<-6, lag=3
coefarray[5, 6, 1, 1, 2] <- .4
# a.2) interaction between 1<-3, lag=1
m1 <- matrix(0, nrow=level[2], ncol=level[4])
m1[1,1:2] <- 1
m1[2,3:4] <- 1
coefarray[1, 3, 1:level[2], 1:level[4], 1] <- m1
# a.3) interaction between 1<-5, lag=9
coefarray[1, 5, 1:level[1], 1:level[5], 3] <- c(0, 1)


# 2) Sample
set.seed(1)
dlist <- mvarsampler(coefarray = coefarray,
                     lags = lags,
                     thresholds = thresholds,
                     sds = sds,
                     type = type,
                     level = level,
                     N = 200,
                     pbar = TRUE)

# 3) Recover
set.seed(1)
mvar_obj <- mvar(data = dlist$data,
                 type = type,
                 level = level,
                 lambdaSel = "CV",
                 lags = c(1, 3, 9),
                 signInfo = FALSE,
                 overparameterize = F)

# Did we recover the true parameters?
mvar_obj$wadj[5, 6, 2] # cross-lagged effect of 6 on 5 over lag lags[2]
mvar_obj$wadj[1, 3, 1] # cross-lagged effect of 3 on 1 over lag lags[1]
mvar_obj$wadj[1, 5, 3] # cross-lagged effect of 1 on 5 over lag lags[3]
```

```
# For more examples see https://github.com/jmbh/mgmDocumentation


## End(Not run)
```

plotRes *Plot summary of resampled sampling distributions*

Description

Plots a summary of sampling distributions resampled with the resample() function

Usage

```
plotRes(object, quantiles = c(.05, .95), labels = NULL,
        decreasing = TRUE, cut = NULL, cex.label = 0.75,
        lwd.qtl = 2, cex.mean = 0.55, cex.bg = 2.7,
        axis.ticks = c(-0.5, -0.25, 0, 0.25, 0.5, 0.75, 1),
        axis.ticks.mod = NULL, layout.width.labels = .2,
        layout.gap.pw.mod = .15, table = FALSE)
```

Arguments

`object`	An output object from the `resample()` function.
`quantiles`	A numerical vector of length two, specifying the desired lower/upper quantiles. Defaults to `quantiles = c(.05, .95)`.
`labels`	A character vector of length p, containing the label of each variable, where p is the number of variables.
`decreasing`	If `TRUE` (default), the edges are ordered by the arithmetic mean of the sampling distribution in decreasing order. If `FALSE` they are ordered in increasing order.
`cut`	A sequence of integers, specifying which edges are represented. For instance, if `decreasing = TRUE` and `cut = 1:10`, summaries for the 10 edges with the largest parameter estimate are displayed. The cut argument can also be used to present the boostrapped CIs in several figures.
`cex.label`	Text size of the labels.
`lwd.qtl`	Line width of line indicating the upper/lower quantiles.
`cex.mean`	Text size of the number indicating the proportion of the estimates whose absolute value is larger than zero.
`cex.bg`	Size of the white background of the number indicating the proportion of the estimates whose absolute value is larger than zero.
`axis.ticks`	A numeric vector indicating the axis ticks and labels for the x-axis.

`axis.ticks.mod` A numeric vector indicating the axis ticks and labels for the x-axis for moderation effects. If `axis.ticks.mod=NULL`, the values from `axis.ticks` for pairwise interactions are used.

`layout.width.labels`
A positive numeric value which specifies the width of the left-hand-side legend relative to the width of the data panel (or data panels, in case of a moderator model), which have width = 1. Defaults to `layout.width.labels = 0.2`.

`layout.gap.pw.mod`
A positive numeric value which specifies the width of the gap between the stability of pairwise effects and moderation effects. Defaults to `layout.gap.pw.mod = 0.15`.

`table` If `table = TRUE`, the output is presented as a table instead of a figure. Defaults to `table = FALSE`.

Details

Currently only supports summaries for resampled `mgm()` objects, and moderated MGMs with a single moderator.

Value

Plots a figure that shows summaries of the resampled sampling distribution for (a set of) all edge parameters. These include the mean, a specified upper and lower quantile and the proportion of parameter estimates whose absolute value is larger than zero.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

See Also

`resample()`, `mgm()`, `mvar()`, `tvmgm()`, `tvmar()`

Examples

```
## Not run:

# Fit initial model
fit_aut <- mgm(data = as.matrix(autism_data$data),
               type = autism_data$type,
               level = autism_data$lev,
               k = 2)


# Fit bootstrapped models
res_aut <- resample(object = fit_aut,
                    data = as.matrix(autism_data$data),
                    nB = 10) # should be more in real applications
```

```
# Plot Summary
plotRes(object = res_aut,
        quantiles = c(.05, .95),
        labels = NULL,
        axis.ticks = c(-.25, 0, .25, .5, .75))


## End(Not run)
```

---

predict.mgm *Compute predictions from mgm model objects*

Description

Computes predictions and prediction errors from a mgm model-object (mgm, mvar, tvmgm or tvmvar).

Usage

```
## S3 method for class 'mgm'
predict(object, data, errorCon, errorCat,
        tvMethod, consec, beepvar, dayvar, ...)
```

Arguments

object — An mgm model object (the output of one of the functions mgm(), mvar(), tvmgm() or tvmvar())

data — A n x p data matrix with the same structure (number of variables p and types of variables) as the data used to fit the model.

errorCon — Either a character vector specifying the types of nodewise errors that should be computed, where the two provided error functions for continuous varaibles are errorCon = "RMSE", the Root Mean Squared Error, and errorCon = "R2", the proportion of explained variance. The default is errorCon = c("RMSE" "R2").

Alternatively, errorCon can be a list, where each list entry is a custom error function of the form foo(true, pred), where true and pred are the arguments for the vectors of true and predicted values, respectively. If predictions are made for a time-varying model and tvMethod = "weighted", the weighted R2 or RMSE are computed. If a custom function is used, an additional argument for the weights has to be provided: foo(true, pred, weights). Note that custom error functios can also be combined with the buildt-in functions, i.e. errorCon = list("RMSE", "CustomError"=foo).

errorCat — Either a character vector specifying the types of nodewise errors that should be computed, where the two provided error functions for categorical variables are errorCat = "CC", the proportion of correct classification (accuracy) and errorCat = "nCC", the proportion of correct classification normalized by the marginal distribution of the variable at hand. Specifically, nCC = (CC - norm_constant) / (1 - norm_constant), where norm_constant is the highest relative frequency across categories. Another provided error is "CCmarg" which returns the accuracy of the intercept/marginal model. The default is to return all types of errors `errorCon = c("CC" "nCC", "CCmarg")`.

Alternatively, `errorCat` can be a list, where each list entry is a custom error function of the form `foo(true, pred)`, where `true` and `pred` are the arguments for the vectors of true and predicted values, respectively. If predictions are made for a time-varying model and `tvMethod = "weighted"`, the weighted R2 or RMSE are computed. If a custom function is used, an additional argument for the weights has to be provided: `foo(true, pred, weights)`. Note that custom error functios can also be combined with the buildt-in functions, i.e. `errorCon = list("nCC", "CustomError"=foo)`.

`tvMethod` Specifies how predictions and errors are computed for time-varying models: `tvMethod = "weighted"` computes errors by computing a weighted error over all cases in the time series at each estimation point specified in `estpoints` in `tvmgm()` or `tvmvar()`. The weighting corresponds to the weighting used for estimation (see `?tvmgm` or `?tvmvar`). `tvMethod = "closestModel"` determines for each time point the closest model and uses that model for prediction. See Details below for a more detailed explanation.

`consec` Only relevant for (time-varying) mVAR models. An integer vector of length `nrow(data)`, indicating the sequence of measurement points in a time series. This is only relevant for mVAR models and time series with unequal time intervals. Defaults to `consec = NULL`, which assumes equal time intervals. `consec` is ignored if a `mgm` or `tvmgm` object is provided to `predict.mgm()`. For details see `?mvar`.

`beepvar` Together with the argument `dayvar`, this argument is an alternative to the `consec` argument (see above) to specify the consecutiveness of measurements. This is tailored to ecological momentary assessment (EMA) studies, where the consectutiveness is defined by the number of notification on a given day (`beepvar`) and the given day (`dayvar`).

`dayvar` See `beepvar`.

`...` Additional arguments.

Details

Nodewise errors in time-varying models can be computed in two different ways: first, one computes the predicted value for each of the N cases in the time series for all models (estimated at different estimation points, see `?tvmgm` or `?tvmvar`). Then the error of each of the N cases for each of the models is weighted by the weight that has been used to estimate a given model at its estimation point. This means that the error of a data point close to the end of a time-series gets a high weight for models estimated in the end of the time-series and a low weight for models estimated in the beginning of the time series.

Second, we determine for each case in the time-series the closest estmation point, and use the model estimated at that estimation point to make predictions for that case.

Note that the error function normalized accuracy (nCC) is negative if the full model performs worse than the intercept model. This can happen if the model overfits the data.

Value

A list with the following entries:

- `call` Contains all provided input arguments.
- `predicted` A n x p matrix with predicted values, matching the dimension of the true values in `true`.
- `probabilities` A list with p entries corresponding to p nodes in the data. If a variable is categorical, the corresponding entry contains a n x k matrix with predicted probabilities, where k is the number of categories of the categorical variable. If a variable is continuous, the corresponding entry is empty.
- `true` Contains the true values. For `mgm` and `tvmgm` objects these are equal to the data provided via `data`. For `mvar` and `tvmvar` objects, these are equal to the rows that can be predicted in a VAR model, depending on the largest specified lag and (if specified) the `consec` argument.
- `errors` A matrix containing the all types of errors specified via `errorCon` and `errorCat`, for each variable. If `tvMethod = "weighted"`, the matrix becomes an array, with an additional dimension for the estimation point.
- `tverrors` If `tvMethod = "weighted"`, this list entry contains a list with errors of the format of `errors`, separately for each estimation point. The errors are computed from predictions of the model at the given estimation points and weighted by the weight-vector at that estimation point. If `tvMethod = "closestModel"`, this entry is empty.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Examples

```
## Not run:
# See examples in ?mgm, ?tvmgm, ?mvar and ?tvmvar.

## End(Not run)
```

print.int — *Print method for int objects*

Description

Returns basic information about objects created with showInteraction()

Usage

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

Arguments

`x`	The output object of showInteraction().
`...`	Additional arguments.

Value

Writes basic information about the object in the console.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

print.mgm — *Print method for mgm objects*

Description

Returns basic information about fit objects, prediction objects and bandwidth-selection objects.

Usage

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

Arguments

`x`	The output object of `mgm()`, `mvar()`, `tvmgm()`, `tvmvar()`, `predict.mgm()` or `bwSelect()`.
`...`	Additional arguments.

Value

Writes basic information about the object in the console.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

`resample` *Resampling scheme for mgm objects*

Description

Fits mgm model types (mgm, mvar, tvmgm, tvmvar) to a specified number of bootstrap samples.

Usage

```
resample(object, data, nB, blocks, quantiles,
         pbar, verbatim, ...)
```

Arguments

`object`	An mgm model object, the output of mgm(), tvmgm(), mvar(), tvmvar(). The model specifications for all fitted models are taken from this model object.
`data`	The n x p data matrix.
`nB`	The number of bootstrap samples.
`blocks`	The number of blocks for the block bootstrap used for time-varying models.
`quantiles`	A vector with two values in [0, 1], specifying quantiles computed on the bootstrapped sampling distributions. Defaults to `quantiles = c(.05, .95)`
`pbar`	If `TRUE`, a progress bar is shown. Defaults to `pbar = TRUE`.
`verbatim`	If `TRUE`, the seed of the current bootstrap sample is printed in the console. Useful to exclude zero-variance bootstrap samples in datasets with low variance.
`...`	Additional arguments.

Details

`resample()` fits a model specified via the `object` argument to `nB` bootstrap samples obtained from the orginial dataset. For stationary models (mgm() and mvar()) objects, we use the standard bootstrap. For time-varying models (tvmgm() and tvmvar()) we use the block bootstrap.

For mvar models, `bootParameters` is a p x p x nlags x nB array, where p is the number of variables, nlags is the number of specified lags, and nB is the number of bootstrap samples. Thus `bootParameters[7, 3, 2, ]` returns the bootstrapped sampling distribution of the lagged effect from variable 3 on 7 for the 2nd specified lag. See also `?mvar`.

For tvmar models, `bootParameters` is a p x p x nlags x nestpoints x nB array, analogously to mvar models. nestpoints is the number of specified estpoints. See also `?tvmvar`.

Resampling is currently only supported for pairwise MGMs (`k = 2`). For mgms, `bootParameters` is a p x p x nB array. For tvmgms, `bootParameters` is a p x p x nestpoint x nB array.

The seeds for the bootstrap samples are randomly sampled. For MGMs, the seeds are resampled until there are nB bootstrap samples on which an MGM can be estimated. This resampling has been implemented, because especially for small data sets, one can obtain bootstrap samples in which one or several variables have zero variance. For the other model classes, an informative error is returned in case the respective model cannot be estimated on one or more of the bootstrap samples.

Value

The output consists of a list with the entries

`call`	Contains the function call.
`models`	A list with `nB` entries, containing the models fit to the bootstrapped sampels.
`bootParameters`	Contains all the bootstrapped sampling distribution of all parameters. The dimension of this object depends on the type of model. Specifically, this object has the same dimension as the main parameter output of the corresponding estimation function, with one dimension added for the bootstrap iterations. See "Details" above.
`bootQuantiles`	Contains the specified quantiles of the bootstrapped sampling distribution for each parameter. Has the same structure as `bootParameters`. See "Details" above.
`Times`	Returns the running time for each bootstrap model in seconds.
`totalTime`	Returns the running time for all bootstrap models together in seconds.
`seeds`	`nB` integers indicating the seeds used to sample the `nB` bootstrap samples.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Examples

```
## Not run:


# 1) Fit mgm to example dataset (true edges: 1-4, 2-3, 1-2)
mgm_obj <- mgm(data = mgm_data$data,
               type = c('g', 'c', 'c', 'g'),
               level = c(1, 2, 4, 1),
               k = 2,
               lambdaSel = 'CV',
```

```
               threshold = 'none')

# 2) Take 50 bootstrap samples using resample()
res_obj <- resample(object = mgm_obj,
                    data = mgm_data$data,
                    nB = 50)

# 3) Plot histogram of bootstrapped sampling distribution of edge 1-4
hist(res_obj$bootParameters[1, 4, ],
     main = "",
     xlab = "Parameter Estimate")

# 4) Plot summary of all sampling distributions
plotRes(object = res_obj,
        quantiles = c(0.05, .95))


# For more examples see https://github.com/jmbh/mgmDocumentation




## End(Not run)
```

---

showInteraction *Retrieving details of interactions*

Description

Retrieves details of a specified interaction from mgm model objects.

Usage

```
showInteraction(object, int)
```

Arguments

object	The output of one of the estimation functions `mgm()`, `tvmgm()`, `mvar()`, `tvmvar()`.
int	An integer vector specifying the interaction. For mVAR models, this vector has length 2. For MGMs the vector can be larger to request details of interaction of order > 2.

Details

Currently the function only returns details of pairwise interactions from output objects of `mgm()`.

Value

`variables`	Integer vector returning the variables specified via the argument `int`
`type`	Character vector returning the type of the specified variables variables
`level`	Integer vector returning the number of levels of the specified variables variables
`parameters`	A list of length equal to the order k of the specified interaction. The entries contain the set of parameters obtained from the nodewise regressions on the k variables. Depending on the type of the variables in the interaction, these sets can obtain one or several parameters. For details see `?mgm` or Haslbeck & Waldorp (2017).

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

See Also

`mgm`, `tvmgm`, `mvar`, `tvmvar`

Examples

```
## Not run:


## We fit a pairwise and 3-order MGM to the mixed Autism dataset (?autism_data)

# 1) Fit Pairwise MGM

# Call mgm()
fit_d2 <- mgm(data = autism_data$data,
              type = autism_data$type,
              level = autism_data$lev,
              k = 2) # ad most pairwise interacitons

# Weighted adjacency matrix
fit_d2$pairwise$wadj # for instance, we see there is an interaction 1-2

# 2) Look at details of interaction 1-2
showInteraction(object = fit_d2,
                int = c(1, 2))

# For more examples see https://github.com/jmbh/mgmDocumentation
```

```
## End(Not run)
```

---

tvmgm *Estimating time-varying Mixed Graphical Models*

Description

Estimates time-varying k-order Mixed Graphical Models (MGMs) via elastic-net regularized kernel smoothed Generalized Linear Models

Usage

```
tvmgm(data, type, level, timepoints, estpoints, bandwidth, ...)
```

Arguments

`data`	n x p data matrix.
`type`	p vector indicating the type of variable for each column in `data`: "g" for Gaussian, "p" for Poisson, "c" for categorical.
`level`	p vector indicating the number of categories of each variable. For continuous variables set to 1.
`timepoints`	A strictly increasing numeric vector of length `nrow(data)` indicating the time points of the measurements in `data`. If `timepoints` is not specified, it is assumed that the time points are equally spaced. For details, see Haslbeck and Waldorp (2018).
`estpoints`	Vector indicating estimation points on the unit interval [0, 1] (the provided time scale is normalized interally to [0,1]).
`bandwidth`	We use a gaussian density on the unit time-interval [0,1] to determine the weights for each observation at each estimated time point. The bandwidth specifies the standard deviation the Gaussian density. To get some intuition, which bandwidth results in the combination of how many data close in time one can plot Gaussians on [0,1] for different bandwidths. The bandwidth can also be selected in a data driven way using the function (see `bwSelect`).
`...`	Arguments passed to `mgm`, specifying the MGM. See `?mgm`.

Details

Estimates a sequence of MGMs at the time points specified at the locations specified via `estpoints`. `tvmgm()` is a wrapper around `mgm()` and estimates a series of MGM with different weightings which are defined by the estimation locations in `estpoints` and the bandwidth parameter specified in `bandwidth`. For details see Haslbeck and Waldorp (2018).

Note that MGMs are not normalizable for all parameter values. See Chen, Witten & Shojaie (2015) for an overview of when pairwise MGMs are normalizable. To our best knowledge, for MGMs with interactions of order > 2 that include non-categorical variables, the conditions for normalizablity are unknown.

Value

A list with the following entries:

- `call` — Contains all provided input arguments. If `saveData = TRUE`, it also contains the data.
- `pairwise` — Contains a list with all information about estimated pairwise interactions. `wadj` contains a p x p x estpoints array containing the weighted adjacency matrix for each estimation point specified in `estpoints`, if p is the number of variables in the network. `signs` has the same dimensions as `wadj` and contains the signs for the entries of `wadj`: 1 indicates a positive sign, -1 a negative sign and 0 an undefined sign. A sign is undefined if an edge is a function of more than one parameter. This is the case for interactions involving a categorical variable with more than 2 categories. `edgecolor` also has the same dimensions as `wadj` contains a color for each edge, depending on `signs`. It is provided for more convenient plotting. If only pairwise interactions are modeled (`k = 2`), `wadj` contains all conditional independence relations.
- `interactions` — Contains a list with one entry for each estimation point specified in `estpoints`; each entry is a list with three entries that relate each interaction in the model to all its parameters. `indicator` contains a list with k-1 entries, one for each order of modeled interaction, which contain the estimated (nonzero) interactions. `weights` contains a list with k-1 entries, which in turn contain R lists, where R is the number of interactions (and rows in the corresponding list entry in`indicator`) that were estimated (nonzero) in the given entry. `signs` has the same structure as `weights` and provides the sign of the interaction, if defined.
- `intercepts` — Contains a list with one entry for each estimation point specified in `estpoints`; each entry is a list with p entries, which contain the intercept/thresholds for each node in the network. In case a given node is categorical with m categories, there are m thresholds for this variable (one for each category).
- `tvmodels` — Contains the MGM model estimated by `mgm()` at each time point specified via `estpoints`. See `?mgm` for a detailed description of this output.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Chen S, Witten DM & Shojaie (2015). Selection and estimation for mixed graphical models. Biometrika, 102(1), 47.

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Yang, E., Baker, Y., Ravikumar, P., Allen, G. I., & Liu, Z. (2014, April). Mixed Graphical Models via Exponential Families. In AISTATS (Vol. 2012, pp. 1042-1050).

Examples

```
## Not run:


## We specify a time-varying MGM and recover it using tvmgm()

# 1) Specify Model

# a) Define Graph
p <- 6
type = c("c", "c", "g", "g", "p", "p")
level = c(2, 3, 1, 1, 1, 1)
n_timepoints <- 1000

# b) Define Interaction
factors <- list()
factors[[1]] <- matrix(c(1,2,
                         2,3,
                         3,4), ncol=2, byrow = T)  # no pairwise interactions
factors[[2]] <- matrix(c(1,2,3,
                         2,3,4), ncol=3, byrow = T) # one 3-way interaction

interactions <- list()
interactions[[1]] <- vector("list", length = 3)
interactions[[2]] <- vector("list", length = 2)
# 3 2-way interactions
interactions[[1]][[1]] <- array(0, dim = c(level[1], level[2], n_timepoints))
interactions[[1]][[2]] <- array(0, dim = c(level[2], level[3], n_timepoints))
interactions[[1]][[3]] <- array(0, dim = c(level[3], level[4], n_timepoints))
# 2 3-way interactions
interactions[[2]][[1]] <- array(0, dim = c(level[1], level[2], level[3], n_timepoints))
interactions[[2]][[2]] <- array(0, dim = c(level[2], level[3], level[4], n_timepoints))
theta <- .3
interactions[[1]][[1]][1, 1, ] <- theta
interactions[[1]][[2]][1, 1, ] <- theta
interactions[[1]][[3]][1, 1, ] <- seq(0, theta, length = n_timepoints)
interactions[[2]][[1]][1, 1, 1, ] <- theta
interactions[[2]][[2]][1, 1, 1, ] <- theta
# c) Define Thresholds
thresholds <- list()
thresholds[[1]] <- matrix(0, nrow = n_timepoints, ncol= level[1])
thresholds[[2]] <- matrix(0, nrow = n_timepoints, ncol= level[2])
thresholds[[3]] <- matrix(0, nrow = n_timepoints, ncol= level[3])
thresholds[[4]] <- matrix(0, nrow = n_timepoints, ncol= level[4])
thresholds[[5]] <- matrix(.1, nrow = n_timepoints, ncol= level[5])
thresholds[[6]] <- matrix(.1, nrow = n_timepoints, ncol= level[6])
# d) define sds
sds <- matrix(.2, ncol=p, nrow=n_timepoints)

# 2) Sample Data
set.seed(1)
```

```
d_iter <- tvmgmsampler(factors = factors,
                       interactions = interactions,
                       thresholds = thresholds,
                       sds = sds,
                       type = type,
                       level = level,
                       nIter = 100,
                       pbar = TRUE)

data <- d_iter$data
head(data)
# delete inf rows:
ind_finite <- apply(data, 1, function(x) if(all(is.finite(x))) TRUE else FALSE)
table(ind_finite) # all fine for this setup & seed
# in case of inf values (no theory on how to keep k-order MGM well-defined)
data <- data[ind_finite, ]


# 3) Recover
mgm_c_cv <- tvmgm(data = data,
                  type = type,
                  level = level,
                  k = 3,
                  estpoints = seq(0, 1, length=10),
                  bandwidth = .1,
                  lambdaSel = "CV",
                  ruleReg = "AND",
                  pbar = TRUE,
                  overparameterize = T,
                  signInfo = FALSE)

# Look at time-varying pairwise parameter 3-4
mgm_c_cv$pairwise$wadj[3,4,] # recovers increase

# 4) Predict values / compute nodewise Errors
pred_mgm_cv_w <- predict.mgm(mgm_c_cv,
                             data = data,
                             tvMethod = "weighted")
pred_mgm_cv_cM <- predict.mgm(mgm_c_cv,
                              data = data,
                              tvMethod = "closestModel")

pred_mgm_cv_w$errors
pred_mgm_cv_cM$errors # Pretty similar!


# For more examples see https://github.com/jmbh/mgmDocumentation


## End(Not run)
```

`tvmgmsampler` *Sample from time-varying k-order Mixed Graphical Model*

Description

Generates samples from a time-varying k-order Mixed Graphical Model

Usage

```
tvmgmsampler(factors, interactions, thresholds, sds, type,
             level, nIter = 250, pbar = TRUE, ...)
```

Arguments

`factors`	The same object as `factors` in `mgmsampler()`. An interaction is specified in `factors` if it should be nonzero at least at one time point in the time series. The values of each parameter at each time point is specified via `interactions`.
`interactions`	The same object as `factors` in `mgmsampler()`, except that each array indicating the parameters of an interaction has an additional (the last) dimension, indicating time. Corresponding to the time vector in `factors`, the time vector has to be a sequence of integers {1, 2, ..., N}. For an illustration see the examples below.
`thresholds`	A list with p entries for p variables, each of which contains a N x m matrix. The columns contain the m thresholds for m categories (for continuous variables m = 1 and the entry contains the threshold/intercept). The rows indicate how the thresholds change over time.
`sds`	N x p matrix indicating the standard deviations of Gaussians specified in `type` for {1, ..., N} time points. Entries not referring to Gaussians are ignored.
`type`	p character vector indicating the type of variable for each column in `data`. "g" for Gaussian, "p" for Poisson, "c" of each variable.
`level`	p integer vector indicating the number of categories of each variable. For continuous variables set to 1.
`nIter`	Number of iterations in the Gibbs sampler until a sample is drawn.
`pbar`	If `pbar = TRUE` a progress bar is shown. Defaults to `pbar = TRUE`.
`...`	Additional arguments.

Details

`tvmgmsampler` is a wrapper function around `mgmsampler`. Its input is the same as for `mgmsampler`, except that each object has an additional dimension for time. The number of time points is specified via entries in the additional time dimension.

Value

A list containing:

`call`	Contains all provided input arguments.
`data`	The N x p data matrix of sampled values

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Yang, E., Baker, Y., Ravikumar, P., Allen, G. I., & Liu, Z. (2014, April). Mixed Graphical Models via Exponential Families. In AISTATS (Vol. 2012, pp. 1042-1050).

Examples

```
## Not run:

# --------- Example 1: p = 4 dimensional Gaussian ---------

# ----- 1) Specify Model -----

# a) General Graph Info
type = c("g", "g", "g", "g") # Four Gaussians
level = c(1, 1, 1, 1)
n_timepoints = 500 #  Number of time points

# b) Define Interaction
factors <- list()
factors[[1]] <- array(NA, dim=c(2, 2)) # two pairwise interactions
factors[[1]][1, 1:2] <- c(3,4)
factors[[1]][2, 1:2] <- c(1,2)

# Two parameters, one linearly increasing from 0 to 0.8, another one lin decreasing from 0.8 to 0
interactions <- list()
interactions[[1]] <- vector("list", length = 2)
interactions[[1]][[1]] <- array(0, dim = c(level[1], level[2], n_timepoints))
interactions[[1]][[1]][1,1, ] <- seq(.8, 0, length = n_timepoints)
interactions[[1]][[2]] <- array(0, dim = c(level[1], level[2], n_timepoints))
interactions[[1]][[2]][1,1, ] <- seq(0, .8, length = n_timepoints)

# c) Define Thresholds
thresholds <- vector("list", length = 4)
thresholds <- lapply(thresholds, function(x) matrix(0, ncol = level[1], nrow = n_timepoints))

# d) Define Standard deviations
sds <- matrix(1, ncol = length(type), nrow = n_timepoints) # constant across variables and time


# ----- 2) Sample cases -----

set.seed(1)
```

```
dlist <- tvmgmsampler(factors = factors,
                      interactions = interactions,
                      thresholds = thresholds,
                      sds = sds,
                      type = type,
                      level = level,
                      nIter = 75,
                      pbar = TRUE)


# ----- 3) Recover model from sampled cases -----

set.seed(1)
tvmgm_obj <- tvmgm(data = dlist$data,
                   type = type,
                   level = level,
                   estpoints = seq(0, 1, length = 15),
                   bandwidth = .2,
                   k = 2,
                   lambdaSel = "CV",
                   ruleReg = "AND")

# How well did we recover those two time-varying parameters?
plot(tvmgm_obj$pairwise$wadj[3,4,], type="l", ylim=c(0,.8))
lines(tvmgm_obj$pairwise$wadj[1,2,], type="l", col="red")
# Looks quite good


# --------- Example 2: p = 5 binary; one 3-way interaction ---------

# ----- 1) Specify Model -----

# a) General Graph Info
p <- 5 # number of variables
type = rep("c", p) # all categorical
level = rep(2, p) # all binary
n_timepoints <- 1000

# b) Define Interaction
factors <- list()
factors[[1]] <- NULL # no pairwise interactions
factors[[2]] <- array(NA, dim = c(1,3)) # one 3-way interaction
factors[[2]][1, 1:3] <- c(1, 2, 3)

interactions <- list()
interactions[[1]] <- NULL # no pairwise interactions
interactions[[2]] <- vector("list", length = 1)  # one 3-way interaction
# 3-way interaction no1
interactions[[2]][[1]] <- array(0, dim = c(level[1], level[2], level[3], n_timepoints))
theta <- 2
interactions[[2]][[1]][1, 1, 1, ] <- seq(0, 2, length = n_timepoints) # fill in nonzero entries

# c) Define Thresholds
```

```
thresholds <- list()
for(i in 1:p) thresholds[[i]] <- matrix(0, nrow = n_timepoints, ncol = level[i])


# ----- 2) Sample cases -----

set.seed(1)
dlist <- tvmgmsampler(factors = factors,
                      interactions = interactions,
                      thresholds = thresholds,
                      type = type,
                      level = level,
                      nIter = 150,
                      pbar = TRUE)


# ----- 3) Check Marginals -----

dat <- dlist$data[1:round(n_timepoints/2),]
table(dat[,1], dat[,2], dat[,3])

dat <- dlist$data[round(n_timepoints/2):n_timepoints,]
table(dat[,1], dat[,2], dat[,3])

# Observation: much stronger effect in second hald of the time-series,
# which is what we expect


# ----- 4) Recover model from sampled cases -----

set.seed(1)
tvmgm_obj <- tvmgm(data = dlist$data,
                   type = type,
                   level = level,
                   estpoints = seq(0, 1, length = 15),
                   bandwidth = .2,
                   k = 3,
                   lambdaSel = "CV",
                   ruleReg = "AND")

tvmgm_obj$interactions$indicator
# Seems very difficult to recover this time-varying 3-way binary interaction
# See also the corresponding problems in the examples of ?mgmsampler


# For more examples see https://github.com/jmbh/mgmDocumentation



## End(Not run)
```

tvmvar *Estimating time-varying Mixed Vector Autoregressive Model (mVAR)*

Description

Estimates time-varying Mixed Vector Autoregressive Model (mVAR) via elastic-net regularized kernel smoothed Generalized Linear Models

Usage

```
tvmvar(data, type, level, timepoints, estpoints, bandwidth, ...)
```

Arguments

`data`	n x p data matrix.
`type`	p vector indicating the type of variable for each column in `data`: "g" for Gaussian, "p" for Poisson, "c" for categorical.
`level`	p vector indicating the number of categories of each variable. For continuous variables set to 1.
`timepoints`	A strictly increasing numeric vector of length `nrow(data)` indicating time points for the measurements in `data`. If `timepoints` is not specified, it is assumed that the time points are equally spaced. For details, see Haslbeck and Waldorp (2018).
`estpoints`	Vector indicating estimation points on interval [0, 1]. Note that we define this unit interval on the entire time series. This also includes measurements that are excluded because not enough previous measurements are available to fit the model. This ensures that the a model estimated at, for example, estimation point 0.15 is actually estimated on data close to data points around this time point. See Haslbeck and Waldorp (2018) Section 2.5 and 3.4 for a detailed description.
`bandwidth`	We use a gaussian density on the unit time-interval [0,1] to determine the weights for each observation at each estimated time point. The bandwidth specifies the standard deviation the Gaussian density. To get some intuition, which bandwidth results in the combination of how many data close in time one can plot Gaussians on [0,1] for different bandwidths. The bandwidth can also be selected in a data driven way using the function (see `bwSelect`).
`...`	Arguments passed to `mvar`, specifying how each single model should be estimated. See `?mvar`.

Details

Estimates a sequence of mVAR models at the time points specified at the locations specified via `estpoints`. `tvmvar()` is a wrapper around `mvar()` and estimates a series of MGM with different weightings which are defined by the estimation locations in `estpoints` and the banwdith parameter specified in `bandwidth`. For details see Haslbeck and Waldorp (2018)

Value

A list with the following entries:

`call`	Contains all provided input arguments. If `saveData = TRUE`, it also contains the data.
`wadj`	A p x p x n_lags x S array, where n_lags is the number of specified lags in `lags` (see `?mvar`) and S is the number of estimation points specified in `estpoints`. For instance, wadj[1, 2, 1, 10] is the cross-lagged predicting variable 1 at time point t by variable 2 at time point t - z, where z is specified by the first lag specified in `lags` (see `?mvar`), in the model estimated at estimation point 10.
`signs`	Has the same structure as `wadj` and specifies the signs corresponding to the parameters in `wadj`, if defined. 1/-1 indicate positive and negative relationships, respectively. 0 indicates that no sign is defined, which is the case for interactions that involve a categorical variable where an interaction can have more than one parameter. If `binarySign = TRUE`, a sign is calculated for interactions between binary variables and binary and continuous variables, where the interaction is still defined by one parameter and hence a sign can be specified. `NA` indicates that the corresponding parameter in `wadj` is zero. See also `?mvar`.
`intercepts`	A list with S entries, where S is the number of estimated time points. Each entry of that list contains a list p entries with the intercept/thresholds for each node in the network. In case a given node is categorical with m categories, there are m thresholds for this variable.
`tvmodels`	Contains the mVAR model estimated by `mvar()` at each time point specified via `estpoints`. See `?mvar` for a detailed description of this output.

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Examples

```
## Not run:


## We set up the same model as in the example of mvar(), but
## specify one time-varying parameter, and try to recover it with
## tvmvar()

# a) Specify time-varying VAR model

p <- 6 # Six variables
type <- c("c", "c", "c", "c", "g", "g") # 4 categorical, 2 gaussians
```

```
level <- c(2, 2, 4, 4, 1, 1) # 2 categoricals with 2 categories, 2 with 5
max_level <- max(level)
n_timepoints <- 4000

lags <- c(1, 3, 9) # include lagged effects of order 1, 3, 9
n_lags <- length(lags)

# Specify thresholds
thresholds <- list()
thresholds[[1]] <- matrix(0, ncol=level[1], nrow=n_timepoints)
thresholds[[2]] <- matrix(0, ncol=level[2], nrow=n_timepoints)
thresholds[[3]] <- matrix(0, ncol=level[3], nrow=n_timepoints)
thresholds[[4]] <- matrix(0, ncol=level[4], nrow=n_timepoints)
thresholds[[5]] <- matrix(0, ncol=level[5], nrow=n_timepoints)
thresholds[[6]] <- matrix(0, ncol=level[6], nrow=n_timepoints)

# Specify standard deviations for the Gaussians
sds <- matrix(NA, ncol=p, nrow=n_timepoints)
sds[, 5:6] <- 1

# Create coefficient array
coefarray <- array(0, dim=c(p, p, max_level, max_level, n_lags, n_timepoints))

# a.1) interaction between continuous 5<-6, lag=3
coefarray[5, 6, 1, 1, 2, ] <- seq(0, .4, length = n_timepoints) # only time-varying parameter
# a.2) interaction between 1<-3, lag=1
m1 <- matrix(0, nrow=level[2], ncol=level[4])
m1[1,1:2] <- 1
m1[2,3:4] <- 1
coefarray[1, 3, 1:level[2], 1:level[4], 1, ] <- m1 # constant across time
# a.3) interaction between 1<-5, lag=9
coefarray[1, 5, 1:level[1], 1:level[5], 3, ] <- c(0, 1) # constant across time


# b) Sample
set.seed(1)
dlist <- tvmvarsampler(coefarray = coefarray,
                       lags = lags,
                       thresholds = thresholds,
                       sds = sds,
                       type = type,
                       level = level,
                       pbar = TRUE)

# c.1) Recover: stationary
set.seed(1)
mvar_obj <- mvar(data = dlist$data,
                 type = type,
                 level = level,
                 lambdaSel = "CV",
                 lags = c(1, 3, 9),
                 signInfo = FALSE)
```

```
# Did we recover the true parameters?
mvar_obj$wadj[5, 6, 2] # cross-lagged effect of 6 on 5 over lag lags[2] (lag 3)
mvar_obj$wadj[1, 3, 1] # cross-lagged effect of 3 on 1 over lag lags[1] (lag 1)
mvar_obj$wadj[1, 5, 3] # cross-lagged effect of 1 on 5 over lag lags[3] (lag 9)


# c.2) Recover: time-varying
set.seed(1)
mvar_obj <- tvmvar(data = dlist$data,
                   type = type,
                   level = level,
                   estpoints = seq(0, 1, length=10),
                   bandwidth = .15,
                   lambdaSel = "CV",
                   lags = c(1, 3, 9),
                   signInfo = FALSE)

# Did we recover the true parameters?
mvar_obj$wadj[5, 6, 2, ] # true sort of recovered
mvar_obj$wadj[1, 3, 1, ] # yes
mvar_obj$wadj[1, 5, 3, ] # yes

# Plotting parameter estimates over time
plot(mvar_obj$wadj[5, 6, 2, ],
     type="l", ylim=c(-.2,.7),
     lwd=2, ylab="Parameter value", xlab="Estimation points")
lines(mvar_obj$wadj[1, 3, 1, ], col="red", lwd=2)
lines(mvar_obj$wadj[1, 5, 3, ], col="blue", lwd=2)
legend("bottomright", c("5 <-- 6", "1 <-- 3", "1 <-- 5"),
       lwd = c(2,2,2), col=c("black", "red", "blue"))


# d) Predict values / compute nodewise error

mvar_pred_w <- predict.mgm(object=mvar_obj,
                           data=dlist$data,
                           tvMethod = "weighted")

mvar_pred_cM <- predict.mgm(object=mvar_obj,
                            data=dlist$data,
                            tvMethod = "closestModel")

mvar_pred_w$errors
mvar_pred_cM$errors

# For more examples see https://github.com/jmbh/mgmDocumentation



## End(Not run)
```

`tvmvarsampler` *Sampling from a time-varying mixed VAR model*

Description

Function to sample from a time-varying mixed VAR (mVAR) model

Usage

```
tvmvarsampler(coefarray, lags, thresholds,
              sds, type, level, pbar)
```

Arguments

`coefarray`	A p x p x max(level) x max(level) x n_lags x N array, where p are the number of variables, level is the input argument `level` and n_lags is the number of specified lags in `lags`, so n_lags = length(n_lags), and N is the number of time points in the time series. The first four dimensions specify the parameters involved in the cross-lagged effects of the lag specified in the 5th dimension. I.e. coefarray[5, 6, 1, 1, 3, 100] indicates the cross-lagged effect of variable 6 on variable 5 (if both are continuous), for the third lag specified in `lags` at time point 100. If variable 1 and 3 are categorical with m = 2 and = 4 categories, respectively, then coefarray[1, 3, 1:2, 1:4, 1, 250] indicates the m*s=8 parameters specifying this interaction for the first lag specified in `lags` at time point 250. See the examples below for an illustration.
`lags`	A vector indicating the lags in the mVAR model. E.g. `lags = c(1, 4, 9)` specifies lags of order 1, 3, 9. The number of specified lags has to match the 5th dimension in `coefarray`.
`thresholds`	A list with p entries, each consisting of a matrix indicating a threshold for each category of the given variable (column) and time point (row). For continuous variable, the matrix has 1 column.
`sds`	A N x p matrix specifying the standard deviation of Gaussian variables (columns) at each time point (rows)If non-Gaussian variables are included in the mVAR model, the corresponding columns are ignored.
`type`	p vector indicating the type of variable for each column in `data`. "g" for Gaussian, "p" for Poisson, "c" for categorical.
`level`	p vector indicating the number of categories of each variable. For continuous variables set to 1.
`pbar`	If `pbar = TRUE`, a progress bar is shown.

Details

We sample from the mVAR model by separately sampling from its corresponding p conditional distributions.

Value

A list with two entries:

`call`	The function call
`data`	The sampled n x p data matrix

Author(s)

Jonas Haslbeck <jonashaslbeck@gmail.com>

References

Haslbeck, J. M. B., & Waldorp, L. J. (2020). mgm: Estimating time-varying Mixed Graphical Models in high-dimensional Data. Journal of Statistical Software, 93(8), pp. 1-46. DOI: 10.18637/jss.v093.i08

Examples

```
## Not run:

## We specify a tvmvar model, sample from it and recover it

# a) Set up time-varying mvar model

p <- 6 # Six variables
type <- c("c", "c", "c", "c", "g", "g") # 4 categorical, 2 gaussians
level <- c(2, 2, 4, 4, 1, 1) # 2 categoricals with 2 categories, 2 with 5
max_level <- max(level)

lags <- c(1, 3, 9) # include lagged effects of order 1, 3, 9
n_lags <- length(lags)

N <- 5000

# Specify thresholds
thresholds <- list()
thresholds[[1]] <- matrix(0, ncol=2, nrow=N)
thresholds[[2]] <- matrix(0, ncol=2, nrow=N)
thresholds[[3]] <- matrix(0, ncol=4, nrow=N)
thresholds[[4]] <- matrix(0, ncol=4, nrow=N)
thresholds[[5]] <- matrix(0, ncol=1, nrow=N)
thresholds[[6]] <- matrix(0, ncol=1, nrow=N)

# Specify standard deviations for the Gaussians
sds <- matrix(NA, ncol=6, nrow=N)
sds[,5:6] <- 1

# Create coefficient array
coefarray <- array(0, dim=c(p, p, max_level, max_level, n_lags, N))
```

```
# a.1) interaction between continuous 5<-6, lag=3
coefarray[5, 6, 1, 1, 2, ] <- c(rep(.5, N/2), rep(0, N/2))
# a.2) interaction between 1<-3, lag=1
m1 <- matrix(0, nrow=level[2], ncol=level[4])
m1[1, 1:2] <- 1
m1[2, 3:4] <- 1
coefarray[1, 3, 1:level[2], 1:level[4], 1, ] <- m1
# a.3) interaction between 1<-5, lag=9
coefarray[1, 5, 1:level[1], 1:level[5], 3, ] <- c(0, 1)

dim(coefarray)


# b) Sample
set.seed(1)
dlist <- tvmvarsampler(coefarray = coefarray,
                       lags = lags,
                       thresholds = thresholds,
                       sds = sds,
                       type = type,
                       level = level,
                       pbar = TRUE)


# c) Recover: time-varying mVAR model
set.seed(1)
tvmvar_obj <- tvmvar(data = dlist$data,
                     type = type,
                     level = level,
                     lambdaSel = "CV",
                     lags = c(1, 3, 9),
                     estpoints = seq(0, 1, length=10),
                     bandwidth = .05)

tvmvar_obj$wadj[5, 6, 2, ] # parameter goes down, as specified
tvmvar_obj$wadj[1, 3, 1, ]
tvmvar_obj$wadj[1, 5, 3, ]

# For more examples see https://github.com/jmbh/mgmDocumentation


## End(Not run)
```

Index