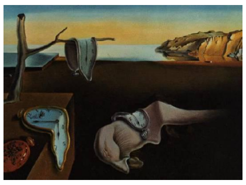
mgm: Structure Estimation for Time-Varying Mixed Graphical Models in high-dimensional Data

Haslbeck, J.M.B.; Waldorp, L.J.

DOI
10.18637/jss.v093.i08
Publication date
2020
Document Version
Final published version
Published in
Journal of Statistical Software
License
CC BY

Link to publication



Journal of Statistical Software

April 2020, Volume 93, Issue 8. *doi: 10.18637/jss.v093.i08*

mgm: Estimating Time-Varying Mixed Graphical Models in High-Dimensional Data

Jonas M. B. Haslbeck
University of Amsterdam

Lourens J. Waldorp
University of Amsterdam

Abstract

We present the R package **mgm** for the estimation of k-order mixed graphical models (MGMs) and mixed vector autoregressive (mVAR) models in high-dimensional data. These are a useful extensions of graphical models for only one variable type, since data sets consisting of mixed types of variables (continuous, count, categorical) are ubiquitous. In addition, we allow to relax the stationarity assumption of both models by introducing time-varying versions of MGMs and mVAR models based on a kernel weighting approach. Time-varying models offer a rich description of temporally evolving systems and allow to identify external influences on the model structure such as the impact of interventions. We provide the background of all implemented methods and provide fully reproducible examples that illustrate how to use the package.

Keywords: structure estimation, mixed graphical models, Markov random fields, dynamic graphical models, time-varying graphical models, vector autoregressive models.

1. Introduction

We present **mgm** (Haslbeck 2020), an R (R Core Team 2020) package for the estimation of (time-varying) k-order mixed graphical models (MGMs) and (time-varying) mixed vector autoregressive (mVAR) models with a specified set of lags. The package is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=mgm`. In this paper we introduce these models, discuss algorithms to estimate them, and present a number of fully reproducible code examples that show how to use the implementations provided by **mgm**.

Graphical models have become a popular way to abstract complex systems and gain insights into relational patterns among observed variables in a large variety of disciplines such as statistical mechanics (Albert and Barabasi 2002), biology (Friedman, Linial, Nachman, and

Pe'er 2000), genetics (Ghazalpour *et al.* 2006), neuroscience (Huang *et al.* 2010) and psychology (Borsboom and Cramer 2013). In many of these applications the data set of interest consists of *mixed variables* such as binary, categorical, ordinal, counts, continuous and/or skewed continuous amongst others. One example is internet-scale marketing data, where it is of interest to relate variables such as clicked links (categorical), time spent on websites (possibly exponential), browsing history (categorical), social media postings (count), friends in social networks (count), and many others. In a medical context, one could be interested in interactions between person characteristics such as gender (categorical) or age (continuous), frequencies of behaviors (count), taking place of events (categorical) and the dose of a drug (continuous).

If measurements are taken repeatedly from a system, one can be either interested in relations between variables at the *same time point* or in relations between variables *across time points*. The former relations are modeled by MGMs, the latter relations are modeled by vector autoregressive (VAR) models, which relate variables over a specified set of time lags. For both types of models it may be appropriate in some situations to relax the assumption of stationarity, such that its parameters are allowed to vary over the measured time period. These time-varying models provide additional information for understanding and predicting organizational processes, the diffusion of information, detecting vulnerabilities and the potential impact of interventions. An example is the developmental cycle of a biological organism, in which different genes interact at different stages of development. In a medical context, the aim could be to study the impact of an intervention on the dependencies between a large number of physiological and psychological variables that capture the health of a patient. Yet another example can be found in the field of psychiatry, where one might be interested in the interaction of negative life events, social contacts and symptoms of psychological disorders such as major depression.

1.1. Implementation and functionality

The **mgm** package is written in R and uses the **glmnet** package (Friedman, Hastie, and Tibshirani 2010) to fit penalized generalized linear models (GLMs) to perform neighborhood selection (Meinshausen and Bühlmann 2006). The **glmnet** package is written in Fortran and is optimized for computational efficiency. In addition, **mgm** depends on the packages **matrixcalc** (Novomestky 2012), **stringr** (Wickham 2019), **Hmisc** (Harrell Jr 2020), **gtools** (Warnes, Bolker, and Lumley 2018) and **qgraph** (Epskamp, Cramer, Waldorp, Schmittmann, and Borsboom 2012).

The main functionality of the **mgm** package is to estimate mixed graphical models (MGMs) and mixed autoregressive (mVAR) models, both as stationary models (`mgm()` and `mvar()`) and time-varying models (`tvmgm()` and `tvmvar()`). In addition, we provide the S3 methods `print()` to summarize model objects and `predict()` to compute predictions and nodewise errors from all types of models, and the function `resample()` to determine the stability of estimates via resampling. Furthermore, **mgm** provides functions to sample from all four models in full flexibility in order to enable the user to investigate the performance of the estimation algorithms in a particular situation. The output of all estimation functions is designed to allow a seamless visualization with the **qgraph** package (Epskamp *et al.* 2012) and we therefore do not provide our own plotting functions.

1.2. Related implementations

Several packages are available to estimate Gaussian graphical models (GGMs): The R packages **glasso** (Friedman, Hastie, and Tibshirani 2019) and **huge** (Jiang *et al.* 2019; Zhao, Liu, Roeder, Lafferty, and Wasserman 2012) implement the graphical lasso (Banerjee, El Ghaoui, and d'Aspremont 2008; Friedman, Hastie, and Tibshirani 2008) which maximizes an ℓ_1-penalized Gaussian log-likelihood. The **huge** package also allows to estimate GGMs via neighborhood selection (Meinshausen and Bühlmann 2006), in which the neighborhood of each node is estimated separately and then the local estimates are combined to obtain the (global) graphical model. The R package **IsingFit** (Van Borkulo and Epskamp 2016; Van Borkulo *et al.* 2014) implements a neighborhood selection based method to estimate the binary-valued Ising model (see, e.g., Wainwright and Jordan 2008; Ravikumar, Wainwright, and Lafferty 2010). The **XMRF** package (Wan, Allen, Baker, Yang, Ravikumar, and Liu 2015) allows to estimate Markov random fields of the multivariate Gaussian distribution, Ising models, log-linear Poisson based graphical model, regular Poisson graphical models, truncated Poisson graphical models and sublinear Poisson graphical models (Yang, Ravikumar, Allen, and Liu 2015, 2013).

For VAR models, the **vars** package (Pfaff 2008b) allows to fit VAR models with Gaussian noise. The **BigVAR** package (Nicholson, Matteson, and Bien 2019) allows to fit VAR models and structured VAR models with Gaussian noise with structured ℓ_1-penalties. The **mlVAR** package (Epskamp, Deserno, and Bringmann 2019) implements multilevel VAR models with Gaussian noise. Graphical VAR models (Wild, Eichler, Friederich, Hartmann, Zipfel, and Herzog 2010), in which lagged coefficients and contemporaneous effects are estimated simultaneously, can be estimated with the **graphicalVAR** package (Epskamp 2018).

For time-varying graphical models, there is a Python (Van Rossum *et al.* 2011) implementation of the SINGLE algorithm of Monti, Hellyer, Sharp, Leech, Anagnostopoulos, and Montana (2014) for time-varying Gaussian graphical models (Monti 2014) and **GraphTime** (Immer and Gibberd 2017), a Python implementation of time-varying (dynamic) graphical models based on the (group) fused-lasso as presented by Gibberd and Nelson (2017). The R package **tvReg** allows to estimate linear VAR models via kernel smoothing (Casas and Fernandez-Casal 2020).

mgm goes beyond the above mentioned packages in that it allows to estimate k-order MGMs and mVAR models (with any set of lags), compute predictions from them and assess model stability via resampling, while the above packages only allow to do this for special cases. In addition, the output of **mgm** is designed to allow a seamless visualization of estimated models using the R package **qgraph** (Epskamp *et al.* 2012). Finally, **mgm** is the first package that allows to estimate time-varying MGMs and mVAR models.

1.3. Overview of the paper

In Section 2, we introduce mixed graphical models (MGMs; Section 2.2) and mixed vector autoregressive (mVAR) models (Section 2.4), and discuss how to estimate these models in their stationary (Section 2.3) and time-varying (Section 2.5) versions. In Section 3, we illustrate how to use the **mgm** package to estimate parameters, compute predictions from and visualize stationary MGMs (Section 3.1), stationary mVAR models (Section 3.2), time-varying MGMs (Section 3.3) and time-varying mVAR models (Section 3.4). All presented examples are fully reproducible, with code either shown in the paper or provided in the online supplementary material.

2. Background

In this section we provide basic concepts related to graphical models (Section 2.1), introduce the model classes mixed graphical models (MGMs; Section 2.2) and mixed vector autoregressive (mVAR) models (Section 2.4), and show how to estimate these models in their stationary (Section 2.3) and time-varying (Section 2.5) versions.

2.1. Graphical models

Undirected graphical models are families of probability distributions that respect a set of conditional independence statements represented in an undirected graph G (Lauritzen 1996). This connection between probability distribution and graph G is formalized by the *global Markov property*, which we define after introducing some notation.

An undirected graph $G = (V, E)$ consists of a collection of nodes $V = \{1, 2, \dots, p\}$ and a collection of edges $E \subset V \times V$. A subset of nodes U is a *node cutset* whenever its removal breaks the graph in two or more nonempty subsets, which is equivalent to U being the set such that all paths from disjoint node sets S and Q go through U (Lauritzen 1996). A *clique* is a subset $C \subseteq V$ such that $(s, t) \in E$ for all $s, t \in C$ where $s \neq t$, and is called a *maximal clique* if inclusion of any other node would make it not a clique. The neighborhood $N(s)$ of a node $s \in V$ is the set of nodes that are connected to s by an edge, $N(s) := \{t \in V | (s, t) \in E\}$. Throughout the paper we use the shorthand $X_{\setminus s}$ for $X_{V \setminus \{s\}}$.

To each node s in graph G we associate a random variable X_s taking values in a space $\mathcal{X}_s$. For any subset $A \subseteq V$, we use the shorthand $X_A := \{X_s, s \in A\}$. For three disjoint subsets of nodes, A, B, and U, we write $X_A \perp\!\!\!\perp X_B | X_U$ to indicate that the random vector X_A is independent of X_B when conditioning on X_U. We can now define graphical models in terms of the Markov property (e.g., Loh and Wainwright 2012):

Definition 1 *(Global Markov property). If $X_A \perp\!\!\!\perp X_B | X_U$ whenever U is a node cutset that breaks the graph into disjoint subsets A and B, then the random vector $X := (X_1, \dots, X_p)$ is Markov with respect to the graph G.*

Note that the neighborhood set $N(s)$ is always a node cutset for $A = \{s\}$ and $B = V \setminus \{s \cup N(s)\}$.

In the remainder of this paper we focus on exponential family distributions, which are strictly positive distributions. For these distributions the global Markov property is equivalent to the *Markov factorization property* by the Hammersley-Clifford Theorem (Lauritzen 1996). Consider for each clique C in the set of all clique sets $\mathcal{C}$ a clique-compatibility function $\psi_C(X_C)$ that maps configurations $x_C = \{x_s, s \in C\}$ to $\mathbb{R}^+$ such that ψ_C only depends on the variables X_C corresponding to the clique C.

Definition 2 *(Markov factorization property). The distribution of X factorizes according to G if it can be represented as a product of clique functions*

$$P(X) \propto \prod_{C \in \mathcal{C}} \psi_C(X_C). \quad (1)$$

This equivalence implies that if we have distributions that are represented as a product of clique functions, then we can represent the conditional dependence statements in this distribution in a graph G. This is the case for the exponential family distributions we use in the present paper

$$P(X) = \exp\left\{\sum_{C\in\mathcal{C}} \theta_C \phi_C(X_C) - \Phi(\theta)\right\}, \tag{2}$$

where the functions $\phi_C(X_C) = \log \psi_C(X_C)$ are sufficient statistic functions specified by the exponential family member at hand (e.g., Gaussian, Exponential, Poisson, etc.), θ_C are parameters associated with the clique functions and $\Phi(\theta)$ is the log-normalizing constant

$$\Phi(\theta) = \log \int_{\mathcal{X}} \sum_{C\in\mathcal{C}} \theta_C \phi_C(X_C)\nu(dx),$$

where depending on the distribution of X, the measure ν is a counting measure or Lesbesgue measure (for details see Wainwright and Jordan 2008).

The graph G represents a *family* of distributions because its edges do not indicate the strength of the dependency and the nodes can represent different conditional distributions. Hence there is a one to one mapping from the density to the graph, but a one to many mapping from graph to densities.

2.2. Mixed graphical models

In this section we first introduce the general class of mixed graphical models, and then provide the Ising-Gaussian model as a specific example.

General mixed graphical models

In this section, we introduce the class of mixed graphical models (MGMs), which are a special case of the distribution in Equation 2 that allow one to combine an arbitrary set of conditional univariate members of the exponential family in a joint distribution (Yang, Baker, Ravikumar, Allen, and Liu 2014; Chen, Witten, and Shojaie 2015).

Consider a p-dimensional random vector $X = (X_1, \dots, X_p)$ with each variable X_s taking values in a potentially different set $\mathcal{X}_s$, and let $G = (V, E)$ be an undirected graph over p nodes corresponding to the p variables. Now suppose the node-conditional distribution of node X_s conditioned on all other variables $X_{\setminus s}$ is given by an arbitrary univariate exponential family distribution

$$P(X_s|X_{\setminus s}) = \exp\{E_s(X_{\setminus s})\phi_s(X_s) + B_s(X_s) - \Phi(X_{\setminus s})\}, \tag{3}$$

where the functions of the sufficient statistic $\phi_s(\cdot)$ and the base measure $B_s(\cdot)$ are specified by the choice of exponential family and the canonical parameter $E_s(X_{\setminus s})$ is a function of all variables except X_s. Wainwright and Jordan (2008) make these functions explicit for a number of exponential family distributions.

These node-conditional distributions are consistent with a joint distribution over the random vector X as in (1), that is Markov with respect to graph $G = (V, E)$ with the set of cliques $\mathcal{C}_k$ of size at most k, if and only if the canonical parameters $\{E_s(\cdot)\}_{s\in V}$ are a linear combination of products of univariate sufficient statistic functions $\{\phi(X_r)\}_{r\in N(s)}$ of order up to k

$$\theta_s + \sum_{r\in N(s)} \theta_{s,r}\phi_r(X_r) + \dots + \sum_{r_1,\dots,r_{k-1}\in N(s)} \theta_{r_1,\dots,r_{k-1}} \prod_{j=1}^{k-1} \phi_{r_j}(X_{r_j}), \tag{4}$$

where $\theta_{s\cdot} := \{\theta_s, \theta_{s,r}, \dots, \theta_{sr_2 \dots r_k}\}$ is a set of parameters and $N(s)$ is the set of neighbors of node s according to graph G (Yang *et al.* 2014). Factorizing p conditional distributions as in Equation 3 gives the joint distribution

$$P(X) = \exp\left\{ \sum_{s \in V} \theta_s \phi_s(X_s) + \sum_{s \in V} \sum_{r \in N(s)} \theta_{s,r} \phi_s(X_s) \phi_r(X_r) + \right. \\ \left. \dots + \sum_{r_1, \dots, r_k \in \mathcal{C}} \theta_{r_1, \dots, r_k} \prod_{j=1}^{k} \phi_{r_j}(X_{r_j}) + \sum_{s \in V} B_s(X_s) - \Phi(\theta) \right\}, \tag{5}$$

where $\Phi(\theta)$ is the log-normalization constant.

The dimensionality of the parameter vector θ depends both on the type of modeled variables and the order of interactions. If one only models continuous variables with pairwise interactions ($k = 2$), the MGM simplifies to the multivariate Gaussian distribution which is parameterized by a $1 \times p$ vector of intercepts and a $p \times p$ matrix of $\binom{p}{2}$ partial correlations. Including all 3-way interactions would lead to an additional $\binom{p}{3}$ parameters, etc. At the end of Section 2.2, we discuss the dimensionality of the parameter vector in the presence of categorical variables.

Necessary conditions for the mixed density in Equation 5 to be normalizable are discussed in Yang *et al.* (2014). Chen *et al.* (2015) show constraints on the parameter space to ensure normalizability for a number of MGMs with at most pairwise interactions. **mgm** does not allow one to implement the constraints, since the underlying **glmnet** package does not support the specification of these constraints. However, Trip and Van Wieringen (2018) recently proposed an algorithm that allows to estimate pairwise MGMs with these constraints.

Example: The Ising-Gaussian model

We take the Ising-Gaussian model as a specific example of the joint distribution in Equation 5. Consider a random vector $X := (Y, Z)$, where $Y = \{Y_1, \dots, Y_{p_1}\}$ are univariate Gaussian random variables, $Z = \{Z_1, \dots, Z_{p_2}\}$ are univariate Bernoulli random variables and we only consider pairwise interactions between sufficient statistics. For the univariate Gaussian distribution (with known variance σ^2) the sufficient statistic function is $\phi_Y(Y_s) = \frac{Y_s}{\sigma_s}$ and the base measure is $B_Y(Y_s) = -\frac{Y_s^2}{2\sigma_s^2}$. The Bernoulli distribution has the sufficient statistic function $\phi_{Z_s} = Z_s$ and the base measure $B_Z(Z_s) = 0$. From the MGM joint distribution in Equation 5 follows that this mixed density is given by

$$P(Y, Z) \propto \exp\left\{ \sum_{s \in V_Y} \frac{\theta_s}{\sigma_s} Y_s + \sum_{r \in V_Z} \theta_r Z_r + \sum_{(s,r) \in E_Y} \frac{\theta_{s,r}}{\sigma_s \sigma_r} Y_s Y_r + \right. \\ \left. \sum_{(s,r) \in E_Z} \theta_{s,r} Z_s Z_r + \sum_{(s,r) \in E_{YZ}} \frac{\theta_{s,r}}{\sigma_s} Y_s Z_r - \sum_{s \in V_Y} \frac{Y_s^2}{2\sigma_s^2} \right\},$$

where the first two terms are thresholds for Gaussian and Bernoulli variables, the third term represents pairwise interactions between Gaussians, the fourth term represents pairwise interactions between Bernoulli variables, the fifth term represents pairwise interactions between Gaussians and Bernoulli variables, and the last term sums over the base measures for the Gaussians.

When the conditional distribution is a Bernoulli random variable Z_r, it is given by

$$P(Z_r|Z_{\backslash r}, Y) \propto \exp\left\{\theta_r Z_r + \sum_{s\in N(r)_Z} \theta_{s,r} Z_s Z_r + \sum_{s\in N(r)_Y} \frac{\theta_{s,r}}{\sigma_r} Z_r Y_s\right\}. \tag{6}$$

Note that the conditional distribution in Equation 6 has the same form as the distribution of a single variable conditioned on all remaining variables in an Ising model plus one additional term for interactions between Bernoulli and Gaussian random variables.

When the conditional distribution is a Gaussian random variable Y_s, it is given by

$$P(Y_s|Y_{\backslash s}, Z) \propto \exp\left\{\frac{\theta_s}{\sigma_s} Y_s + \sum_{r\in N(s)_Y} \frac{\theta_{s,r}}{\sigma_s \sigma_r} Y_s Y_r + \sum_{r\in N(s)_Z} \frac{\theta_{s,r}}{\sigma_s} Y_s Z_r - \frac{Y_s^2}{2\sigma_s^2}\right\}.$$

Now, let $\sigma = 1$, factor out Y_s and let $\mu_s = \theta_s + \sum_{r\in N(s)_Y} \theta_{s,r} Y_r + \sum_{r\in N(s)_Z} \theta_{s,r} Z_r$. Finally, when taking $\frac{\mu_s^2}{2}$ out of the log normalization constant, we arrive with basic algebra at the well-known form of the univariate Gaussian distribution with unit variance

$$P(Y_s|Y_{\backslash s}, Z) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{(Y_s - \mu_s)^2}{2}\right\}.$$

Relationship between model parameters and edges in graph

For pairwise MGMs (i.e., the size of cliques is at most $k = 2$), a pairwise interaction between two continuous variables s and r is parameterized by a single parameter $\theta_{s,r}$. Now, whether the edge between s and r is present depends on whether $\theta_{s,r}$ is zero or not, that is, $(s,r) \in E \iff \theta_{s,r} \neq 0$. Thus, if only pairwise interactions between continuous variables are modeled, any given edge is a function of a *single* parameter. This implies that a *weighted* graph fully represents the parameterization of interactions in the underlying model (or the full parameterization minus the threshold parameters). Interactions between categorical variables with $m > 2$, however, are specified by more than one parameter. For instance, a pairwise interaction between two categorical variables with m and u categories is parameterized by $R = (m-1) \times (u-1)$ parameters associated with corresponding indicator functions for all R states (e.g., Agresti 2003). A pairwise interaction between a categorical variable with m categories and a continuous variable has $R = 1 \times (m-1)$ parameters associated with $m-1$ indicator functions multiplied with the continuous variable. In this case, $\theta^z_{s,r}$ is a parameter defining the interaction between the nodes s and r indexed by $z \in \{1,\dots,R\}$. In such a situation, an edge is present between s and r if all parameters do *not* have the same value, indicating that not all states have the same probability. In **mgm** we use the parameterization for multinomial regression in **glmnet**, which models the probability of each state of the predicted variable, and codes the first category of the predictor variable as the reference category that is absorbed in the intercept (for details see Friedman *et al.* 2010). This results in $m \times (u-1)$ parameters, where m indicates the number of categories of the predicted variable. In this parameterization, an edge is present if *any* of the parameters in $\theta_{s,r}$ are nonzero, that is, $(s,r) \in E \iff \exists r : |\theta^z_{s,r}| > 0$. Therefore, depending on which variables an edge connects it is defined with respect to one or several parameters.

For general k-order MGMs, an edge between nodes s and r is a function of all cliques of size up to k that include both s and r. Therefore, for instance, it is not clear from the graph

G whether the edge (s, r) is due to a pairwise interaction or from higher order interactions (cliques) that include s and r, or both. The number of parameters associated to each clique discussed above for pairwise interactions extends to k-order interactions. An interaction between k continuous variables is parameterized by a single parameter $\theta_{r_1,\dots,r_{k-1}}$ and an interaction between k categorical variables is parameterized by $(m_1 - 1) \times \cdots \times (m_k - 1)$ parameters, where $m_1, m_2, \dots, m_k$ are the number of categories of each categorical variable.

In this paper we focus mainly on the estimation of pairwise MGMs, where each edge is a function of the parameter(s) of a single pairwise interaction. However, in Section 3.1 we estimate a higher order MGM and visualize the dependency structure in a factor graph. The factor graph representation has the advantage one can still see on which set of cliques a dependency between two nodes depends (Koller and Friedman 2009).

2.3. Estimating mixed graphical models

In this section, we discuss how to estimate the parameters of a joint distribution of the form as in Equation 5 from observations. The graphical model G is then obtained from the parameter estimates as discussed in the previous section.

We know that the joint distribution in Equation 5 can be represented as a factorization of univariate conditional distributions. Thus, if we estimate the p univariate conditional distributions with the parameterization in Equation 4, we obtain the joint distribution. Since all univariate conditional distributions are members of the exponential family, it is possible to estimate the joint distribution in Equation 5 by a series of p regressions in the generalized linear model (GLM) framework (see, e.g., Nelder and Baker 1972). From a graphical models perspective this means that we estimate the neighborhood $N(s)$ of each node $s \in V$ and then combine all neighborhoods to obtain an estimate of the graph G (Meinshausen and Bühlmann 2006).

In order to obtain parameter estimates that are exactly zero (and thereby imply absent edges in the graph) we minimize the log-likelihood $\mathcal{L}(\theta, X)$ together with the ℓ_1-norm of the parameter vector

$$\hat{\theta} = \arg\min_{\theta} \left\{ -\mathcal{L}(\theta, X) + \lambda \|\theta\|_1 \right\}, \tag{7}$$

where $\|\theta\|_1 = \sum_{j=1}^{J} |\theta_j|$, J is the length of the parameter vector θ, and λ is a regularization parameter that determines the relative weights of the log-likelihood and the ℓ_1-norm of the parameter vector. The log-likelihood $\mathcal{L}(\theta, X)$ is defined with respect to the exponential family distribution of the node at hand. In the Gaussian case, minimizing the negative log-likelihood is equivalent to minimizing the squared loss $-\mathcal{L}(\theta, X) = \|X_s - X_{\setminus s}\theta\|_2^2$. In other words, we are performing an ℓ_1-penalized (LASSO) regression in the GLM framework with a link-function appropriate for the node at hand (see, e.g., Nelder and Baker 1972). The ℓ_1-penalty ensures that the model is identified in the high-dimensional setting $p > n$, where we have more parameters than observations (Hastie, Tibshirani, and Wainwright 2015).

The design matrix is defined with respect to the conditional distribution of node s in the k-order MGM. For example, if $k = 2$, the design matrix for the regression on node s contains all other variables or the corresponding indicator functions (for categorical variables). If $k = 3$, the design matrix for the regression on node s contains all other variables or the corresponding indicator functions, plus the products of all pairs of variables in $V_{\setminus s}$, or the $(m - 1) \times (u - 1)$ indicator functions in the case of categorical variables with m and u categories.

To give non-asymptotic guarantees of false and true positive rates for the ℓ_1-regularized regression estimator it is necessary to put a lower bound τ on the size of the parameters in the true model. This assumption is often called the *beta min condition* (see, e.g., Hastie *et al.* 2015). By thresholding estimates at τ, we approximately enforce this condition (see also Loh and Wainwright 2012). For estimating the joint distribution in Equation 5 we show in Haslbeck and Waldorp (2015) that

$$\tau \asymp s_0 \sqrt{\log \frac{p}{n}} \leq s_0 \lambda, \tag{8}$$

where s_0 is the true number of neighbors. If all variables are continuous, the number of neighbors is equal to the number of nonzero parameter estimates $s_0 = \|\theta^*\|_0$, where θ^* is the true parameter vector. In the case of categorical variables, interactions are parameterized by several parameters. In this case the categorical neighbor is present if at least one of the parameters defining the interaction is nonzero. Since the true parameter vector θ^* is unknown, we plug in the estimated parameter vector $\hat{\theta}$ to obtain the *estimated* number of neighbors $\hat{s}_0 = \|\hat{\theta}\|_0$. For interactions involving more than one parameter, we plug in the aggregated parameter (see Algorithm 1). Note that **mgm** allows to switch off this thresholding (see Section 3.1). Of course, switching off the thresholding gives a solution that does not have the guarantees of false and true positive rates.

We determine whether an edge is present or not as described in Section 2.2. In addition, we compute a *weight* from the set of parameters of each interaction. If the interaction only involves continuous variables there is only one parameter and we take its value. If the interaction involves categorical variables, we take the mean of the absolute value of all parameters as the weight of the edge. From the nodewise regressions we obtain k edge-weights for each k-order interaction. For example, for a a pairwise interaction ($k = 2$) between nodes s and r, we obtain one parameter $\theta_{s,r}$ from the regression on s and $\theta_{r,s}$ from the regression on r. To obtain a final conditional dependence graph G we need to combine these into a final weight. This can be done either by using the OR-rule (take arithmetic mean of k parameter estimates) or the AND-rule (take arithmetic mean of k parameter estimates if all parameter estimates are nonzero, otherwise set the parameter to zero). Algorithm 1 summarizes this procedure:

Algorithm 1 *(Estimating mixed graphical models via neighborhood regression.)*

```
1. For each $s \in V$:
    (a) Construct the design matrix defined by $k$, the order of the MGM.
    (b) Solve the lasso problem in Equation 7 with regularization parameter $\lambda$.
    (c) Threshold the estimates at $\tau$.
    (d) Aggregate interactions with several parameters into a single edge-weight.
2. Combine the edge-weights with the AND- or OR-rule.
3. Define $G$ based on the zero/nonzero pattern in the combined parameter vector.
```

The regularization parameter λ can be selected using cross-validation or a model-selection criterion such as the extended Bayesian information criterion (EBIC):

$$\text{EBIC}_\gamma(\hat{\theta}) = -2\mathcal{L}(\hat{\theta}) + \hat{s}_0 \log n + 2\gamma \hat{s}_0 \log p, \tag{9}$$

where $\mathcal{L}$ is the log-likelihood of the conditional density specified by the estimated parameter vector $\hat{\theta}$, $\hat{s}_0$ is the number of nonzero neighbors in the candidate model, and γ is a tuning parameter. Note that if $\gamma = 0$ the EBIC is equal to the BIC (Schwarz 1978). The EBIC has been shown to perform well asymptotically in selecting sparse graphs (Foygel and Drton 2010, 2015) for any value of γ. In practice, the choice of γ will control the trade-off between sensitivity and precision. Foygel and Drton (2010) used values $\gamma \in \{0, 0.25, 0.5, 0.75, 1\}$ and showed that increasing γ from 0 to 0.25 led to a considerable decrease in false positives, without increasing false negatives too much. We therefore adopted $\gamma = 0.25$ as a default value. However, to make an *optimal* choice for γ, it is necessary to take into account the true model, the number of available observations and the cost of false positives and false negatives. While the true model is unknown in real data, a reasonable γ can be selected by running a simulation study roughly reflecting the scenario at hand and choosing the γ with the most desirable performance. To this end we provide flexible sampling functions (see Section 3).

The computational complexity of Algorithm 1 is $\mathcal{O}(p \log(p2^{k-1}))$. Thus the algorithm does not scale well for large k, the order of interactions in the MGM. However, in most situations k will be small, because interactions with a high order are increasingly difficult to interpret and therefore often not of interest.

Note that using a single regularization parameter λ for a model including different edge types may lead to a different penalization for different edge types. This is because edge-parameters are scaled with the sufficient statistic they are associated with and this scaling can differ across exponential family members. While we can bring Gaussian variables on the same scale by subtracting their mean and dividing by their standard deviation, this is not possible for categorical or Poisson random variables. A potential solution would be to introduce a different penalization parameter for each edge type. But this would make the selection of regularization parameters λ considerably more complicated, because now a u-dimensional space of λ values has to be searched, where u is the number of different edge types. This is why we currently do not have a procedure in **mgm** that allows different penalties for different edge types.

The performance of Algorithm 1 depends on the number of variables, the order of interactions, the type of variables, the size of parameters relative to the variance of associated variables, the sparsity of the parameter vector and the structure of the dependency graph. The best way to determine the performance for a given situation is therefore to obtain it with a simulation study. To this end **mgm** provides a flexible function to sample from MGMs such that the performance of Algorithm 1 in a given situation can be evaluated via simulations.

2.4. Mixed autoregressive models

In vector autoregressive (VAR) models, each node s at time point t is modeled as a linear combination of all variables (including s) at a set of earlier time points. The standard VAR model is defined with a Gaussian noise process, such that the model can be split up into p conditional Gaussian distributions (see, e.g., Hamilton 1994; Pfaff 2008a). Instead of a univariate conditional Gaussian distribution, one can also associate other univariate exponential family members with a given node. This leaves us with an almost identical model and estimation problem as discussed in Section 2.3. The only difference is that the canonical parameter of the node-conditional at hand is not a function of parameters associated with interactions of variables at the *same time point*, but a function of parameters associated with variables at *previous time points*. To distinguish this VAR model over mixed variables from the VAR model that is typically defined with only Gaussian variables, we call this model *mixed autoregressive (mVAR)* model.

The mVAR model can be estimated by estimating the parameters of the conditional probability of each variable s as a function of all variables (including itself) at a set of specified previous time points, denoted by L. For example, $L = \{1, 2, 3\}$ specifies a VAR model with lags 1, 2 and 3. We introduce a time index as a superscript t for all variables since we are now dealing with time-ordered observations. We then define the canonical parameter $E_s^t(X)$ of the conditional distribution $P(X_s^t | X^{t-1}, X^{t-2}, X^{t-3})$ at time t

$$E_s^t(X) = \theta_s + \sum_{j \in L} \sum_{r \in N(s)} \theta_{s,r}^{t-j} \phi_r(X_r^{t-j}). \tag{10}$$

We only included pairwise interactions because **mgm** does not implement higher order interactions for mVAR models. The canonical parameter function in Equation 10 defines the log-likelihood $\mathcal{L}(\theta, X)$ in Equation 7 and we can therefore use Algorithm 1 with two modifications: First, we define the design matrix as a function of the included lags L instead of the maximal order of the interactions, which we here fix to $k = 2$ (only pairwise interactions). Second, we do not apply an AND/OR rule, because the cross-lagged effect of X_s^{t-1} on X_r^t is a different effect than the cross-lagged effect of X_r^{t-1} on X_s^t and thus no parameter is estimated twice. Here we state the modified algorithm explicitly:

Algorithm 2 *(Estimating mixed VAR models via nodewise regression.)*

```
1. For each $s \in V$:
    (a) Construct the design matrix defined by $L$, the set of included lags.
    (b) Solve the lasso problem in Equation 7 with regularization parameter $\lambda$.
    (c) Threshold the estimates at $\tau$.
2. Define the directed graphs $D_j$ based on the zero/nonzero pattern in the combined parameter vector for each lag $j \in L$.
```

The computational complexity of Algorithm 2 is $\mathcal{O}(p \log(p|L|))$. Similarly to Algorithm 1, the regularization parameter λ can be selected using cross-validation or an information criterion such as the EBIC.

Note that the directed graphs in the $p \times p \times |L|$ array D are not encoding conditional independence statements as the graph G for MGMs. But they are a useful summary of the parameters of the mixed VAR model, especially because it allows a visualization as a series of directed networks (see Section 3.1 for illustrations).

Note that the performance of Algorithm 2 depends on the number of variables and the number of lags, the type of variables, the size of parameters relative to the variance of associated variables, the sparsity of the parameter vector and the structure of the dependency graph. **mgm** offers a flexible function to sample from mixed VAR models such that the performance of Algorithm 2 in a given situation can be evaluated via simulation studies. Haslbeck, Bringmann, and Waldorp (2019) report the performance of Algorithm 2 in recovering VAR models with Gaussian noise process in a variety of situations.

2.5. Estimating time-varying models

For both MGMs (Section 2.2) and mixed VAR models (Section 2.4) for time series data, we assumed so far that the models are stationary. This means that the observations at each time t point are generated from the same distribution parameterized by θ. In time-varying models we relax this assumption, such that the parameters θ^t can be different at each time point $t \in \mathcal{T}_n = \{\frac{1}{n}, \frac{2}{n}, \dots, 1\}$, where n is the number time points in the time series. Note that we use n to denote the number of observations both for cross-sectional data (observations are measurements of different systems from some population) and time series data (repeated measurements of the same system).

Since one cannot estimate a model from a single time point, we have to make assumptions about how the parameters of the true model vary as a function of time. These assumptions are usually assumptions about *local stationarity* (e.g., Zhou, Lafferty, and Wasserman 2010) and come in one of two flavors: We either assume that there exists a partition $\mathcal{B}$ of $\mathcal{T}_n$ in which time points are consecutive and in each of the subsets $B \in \mathcal{B}$ the model is stationary, that is, $\forall i, j \in B : \theta^i = \theta^j$. These piecewise constant time-varying models can be estimated with a fused lasso penalty, which puts an additional penalty on parameter changes from one time point to the subsequent time point (see, e.g., Monti *et al.* 2014; Kolar and Xing 2012; Gibberd and Nelson 2016, 2017).

The other type of local stationarity, which we focus on in this paper, requires that the model θ^t is a smooth function of time. In this case we can combine observations close in time for estimation, because we know that their generating models are similar. This idea is implemented by fitting local models $\hat{\theta}^{t^e}$ across the time series, which only give high weight to data points close to the given estimation point t^e. The weight function is usually non-negative and symmetric over t^e (see, e.g., Song, Kolar, and Xing 2009; Zhou *et al.* 2010; Kolar, Song, Ahmed, and Xing 2010; Kolar and Xing 2013; Tao, Huang, Wang, Xi, and Li 2016; Chen and He 2018). The full time-varying model is then the set of all local estimates $\{\theta^{e_1}, \theta^{e_2}, \dots, \theta^{|\mathcal{E}|}\}$ at estimation points $\mathcal{E} = \{t_1^e, t_2^e, \dots, t^{|\mathcal{E}|}\}$, where the entries in $\mathcal{E}$ are usually equally spaced across the time series and the number of estimation points $|\mathcal{E}|$ is chosen depending on how fine-grained one would like to describe θ^t as a function of time t.

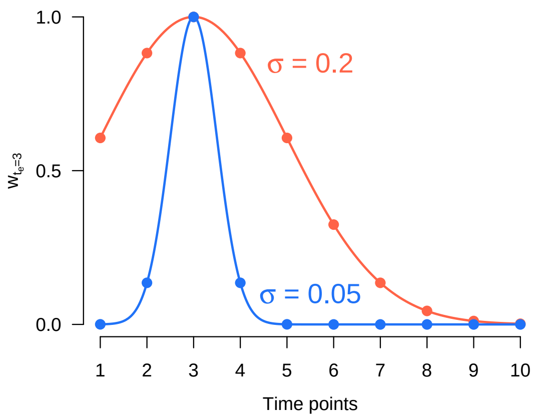
Stating the above formally, we estimate the model θ^{t^e} at time point t^e by minimizing a weighted version of the loss function in Equation 7 in Section 3.1:

$$\hat{\theta}^{t^e} = \arg\min_{\theta} \left\{ -\frac{1}{\sum_{t=1}^{n} w_t^{t^e}} \sum_{t=1}^{n} w_t^{t^e} \mathcal{L}(\theta, X^t) + \lambda \|\theta\|_1 \right\}, \tag{11}$$

where $w_t^{t^e}$ is a function of t defined by a kernel centered over t^e. Specifically, we define the weight function $w_t^{t^e}$ to be a Gaussian kernel, normalized such that the largest weight is equal to one (Zhou *et al.* 2010):

$$w_t^{t^e} = \frac{Z_t}{\max_{(t \in \mathcal{T}_n)} \{\cup_t Z_t\}}, \quad \text{where} \quad Z_t = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{(t - t^e)^2}{2\sigma^2} \right\}. \tag{12}$$

This particular scaling of the weight function has the convenient property that the sum of all weights $n_{\sigma,t^e} = \sum_{t=1}^{n} w_t^{t^e}$ (or the area under the curve) used at a given estimation point t^e indicates the amount of data used for estimation at t^e relative to the full time series (the full rectangle). Note that we indexed n_{σ,t^e} also with the estimation point t^e, because less



$$\begin{array}{c|cccccc}
\text{Time} & X_{t,1} & X_{t-1,1} & \dots & X_{t,p} & w^{t_e=3} & w^{t_e^*=3} \\
\hline
1 & 0.03 & -0.97 & \dots & -0.08 & 0.61 & 0.00 \\
2 & 1.15 & -1.07 & \dots & -0.56 & 0.88 & 0.14 \\
3 & 0.11 & 0.63 & \dots & 1.09 & 1.00 & 1.00 \\
4 & -1.08 & 0.13 & \dots & 1.88 & 0.88 & 0.14 \\
5 & -0.93 & 1.00 & \dots & -0.29 & 0.61 & 0.00 \\
6 & -1.08 & 0.17 & \dots & -1.36 & 0.32 & 0.00 \\
7 & 0.27 & -1.72 & \dots & -1.13 & 0.14 & 0.00 \\
8 & 0.03 & -1.26 & \dots & -0.97 & 0.04 & 0.00 \\
9 & -1.29 & -1.05 & \dots & -0.10 & 0.01 & 0.00 \\
10 & -0.07 & -0.04 & 1.05 & -0.12 & 0.00 & 0.00
\end{array}$$

Figure 1: Illustration of two kernel weighting functions with different bandwidth parameter defined for the estimation point $t^e = 3$. Left panel: weights as a function of time; right panel: equivalent representation of the weights across time, combined with the time series data.

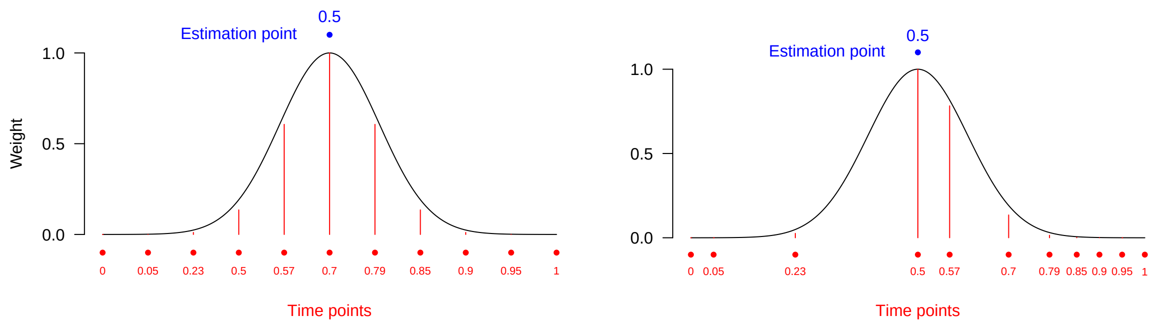
data is used at the beginning and the end of the time series, where the weighting function is truncated (see left panel Figure 1).

The example in Figure 1 illustrates this estimation procedure. Here we have a time series of $n = 10$ measurements of p continuous variables, and we would like to estimate the model at time point $t^e = 3$. To this end we first define a kernel function w_t^{te} as in Equation 12. The bandwidth σ of the kernel, which is here equal to the standard deviation of the Gaussian distribution, indicates how many observations close in time we combine to estimate the node at estimation point t^e.

Figure 1 displays the kernel function w_t^{te} for two different choices of bandwidth, $\sigma = 0.05$ and $\sigma = 0.2$. The kernel function with $\sigma = 0.05$ gives only time points very close to $t^e = 3$ a nonzero weight, while other time points get a weight close to zero and have therefore almost no influence on the parameter estimated at $t^e = 3$. In contrast, the kernel function with $\sigma = 0.2$ distributes weights more evenly, which implies that also time-points quite distant from $t^e = 3$ influence the parameter estimates at $t^e = 3$. The values of both weighting functions at the measured time points are also illustrated together with the data matrix in the right panel of Figure 1.

The choice of bandwidth involves a trade-off between the sensitivity to time-varying parameters and the stability of the estimates: If we combine only a few observations close in time (small bandwidth σ) the algorithm can detect parameter-variation at small time scales, however, because we use little data, the estimates will be unstable. If we combine many observations around the estimation point (large σ), parameter-variation at small time scales will be lost due to aggregation, however, the estimates will be relatively stable. Note that if we keep increasing the bandwidth σ, the weights on $[0, 1]$ will converge to a uniform distribution and give the same estimates as the stationary version of the model, thereby becoming relatively stable, but losing all sensitivity to detect changes in parameters over time.

The ideal bandwidth σ^* results in the estimated parameter vector $\hat{\theta}^t$ which minimizes the distance to the true time-varying model $\hat{\theta}^{t*}$ as a function of σ. We can estimate the ideal



(a) Assumed equal spacing (b) Include true time points

Figure 2: Left panel: the weighting function is computed by assuming that the true time points are equally spaced. Since the true time points are not equally spaced, this creates a mismatch between the time scale of estimation points and the true time scale. Right panel: the true time interval is used to compute the weights and hence the two scalings match.

bandwidth σ^* using a time-stratified cross-validation scheme, where one searches a specified σ-sequence and selects the σ which minimizes the mean (across folds and variables) out-of-sample prediction error (see Section 3.3 for a description of the time-stratified cross-validation scheme).

So far we assumed that the measurements in the time-series are taken at equal time intervals. But this need not be the case, because measurements can be missing randomly or by design. Simply treating the time points as equally distributed leads to an incorrect estimate of the time-varying model. Figure 2 illustrates this issue.

Here we have a time series with $n = 10$ time points, measured at irregular time intervals. In Figure 2(a) we distribute these time points evenly across the time interval, which results in that the assigned time points in the normalized time interval $[0, 1]$ do not correspond to the true time points (values in red). Now if we estimate the time-varying model at time point $t^e = 0.5$, we see that the true time point 0.7 gets the highest weight. Thus, the model at $t^e = 0.5$ is more strongly influenced by the observations at the true time point 0.7 than by the observations at time point 0.5. Clearly, this is undesirable.

In Figure 2(b) we avoid this problem by using the true time points in order to define the weighting function $w_t^{t^e}$. We again estimate the model at $t^e = 0.5$ and see that the time scale of the estimation point is now aligned with the true time scale. This results in a different amount of data used for estimation n_{σ,t^e}, depending on how many measurements are available around a given time point. If there is less data available, the algorithm becomes more conservative, since we plug in n_{σ,t^e} for n in the τ threshold in Equation 8. In the extreme case where there is no data close to an estimation point, n_{σ,t^e} will be extremely small, which implies that the algorithm sets all estimates to zero. This makes sense, because if there is no data available close to a given estimation point t^e, we cannot expect to obtain reliable estimates at t^e.

Note that the only difference between the stationary models and the time-varying models is that we introduce a weight for each time point in the cost function in Equation 11 and repeatedly estimate the model at different estimation points. Therefore we can easily adapt the estimation algorithms for the stationary MGM (Algorithm 1) and mixed VAR model

(Algorithm 2) to their time-varying versions. We first state the algorithm for time-varying MGMs.

Algorithm 3 *(Estimating time-varying MGMs via kernel-smoothed neighborhood regression.)*

```
1. For each estimation point $t^e \in \mathcal{E}$:
   (a) For each variable $s \in V$:
       i. Construct the design matrix defined by $k$, the order of the MGM.
       ii. Solve the weighted lasso problem in Equation 11 with regularization parameter
           $\lambda$ and the weighting function $w^{t^e}$ defined by $t^e$ and bandwidth $\sigma$.
       iii. Threshold the estimates at $\tau_{\sigma,t^e}$.
   (b) Combine the parameter estimates with the AND- or OR-rule.
   (c) Define $G^e$ based on the zero/nonzero pattern in the combined parameter vector $\theta^e$.
```

Thus we obtain a parameter vector θ^{t^e} of the MGM in Equation 5 and a graph G^{t^e} defined by θ^{t^e}, for each estimation point $t^e \in \mathcal{E}$. From Algorithm 1 follows that Algorithm 3 has a computational complexity of $\mathcal{O}(|\mathcal{E}|p \log(p2^{k-1}))$.

Similarly, we can adapt Algorithm 2 for the estimation of time-varying mixed VAR models.

Algorithm 4 *(Estimating time-varying mixed VAR models via kernel-smoothed neighborhood regression.)*

```
1. For each estimation point $t^e \in \mathcal{E}$:
   (a) For each variable $s \in V$:
       i. Construct the design matrix defined by the $L$, the set of included lags.
       ii. Solve the weighted lasso problem in Equation 11 with regularization parameter
           $\lambda$ and the weighting function $w^{t^e}$ defined by $t^e$ and bandwidth $\sigma$.
       iii. Threshold the estimates at $\tau_{\sigma,t^e}$.
   (b) Define the directed graphs $D^e_j$ based on the zero/nonzero pattern in the parameter
       vector $\theta^e$ for each lag $j \in L$.
```

Here we obtain a parameter vector θ^{t^e} of the mVAR model and a directed graph $D^{t^e}_j$ for each lag, defined by θ^{t^e}, for each estimation point $t^e \in \mathcal{E}$. From Algorithm 2 follows that Algorithm 4 has a computational complexity of $\mathcal{O}(|\mathcal{E}|p \log(p|L|))$. Haslbeck *et al.* (2019) report the performance of Algorithm 4 in recovering time-varying VAR models with Gaussian noise process for a variety of situations.

Fitting a time-varying model with the above method requires to specify an appropriate bandwidth parameter σ. In Section 3.3, we describe a time-stratified cross-validation scheme to select σ in a data-driven way. The EBIC is not suitable to select σ. The reason is that threshold (intercept) parameters are neither included in the ℓ_1-penalty, nor in the EBIC. This results in the EBIC selecting always the model with the smallest specified bandwidth, which includes no interaction parameters, but achieves an extremely good fit through highly local (time-varying) thresholds (intercepts). This problem is avoided when using a cross-validation scheme, where fitting local means leads to high out-of-fold prediction error.

Note that the performance of Algorithms 3 and 4 depends on the number of variables, the type of variables, the size of parameters relative to their variance, the sparsity of the parameter vectors, the structure of the dependency graph and how non-linear the parameters vary as a function of time. The best way to obtain the performance of Algorithms 3 and 4 is to set up a suitable simulation study. To this end **mgm** offers flexible functions to sample from time-varying MGMs and time-varying mVAR models.

3. Usage and examples

The **mgm** package can be installed from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=mgm`:

```
R> install.packages("mgm")
R> library("mgm")
```

In the following four sections, we show for each of the four model types how to:

1. Sample observations from a specified model.
2. Estimate the model from data.
3. Make predictions from an estimated model.
4. Visualize the estimated model.
5. Assess the stability of estimates.

The sampling functions are included to enable the user to determine the performance of the estimation algorithm in a specific situation via simulations. All data sets used are loaded automatically with the **mgm** package. All analyses in the paper are fully reproducible, and the necessary code is either shown in the paper or can be found in the online supplementary material or the Github repository `https://github.com/jmbh/mgmDocumentation`. For all code examples we use **mgm** version 1.2-9 and R version 3.6.

3.1. Stationary mixed graphical models

In this section we first use a simulated data set to show how to estimate a pairwise MGM, compute predictions from it, visualize it and assess the stability of its parameters. Then we fit a pairwise MGM to a larger empirical data set related to autism spectrum disorder (ASD). Finally, we give an example of a higher-order MGM by showing how to estimate a $k = 3$ MGM to a data set consisting of symptoms of post-traumatic stress disorder (PTSD).

Estimating mixed graphical models

In this section we show how to use the function `mgm()` to estimate a pairwise MGM to a data set with $n = 500$ observations of two continuous, and two categorical variables with $m = 2$ and $u = 4$ categories, respectively. The true model includes the pairwise interactions 1–4, 2–3 and 1–2. For the exact parameterization of the true model and for a description of how to sample from this MGM using the `mgmsampler()` function see the section on sampling below.

Next to the data, we specify the type of each variable (`"g"` for Gaussian, `"p"` for Poisson, `"c"` for categorical) and the number of levels of each variable (1 for continuous variables by convention). Here we use the example data `mgm_data` which is automatically loaded with **mgm**. Next, we indicate the order of the graphical model: We choose $k = 2$, which corresponds to a pairwise MGM (containing at most 2-way interactions). If we specified $k = 3$, we would fit an MGM including all 2-way and all 3-way interactions, $k = 4$ would include all 2-way, 3-way and 4-way interactions, etc. After that, we specify how we select the penalization parameter λ in Algorithm 1. The two available options are the EBIC or cross-validation. Here we choose cross-validation with 10 folds. If not otherwise specified via the argument `lambdaSeq`, the considered λ-sequence is determined as in the **glmnet** package: a sequence is defined from $\lambda_{\max}$, the smallest (data derived) value for which all coefficients are zero, and $\lambda_{\min}$, a fraction of $\lambda_{\max}$, which is 0.01 in the high-dimensional setting ($n < p$) and 0.0001 if $n > p$. Finally, we indicate that estimates across neighborhood regressions should be combined with the AND-rule. Since we use cross-validation, we set a random seed outside the function to ensure that the analysis is reproducible.

```
R> set.seed(1)
R> fit_mgm <- mgm(data = mgm_data$data, type = c("g", "c", "c", "g"),
+    levels = c(1, 2, 4, 1), k = 2, lambdaSel = "CV", lambdaFolds = 10,
+    ruleReg = "AND")
```

`mgm()` returns a list with the following entries: `fit_mgm$call` returns the call of the function; `fit_mgm$pairwise` contains the weighted adjacency matrix and the signs (if defined) of the parameters in the weighted adjacency matrix; `fit_mgm$interactions` contains a list that shows all recovered interactions (cliques) and a list that returns the parameters associated with all cliques; `fit_mgm$intercepts` stores all estimated thresholds/intercepts and `fit_mgm$nodemodels` is a list with the p **glmnet** objects from which all above results are computed. We inspect the weighted adjacency matrix stored in `fit_mgm$pairwise$wadj`:

```
R> round(fit_mgm$pairwise$wadj, 2)

     [,1] [,2] [,3] [,4]
[1,] 0.00 0.53 0.00 0.46
[2,] 0.53 0.00 0.09 0.00
[3,] 0.00 0.09 0.00 0.00
[4,] 0.46 0.00 0.00 0.00
```

and see that we correctly recovered the pairwise dependencies 1–4, 2–3 and 1–2. The list entry `fit_mgm$pairwise$signs` indicates the sign for each interaction, if a sign is defined. By default, a sign is only defined for interactions between non-categorical variables (Gaussian, Poisson). Interactions involving categorical variables with $m > 2$ categories are defined by more than one parameter and hence no sign can be defined. The function `showInteraction()` provides an alternative way to inspect a given interaction. For instance, one can obtain the details about the interaction 1–4 like this:

```
R> showInteraction(object = fit_mgm, int = c(1, 4))
```

```
Interaction: 1-4
Weight:  0.4586544
Sign:  1  (Positive)
```

We use the **glmnet** package to fit the regularized nodewise regressions, which directly models the probabilities of categorical variables instead of the ratio relative to a reference category. This is possible, because the regularization ensures that this model is identified (for details see Friedman *et al.* 2010). This means that an interaction between two categorical variables $X_1 \in \{1, \dots, m\}$ and $X_2 \in \{1, \dots, u\}$ has $m \times (u-1)$ parameters in the regression on X_1 and $u \times (m-1)$ parameters in the regression on X_2. In addition, all estimation functions in **mgm** also allow an overparameterization (specified via the argument `overparameterize = TRUE`), where an indicator function is defined for *each* state of the categorical predictor variable. In the previous example of a pairwise interaction, this leads to $m \times u$ parameters specifying the interaction between X_1 and X_2. The overparameterization is useful when one is interested in parameters associated with indicator functions that are otherwise absorbed by the threshold (intercept) parameters (also called reference category). We give an example when estimating a $k = 3$ order MGM at the end of this section.

If the argument `binarySign` is set to `TRUE`, all binary variables have to be coded as $\{0, 1\}$ and a sign is defined in the following way: For an interaction between two binary variables $X_1, X_2 \in \{0, 1\}$, if the parameter associated with the indicator function $\mathbb{I}_{X_2=1}$ in the equation modeling $P(X_1 = 1)$ has a positive sign (which implies that the parameter associated with $\mathbb{I}_{X_2=1}$ in the equation modeling $P(X_1 = 0)$ has a negative sign, see Friedman *et al.* 2010), then we assign a positive sign to the binary-binary interaction. For an interaction between a binary variable X_1 and a *continuous* variable X_2 we take the sign of the parameter associated with X_2 in the equation modeling $P(X_1 = 1)$. In addition, it is possible to specify a weight for each observation via the argument `weights` to perform weighted regression.

In the example above we used an ℓ_1-penalized GLM to estimate the MGM, which implies that we assume that the true MGM is sparse. However, a different penalty may be appropriate in some situations. Via the argument `alphaSeq` one can specify any convex combination of the ℓ_1- and ℓ_2-penalty (the elastic net penalty, see Zou and Hastie 2005). `alphaSeq = 1` corresponds to the ℓ_1-penalty (default) and `alphaSeq = 0` to the ℓ_2-penalty. If a sequence of values is provided to `alphaSeq`, the function will select the best α value based on the EBIC or cross-validation, specified via the argument `alphaSel`.

Making predictions from mixed graphical models

We now use the `predict()` function to compute predictions and nodewise errors from the model estimated in the previous section. This function takes the model object and data of the same format as the data used for estimation as input. It also allows to specify which error functions should be used to compute nodewise prediction errors. The error functions $F(\hat{y}, y)$ for continuous and categorical variables are specified via the `errorCon` and `errorCat` arguments, respectively. Here we specified the root mean squared error (`"RMSE"`) and the proportion of explained variance (`"R2"`) as error functions for the continuous variables, and the proportion of correct classification (or accuracy, `"CC"`) and the normalized proportion of correct classification (`"nCC"`) for categorical variables. `"nCC"` indicates the increase in accuracy beyond the intercept model, divided by the maximal possible increase, and thereby captures how well a node is predicted by other nodes beyond the intercept model. Specifically, let $\mathcal{A} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(y_i = \hat{y}_i)$ be the proportion of correct classifications, and let $p_0, p_1, \ldots p_m$ be the marginal probabilities of the categories, where $\mathbb{I}$ is the indicator function for the event $R_i = \hat{R}_i$. In the binary case these are p_0 and $p_1 = 1 - p_0$. We then define the normalized accuracy as

$$\mathcal{A}_{\text{norm}} = \frac{\mathcal{A} - \max\{p_0, p_1, \ldots, p_m\}}{1 - \max\{p_0, p_1, \ldots, p_m\}}.$$

For details see Haslbeck and Waldorp (2018). If one is not interested in computing nodewise prediction errors, the arguments `errorCon` and `errorCat` can be simply ignored.

We provide the object with fit **mgm** and the data as input arguments and a choice of prediction error measures to the `predict()` function:

```
R> pred_mgm <- predict(object = fit_mgm, data = mgm_data$data,
+    errorCon = c("RMSE", "R2"), errorCat = c("CC", "nCC"))
```

The output object `pred_mgm` is a list that contains the function call, the predicted values, the predicted probabilities of each category in case the model includes categorical variables, and a table with nodewise prediction errors. Here we print the nodewise error table:

```
R> pred_mgm$errors
```

```
     Variable Error.RMSE Error.R2 Error.CC Error.nCC
[1,]        1      0.781    0.389       NA        NA
[2,]        2         NA       NA    0.842     0.225
[3,]        3         NA       NA    0.342     0.000
[4,]        4      0.855    0.268       NA        NA
```

The RMSE and R^2 are shown for the two continuous variables, the accuracy and normalized accuracy are shown for the two categorical variables. It is possible to provide an arbitrary number of customary error functions for both continuous and categorical variables to `predict()`, for details see `?predict.mgm`.

In this example we used the same data for estimation and prediction, which means that we computed *within sample* prediction errors. In order to evaluate how well the model generalizes out of sample, the predictions have to be made on a fresh test data set. This can be done by providing new data of the same format to the `predict()` function.

Visualizing mixed graphical models

We visualize interaction parameters of the pairwise model together with the nodewise errors using the **qgraph** package (Epskamp *et al.* 2012). To this end we first install and load the **qgraph** package and compute a vector containing the nodewise errors we would like to display:

```
R> install.packages("qgraph")
R> library("qgraph")
R> errors <- c(pred_mgm$errors[1, 3], pred_mgm$errors[2:3, 4],
+    pred_mgm$errors[4, 3])
```

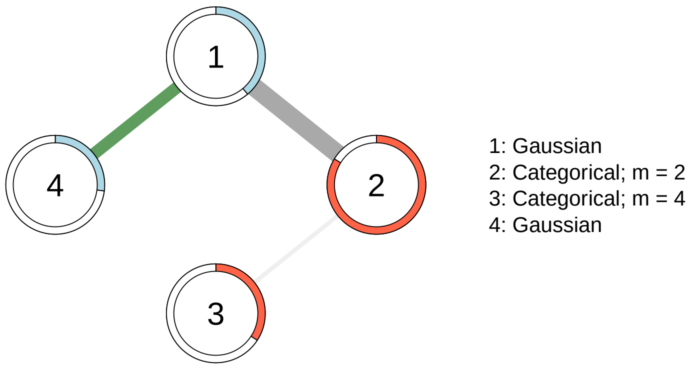


Figure 3: Visualization of the edge-parameters and nodewise errors of the estimated MGM. Green edges indicate positive relationships. Grey edges indicate pairwise interactions for which no sign is defined (interactions involving categorical variables). The width of the edges is proportional to the absolute value of the associated edge-parameter.

Here we decided to display the proportion of explained variance for the continuous variables and the accuracy for the categorical variables. In order to plot the model, one provides the weighted adjacency matrix and the errors to the function `qgraph()`. We also provide a matrix of edge colors that specify the sign of each interaction (green = positive, red = negative, grey = undefined) that is stored in the object fit with **mgm**. Finally we provide colors for the different error measures and variable names for the legend.

```
R> qgraph(fit_mgm$pairwise$wadj, edge.color = fit_mgm$pairwise$edgecolor,
+    pie = errors, pieColor = c("lightblue", "tomato", "tomato",
+    "lightblue"), nodeNames = c("Gaussian", "Categorical; m = 2",
+    "Categorical; m = 4", "Gaussian"), legend = TRUE)
```

Figure 3 shows the resulting visualization. The green edge between variable 1 and variable 2 indicates a positive linear relationship between the two Gaussian variables and the two grey edges indicate relationships between categorical variables, for which no sign is defined. The exact nature of these interactions can be found by inspecting them using the output object of the `showInteraction()` function. The width of the edges is proportional to the size of the corresponding edge-parameter. The blue rings indicate the proportion of variance explained by neighboring nodes for the Gaussian variables, and the red rings indicate the accuracy of the categorical nodes.

Bootstrap sampling distributions

Obtaining the sampling distributions for parameter estimates can be useful if one is interested in the stability of estimates (Hastie *et al.* 2015). The function `resample()` obtains empirical sampling distributions with the nonparametric bootstrap (Efron 1992; Efron and Tibshirani 1994). Its input is the **mgm** model object `fit_mgm` (the output of the function `mgm()`, see above), the data, and the desired number of bootstrap samples B via the argument `nB`. The argument `quantiles` specifies lower/upper quantiles of the sampling distributions, which are

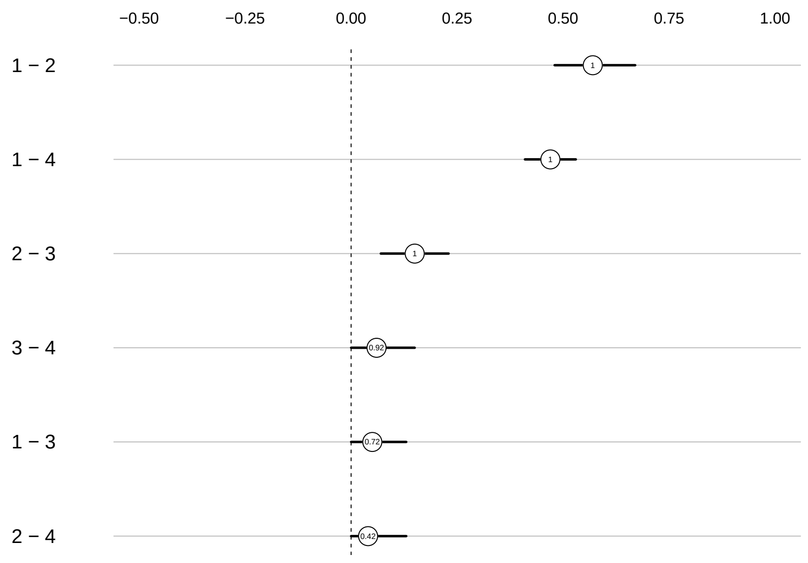


Figure 4: Summaries of bootstrapped sampling distributions separately for the weight of each edge. The value indicates the proportion of nonzero estimates across the B bootstrap samples and is plotted at the arithmetic mean of the sampling distribution. The black horizontal lines indicate the 0.05 and 0.95 quantiles of the bootstrapped sampling distribution.

added to the output. Here we choose `quantiles = c(0.05, 0.95)`. Finally, we set a random seed to make the analysis reproducible.

```
R> set.seed(1)
R> res_obj <- resample(object = fit_mgm, data = mgm_data$data, nB = 50,
+    quantiles = c(0.05, 0.95))
```

The bootstrapped sampling distributions of the edge weights can be found in a $B \times p \times p$ array stored in the list entry `res_obj$bootParameters`. For example, the vector of length B with the bootstrapped sampling distribution of the weight of the edge 3–4 is stored in the entry `res_obj$bootParameters[, 3, 4]`. The output object `res_obj` also contains the specified lower/upper quantiles of each sampling distribution, the function call, the B estimated models and the running time for each bootstrap sample. The function `plotRes()` provides a plot that summarizes the bootstrapped sampling distributions. For each edge weight, it displays the proportion of nonzero estimates across all B models, printed at the arithmetic mean of the sampling distribution. In addition, it displays specified lower/upper quantiles. Here we choose the 5% and 95% quantiles by setting `quantiles = c(0.05, 0.95)`.

```
R> plotRes(object = res_obj, quantiles = c(0.05, 0.95))
```

The resulting plot is displayed in Figure 4. It shows that the sampling distributions for the edges 1–2 and 1–4 are located far from zero, have a small standard deviation and 100% of the B estimates were nonzero. For the edges 3–4, 2–3 and 1–3 that are absent in the true graph, the sampling distribution is close to zero and the proportion of estimated nonzero effects is much smaller.

While bootstrapped sampling distributions are useful to determine the stability of estimates, they are not suited for performing hypothesis tests, for instance, against the null hypothesis that the population parameter is equal to zero. The reason is that the sampling distributions of parameters obtained with ℓ_1-regularized regression have a zero mass around zero (Bühlmann, Kalisch, and Meier 2014). The R package **bootnet** (Epskamp, Borsboom, and Fried 2018) also implements a bootstrap scheme for **mgm** objects and provides similar plotting options.

Sampling from mixed graphical models

Here we illustrate how to use the function `mgmsampler()` to create the data set `mgm_data` that was used for estimation above. These data were created by specifying an MGM consisting of two continuous Gaussian nodes (`"g"`), and two categorical nodes (`"c"`) with $m = 2$ and $u = 4$ categories, and three pairwise interactions between these four variables. A third option in `mgmsampler()` are Poisson nodes (`"p"`). Note that we use the overparameterized representation of interactions between categorical variables to specify the model, which means that the pairwise interaction between the categorical variables has $m \times u$ parameters. We begin by specifying the type and number of categories for each node. By convention, for continuous variables we set the number of categories to 1.

```
R> type <- c("g", "c", "c", "g")
R> level <- c(1, 2, 4, 1)
```

Next, we specify a list containing the thresholds for each variable:

```
R> thresholds <- list()
R> thresholds[[1]] <- 0
R> thresholds[[2]] <- rep(0, level[2])
R> thresholds[[3]] <- rep(0, level[3])
R> thresholds[[4]] <- 0
```

We specify a zero threshold (intercept) for the two Gaussian nodes, and for each of the categories of both categorical variables. Thresholds correspond to the first summation in the joint MGM density in Equation 5. Next, we specify a vector containing the standard deviations for the Gaussian variables:

```
R> sds <- rep(1, 4)
```

The entries in `sds` corresponding to non-Gaussian nodes (here 2 and 3) are ignored. Finally, we specify three pairwise interactions between the variables 1–2, 2–3 and 1–4 in two steps: First, we create a matrix, in which each row indicates one pairwise interaction:

```
R> factors <- list()
R> factors[[1]] <- matrix(c(1, 4, 2, 3, 1, 2), ncol = 2, byrow = TRUE)
```

We assign the matrix to the first list entry `factors[[1]]`, which contains pairwise interactions. The second list entry `factors[[2]]` contains a $q \times 3$ matrix of q 3-way interactions, the third entry contains a $w \times 4$ matrix of w 3-way interactions, etc. Since we only specify pairwise interactions in this example, we only use the first entry. A description and examples of how to specify higher order interactions are given in the help file `?mgmsampler`. In a second step, we specify the parameters of the three interactions:

```
R> interactions <- list()
R> interactions[[1]] <- vector("list", length = 3)
R> interactions[[1]][[1]] <- array(0.5, dim = c(level[1], level[4]))
R> int_2 <- matrix(0, nrow = level[2], ncol = level[3])
R> int_2[1, 1:2] <- 1
R> interactions[[1]][[2]] <- int_2
R> int_1 <- matrix(0, nrow = level[1], ncol = level[2])
R> int_1[1, 1] <- 1
R> interactions[[1]][[3]] <- int_1
```

The interaction between the continuous variables 1–2 is parameterized by one parameter with value 0.5. The interaction between the two categorical variables is specified by a 2×4 parameter matrix. We give the entries $(1, 1)$ and $(1, 2)$ a value of 1, which means that these two states have a higher probability than the remaining states, which are associated with a value of 0. Finally, we specify the interaction between the continuous Gaussian node 1 and the binary node 2, which has two parameters associated with the two indicator functions for the binary variable multiplied with the continuous variable. Now we provide these arguments to the `mgmsampler()` function, together with $n = 500$, which samples 500 observations from the model:

```
R> set.seed(1)
R> mgm_data <- mgmsampler(factors = factors, interactions = interactions,
+    thresholds = thresholds, sds = sds, type = type, level = level, N = 500)
```

The function returns a list containing the function call in `mgm_data$call` and the data in `mgm_data$data`. For more details on how to specify k-order MGMs we refer the reader to the help file `?mgmsampler`.

Application: Autism and well-being

Here we show how to estimate an MGM on a real data set consisting of responses of 3521 individuals from the Netherlands, who were diagnosed with autism spectrum disorder (ASD), to 28 questions on demographics, psychological aspects, conditions of the social environment and medical measurements (for details see Begeer, Wierda, and Venderbosch 2013; Deserno, Borsboom, Begeer, and Geurts 2017). The data set is included in the **mgm** package and loaded automatically. It includes continuous variables, count variables and categorical variables (see `autism_data_large$type`), and the latter have between 2 and 5 categories (see `autism_data_large$level`).

We choose a pairwise model ($k = 2$) and select the regularization parameters λ using the EBIC with a hyper-parameter $\gamma = 0.25$:

```
R> fit_ADS <- mgm(data = autism_data_large$data,
+    type = autism_data_large$type, level = autism_data_large$level,
+    k = 2, lambdaSel = "EBIC", lambdaGam = 0.25)
```

The 28×28 weighted adjacency matrix is too large to be displayed here. Instead, we directly visualize it using the **qgraph** package. In addition to the weighted adjacency matrix and the matrix containing edge colors that indicate the signs of edge parameters, we also provide a grouping of the variables into the categories *Demographics*, *Psychological*, *Social environment* and *Medical* measurements as well as colors for the grouping, both of which are contained in the data list `autism_data_large`. The remaining arguments are chosen to improve the visualization, for details we refer the reader to the help file `?qgraph`.

```
R> qgraph(fit_ADS$pairwise$wadj, layout = "spring", repulsion = 1,
+    edge.color = fit_ADS$pairwise$edgecolor,
+    nodeNames = autism_data_large$colnames,
+    color = autism_data_large$groups_color,
+    groups = autism_data_large$groups_list,
+    legend.mode = "style2", legend.cex = 0.4, vsize = 3.5, esize = 15)
```

The resulting visualization is shown in Figure 5. The layout of node positions was computed with the Fruchterman Reingold algorithm, which places nodes such that all the edges are of more or less equal length and there are as few crossing edges as possible (Fruchterman and Reingold 1991). Green edges indicate positive relationships, red edges indicate negative relationships and grey edges indicate relationships involving categorical variables for which no sign is defined. The width of the edges is proportional to the absolute value of the edge-weight. The node color indicates the different categories of the variables.

We observe a strong positive relationship between *age* and *age of diagnosis*, which makes sense because the two variables are logically connected. The negative relationship between *number of unfinished educations* and *satisfaction at work* seems plausible, too. Well-being is strongly connected in the graph, with the strongest connections to *satisfaction with social contacts* and *integration in society*. These three variables are categorical variables with 5, 3 and 3 categories, respectively. In order to investigate the exact nature of these interactions, one can look up all parameters using the function `showInteraction()`.

Estimating higher-order mixed graphical models

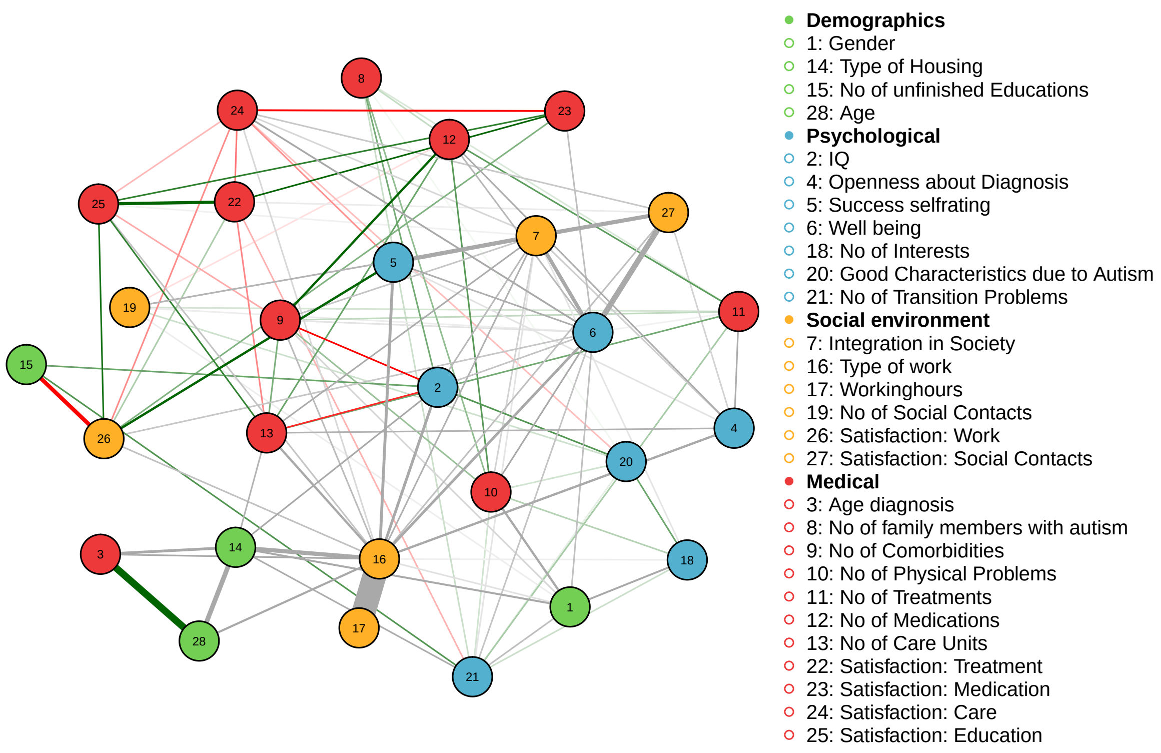
In the previous section, we focused on the estimation of pairwise ($k = 2$) MGMs. Here, we show how to estimate an MGM of order $k = 3$ to a data set consisting of post-traumatic stress disorder (PTSD) symptoms reported from 344 survivors of the Wenchuan earthquake in China reported in McNally, Robinaugh, Wu, Wang, Deserno, and Borsboom (2015). The data set is loaded automatically with **mgm** and includes the following symptoms:

```
R> PTSD_data$names

[1] "intrusion" "dreams"    "flash"     "upset"     "physior"   "avoidth"
```

We first specify the data, type, levels and the desired method to select the regularization parameter λ, similarly to the pairwise MGM. But here we specify with `k = 3` to estimate *all* pairwise and *all* 3-way interactions.

In addition, we choose to use the overparameterized version of the representation of categorical variables by setting `overparameterize = TRUE`. This results in that all states of categorical



● Demographics
○ 1: Gender
○ 14: Type of Housing
○ 15: No of unfinished Educations
○ 28: Age
● Psychological
○ 2: IQ
○ 4: Openness about Diagnosis
○ 5: Success selfrating
○ 6: Well being
○ 18: No of Interests
○ 20: Good Characteristics due to Autism
○ 21: No of Transition Problems
● Social environment
○ 7: Integration in Society
○ 16: Type of work
○ 17: Workinghours
○ 19: No of Social Contacts
○ 26: Satisfaction: Work
○ 27: Satisfaction: Social Contacts
● Medical
○ 3: Age diagnosis
○ 8: No of family members with autism
○ 9: No of Comorbidities
○ 10: No of Physical Problems
○ 11: No of Treatments
○ 12: No of Medications
○ 13: No of Care Units
○ 22: Satisfaction: Treatment
○ 23: Satisfaction: Medication
○ 24: Satisfaction: Care
○ 25: Satisfaction: Education

Figure 5: Visualization of the MGM estimated on the autism data set. Green edges indicate positive relationships, red edges indicate negative relationships and grey edges indicate relationships involving categorical variables for which no sign is defined. The width of the edges is proportional to the absolute value of the edge-parameter. The colors of the nodes map to the different domains *Demographics*, *Psychological*, *Social Environment* and *Medical*.

variables up to degree k are modeled explicitly. This overparameterization is possible due to the ℓ_1-penalization (for details see Friedman *et al.* 2010). The standard and the overparameterized parameterization are statistically equivalent and therefore one has to choose one over the other based on which parameterization lends itself to the most useful interpretation in a given application: If it is more sensible to compare all categories to a reference category the standard parameterization is preferable. If one is interested in all categories equally, the overparameterization might be better. We call `mgm()` with the above discussed specifications:

```
R> fit_mgmk <- mgm(data = PTSD_data$data, type = PTSD_data$type,
+     level = PTSD_data$level, lambdaSel = "EBIC", lambdaGam = 0.25,
+     k = 3, overparameterize = TRUE)
```

The output object `fit_mgmk` has the same structure as the pairwise MGM discussed above. We still find the pairwise interactions in `fit_mgmk$pairwise` but these do not represent the full parameterization anymore, since we also estimated 3-way interactions. All interactions that have been estimated to be nonzero can be found in the list `fit_mgmk$interactions`: The entry `fit_mgmk$interactions$indicator` contains a list showing all nonzero estimated interactions, separately for each order (here 2 and 3):

```
R> fit_mgmk$interactions$indicator

[[1]]
[,1] [,2]
[1,]    1    2
[3,]    4    5

[[2]]
[,1] [,2] [,3]
[1,]    1    3    4
[2,]    1    3    5
[3,]    2    3    4
[4,]    3    5    6
[5,]    4    5    6
```

The output indicates that we estimated two nonzero pairwise interactions, and five nonzero 3-way interactions. For example, the third row in the second list entry indicates that there is a 3-way interaction between variables 2–3–4 (*Dreams*, *Flashbacks* and *Upset*). The list `fit_mgmk$interactions` also contains additional entries for the strength of each interaction, and all parameters specifying the interaction (more than one parameter in case of categorical variables, see Section 2.1).

If we were to visualize the dependency structure of this $k = 3$-order MGM in a common undirected graph, we would lose the information about on which interaction(s) a dependency (edge) is based on. For instance, an edge between the nodes 1 and 2 could either be due to a pairwise interaction between 1 and 2, or due to any 3-way interaction including the nodes 1 and 2, or both. A visualization that allows to represent different orders of interactions is the factor graph (e.g., Koller and Friedman 2009). A factor graph is a bipartite graph that includes nodes for variables on the one hand, and nodes for interactions on the other hand. We use the function `FactorGraph()` to plot such a factor graph.

```
R> FactorGraph(object = fit_mgmk, labels = PTSD_data$names,
+    PairwiseAsEdge = FALSE)
```

This results in Figure 6(a). The six circle nodes represent the six variables in the data set. The red square factor nodes indicate pairwise interactions and the blue square factor nodes indicate 3-way interactions. Each factor node connects to two (pairwise) or three (3-way) variables, indicating an interaction between the respective variables. The width of the edges are proportional to the absolute value of the weight of the corresponding interaction.

We have a closer look at the 3-way interaction 2–3–4 (*Dreams*, *Flashbacks*, and *Feeling Upset*) in Figure 6(b): First look at the marginal probability cross-table of the variables *Dreams* and *Upset*, which shows unequal cell probabilities and hence an interaction between those two variables. Now we condition on the two states of a third variable *Flashbacks* and see that the interaction between *Dreams* and *Upset* considerably depends on whether an individual has *Flashbacks* or not.

Interpreting a k-way interaction by interpreting the $k-1$ way interaction for several levels of the variable X_j in the interaction can be seen as a moderation by X_j. In Haslbeck,

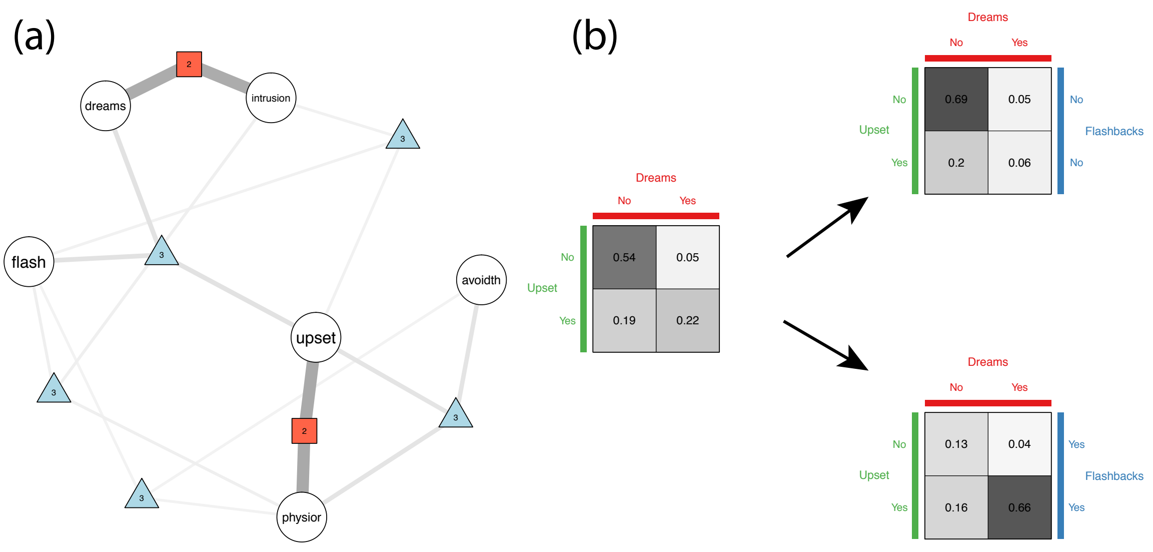


Figure 6: (a) Factor graph visualization of the estimated $k = 3$ MGM. The circle nodes refer to variables, the quadratic nodes refer to factors over two variables, and the triangle nodes refer to factors over three variables. The width of the edges is proportional to the strength of the factor; (b) the marginal sample probability cross-table of *Dreams* and *Upset*, and the same table conditioned on the two states of *Flashbacks*. We see that the relationship between *Dreams* and *Upset* depends on *Flashbacks*

Borsboom, and Waldorp (2020) we explain this approach of interpreting $k = 3$ interactions as moderation in more detail, and provide further examples for estimating and interpreting higher-order MGMs for the special case of continuous variables.

3.2. Stationary mixed VAR models

In this section, we first show how to estimate a mixed VAR model, compute predictions from it and visualize it based on simulated data. Then we show how to specify and sample from a mixed VAR model in order to generate the data used earlier for estimation. We then fit a mixed VAR model of order 3 to resting state fMRI data.

Estimating mixed VAR models

Here we show how to use the function `mvar()` to fit a mixed VAR model to a time series of six variables, consisting of four categorical variables (with 2, 2, 4 and 4 categories) and two Gaussian variables. In the true mVAR model from which the time series was sampled, there are effects of lag order 1 from variable 6 on 5, from 5 on 1 and from 3 on 1. The exact parameterization of these interactions is shown later in this section, where we create this data set with the function `mvarsampler()`.

We provide the data (which is an example data set automatically loaded with **mgm**), and specify the type of each variable in `type`, where `"g"` stands for Gaussian, `"p"` for Poisson, and `"c"` for categorical. Next, we provide the number of levels for each variable via `levels`, where we choose 1 for continuous variables by convention. We specify a lag of order 1 and select the regularization parameter λ using the EBIC with tuning parameter $\gamma = 0.25$:

```
R> fit_mvar <- mvar(data = mvar_data$data,
+    type = c("c", "c", "c", "c", "g", "g"), level = c(2, 2, 4, 4, 1, 1),
+    lambdaSel = "EBIC", lambdaGam = 0.25, lags = 1)
```

`mvar()` returns a list with several entries: `fit_mvar$wadj` is a $p \times p \times |L|$ array of edge weights, where $|L|$ is the number of specified lags. For example, `fit_mvar$wadj[3, 5, 1]` corresponds to the parameter for the crossed lagged effect of 5 on 3 over the first lag specified in `lags` (in this example we only specified one lag). `fit_mvar$signs` has the same dimension as `fit_mvar$wadj` and contains the signs of all parameters, if defined. `fit_mvar$rawlags` contains the full parameterization of the cross-lagged effects. If the mixed VAR model contains only continuous variables, the information in `fit_mvar$wadj` and `fit_mvar$rawlags` is equivalent. Similarly to `mgm()`, the entry `fit_mvar$intercepts` contains all thresholds (intercepts) and `fit_mvar$nodemodels` contains the p **glmnet** models of the p neighborhood regressions. Here we show the interaction parameters of the fitted VAR model for the single specified lag of order 1:

```
R> round(fit_mvar$wadj[, , 1], 2)

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0 0.33 0.06 0.41 0.00
[2,]    0    0 0.00 0.00 0.00 0.00
[3,]    0    0 0.00 0.00 0.00 0.00
[4,]    0    0 0.00 0.00 0.00 0.00
[5,]    0    0 0.00 0.00 0.00 0.31
[6,]    0    0 0.00 0.00 0.00 0.00
```

The autoregressive effects are on the diagonal and the cross-lagged effects are on the off-diagonal. We use a representation in which columns predict rows, which means that the entry `fit_mvar$wadj[5, 3, 1]` corresponds to the cross-lagged effect of 3 on 5 at lag 1. Comparing the estimates with the true cross-lagged effects indicated above, we see that all three true cross-lagged effects have been recovered and all other effects are correctly set to zero.

The additional arguments that can be provided to `mvar()` are similar to the ones in `mgm()`: The regularization parameter λ can be selected using the EBIC with a specified hyperparameter γ or with cross-validation with a specified number of folds. The candidate λ sequence is computed as in `mgm()` (see Section 3.1). The α in the elastic-net penalty can be selected with the EBIC or cross-validation, similarly to how λ is selected. Again similarly to `mgm()`, the `weights` argument allows to weight observations, `binarySign` allows signs for interactions involving binary variables, `threshold` defines the type of thresholding (see Section 2.3) and `overparameterize` allows to choose the preferred type of parameterization of interactions involving categorical variables. For additional input arguments see `?mvar`.

In many situations, one fits a VAR model to data that do *not* consist of a sequence of measurements that are equally spaced in time. The reason for this can be (randomly) missing measurements and gaps implied by the measurement process: For instance, in an experience sampling study, individuals may be asked to respond to questions about symptoms 6 times a day at equal time intervals of three hours. A mixed VAR model would then show how the presence of a symptom at a given time point is related to the presence of that and other symptoms at earlier time points (3h ago, 6h ago, etc.). However, because the individual sleeps at night, there are gaps in the time series. If one did not take this information into account, every seventh data point in the time series would represent a lag with the length of the night-gap, whereas the other six are representing a lag of three hours. This problem can be avoided by providing an integer sequence via the argument `consec`, which indicates whether measurements are consecutive. For instance if one has a time series with 12 time points (2 days of measurements in the above example), one would provide the vector `c(1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6)`. If one specifies a lag of order 1, `mvar()` then excludes the time step (over night) from measurement 6 to 7. Alternatively one can specify the notification number and the day number via the arguments `beepvar` and `dayvar`, respectively. Then the `consec` variable is computed internally. If a larger number of lags is included, more measurements are excluded accordingly. Information about which cases were excluded as well as the final data matrix used for estimation can be found in `mvar$call`. For more details see `?mvar` and the application example for the time-varying mVAR model below.

Making predictions from mixed VAR models

Here we show how to use the `predict()` function to compute predictions and nodewise errors from the model estimated in the previous section. We provide the fit object `mvar_fit` and the data as arguments:

```
R> pred_mgm <- predict(object = fit_mvar, data = mvar_data$data,
+    errorCon = c("RMSE", "R2"), errorCat = c("CC", "nCC"))
```

`pred_mgm$call` contains the function call, `pred_mgm$predicted` the predicted values for each row in the provided data matrix, and `pred_mgm$probabilities` contains the predicted probabilities for categorical variables. `pred_mgm$errors` contains a table of nodewise errors. Similarly to Section 3.1 we specified the root mean squared error (RMSE) for continuous variables and the (normalized) accuracy for categorical variables:

```
R> pred_mgm$errors

Variable  RMSE    R2    CC   nCC
1        1    NA    NA 0.754 0.495
2        2    NA    NA 0.523 0.000
3        3    NA    NA 0.302 0.000
4        4    NA    NA 0.266 0.000
5        5 0.916 0.157    NA    NA
6        6 0.998 0.000    NA    NA
```

Node 1 has the highest normalized accuracy, which makes sense because it is predicted by three other nodes at the previous time point. Nodes 2, 3 and 4 have a normalized accuracy of 0, because they are not predicted by any other node. Node 6 has a proportion of explained variance of 0, because it is not predicted by any other node, and node 5 has a nonzero proportion of explained variance because it is predicted by node 6.

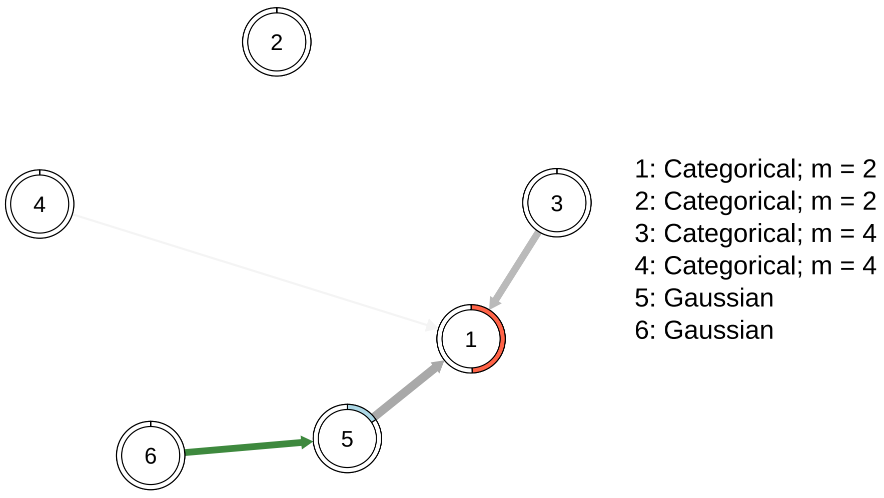


Figure 7: We visualize the lagged interaction parameters of the mixed VAR model estimated together with the nodewise errors. Green edges indicate positive relationships. Grey edges indicate that no sign is defined for the pairwise interaction (in the case the interaction involves categorical variables). The width of the edges is proportional to the absolute value of the edge-parameter.

One can also provide customary error functions via the `errorCon` and `errorCat` arguments. For details, see `?predict.mgm`.

Visualizing mixed VAR models

We visualize the lagged interaction parameters of the mixed VAR model estimated above together with the nodewise errors computed in the previous section. Specifically, we visualize the proportion of explained variance for the two continuous variables, and the normalized accuracy for the four categorical variables:

```
R> errors <- c(pred_mgm$errors[1:4, 5], pred_mgm$errors[5:6, 3])
R> qgraph(t(fit_mvar$wadj[, , 1]), edge.color = t(fit_mvar$edgecolor[, , 1]),
+    pie = errors, pieColor = c(rep("tomato", 4), rep("lightblue", 2)),
+    nodeNames = c(paste0("Categorical; m=", c(2, 2, 4, 4)),
+      rep("Gaussian", 2)))
```

We transposed the parameter matrix `fit_mvar$wadj[, , 1]` because `qgraph()` draws arrows from rows to columns instead of columns to rows, the latter of which is the data structure used in `mvar()`. The resulting plot is shown in Figure 7.

The green edge indicates a positive linear relationship for the cross-lagged effect from node 6 on node 5. The remaining edges are grey, indicating that no sign is defined. This is because these interactions are defined by several parameters, so no sign can be defined. The width of the edges is proportional to the absolute value of the estimated edge-weights (the values in `fit_mvar$wadj[, , 1]`).

Sampling from mixed VAR models

We now use the function `mvarsampler()` to sample the data set `mvar_data` used in the previous section. We specify a model with only one lag of order one and $p = 6$ variables, four categorical (with 2, 2, 4 and 4 categories) and two Gaussians:

```
R> p <- 6
R> type <- c("c", "c", "c", "c", "g", "g")
R> level <- c(2, 2, 4, 4, 1, 1)
R> max_level <- max(level)
R> lags <- 1
R> n_lags <- length(lags)
```

Next, we specify the thresholds for each variable. We assign one threshold (intercept) to the Gaussians, and a separate threshold for each of the categories of each of the categorical variables. These thresholds correspond to the first summation in the joint MGM density in Equation 5. In addition, we define a vector indicating the standard deviations of the Gaussian nodes. Note that entries of that vector that do not correspond to Gaussian variables in `type` are ignored.

```
R> thresholds <- list()
R> for (i in 1:p) thresholds[[i]] <- rep(0, level[i])
R> sds <- rep(1, p)
```

Finally, we specify the lagged effects in a 5-dimensional $p \times p \times \max\{\text{levels}\} \times \max\{\text{levels}\} \times |L|$ array, where $|L|$ is the number of lags `n_lags`. We first specify the lagged effect from the continuous variable 6 on the continuous variable 5, which consists of a single parameter:

```
R> coefarray <- array(0, dim = c(p, p, max_level, max_level, n_lags))
R> coefarray[5, 6, 1, 1, 1] <- 0.4
```

We specify two additional lagged effects: one from the categorical variable 3 on the categorical variable 1, which is parameterized by 2×4 parameters; and one from the continuous variable 5 to the binary variable 1, which is parameterized by 2×1 parameters.

```
R> coefarray[1, 5, 1:level[1], 1:level[5], 1] <- c(0, 1)
R> m1 <- matrix(0, nrow = level[2], ncol = level[4])
R> m1[1, 1:2] <- 1
R> m1[2, 3:4] <- 1
R> coefarray[1, 3, 1:level[2], 1:level[4], 1] <- m1
```

Finally, all arguments are provided to `mvarsampler()`:

```
R> mvar_data <- mvarsampler(coefarray = coefarray, lags = lags,
+    thresholds = thresholds, sds = sds, type = type, level = level,
+    N = 200, pbar = TRUE)
```

These sampled data correspond to the example data set in `mvar_data` we used above to illustrate how to estimate a mVAR model.

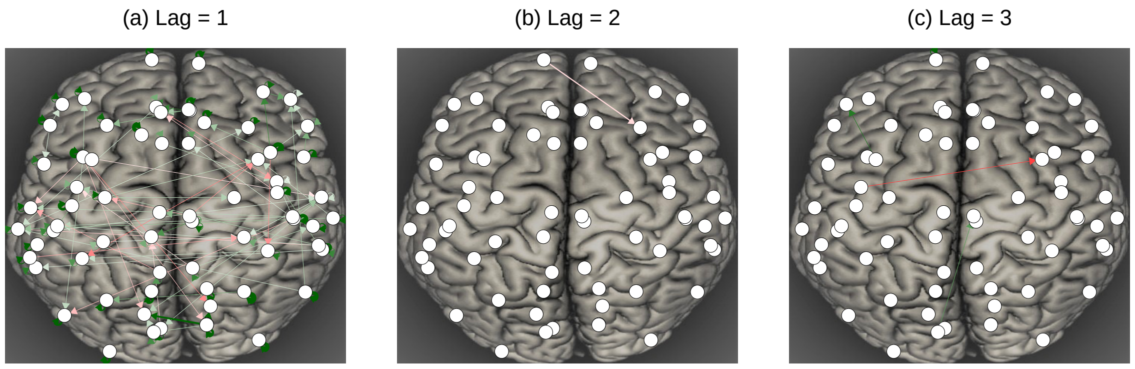


Figure 8: Visualization of the fitted mVAR model, where we depict the parameters separately for each lag. Red edges indicate positive relationships, green edges indicate negative relationships. The width of the edges is proportional to the absolute value of the edge-parameter.

Application: Resting state fMRI data

We fit an mVAR model with lags 1, 2 and 3 to resting state fMRI data of a single participant. The data set consists of BOLD measurements of 68 voxels for 240 time points, where the average sampling frequency is 2 seconds (for details see Schmittmann, Jahfari, Borsboom, Savi, and Waldorp 2015). The data set is loaded automatically with the **mgm** package. All BOLD measurements are modeled as conditional Gaussians, and accordingly we specify the number of levels to be equal to 1 for all variables. We select the regularization parameter λ with the EBIC with tuning parameter $\gamma = 0.25$, and we include lags of order 1, 2 and 3.

```
R> rs_mvar <- mvar(data = restingstate_data$data, type = rep("g", 68),
+    level = rep(1, 68), lambdaSel = "EBIC", lambdaGam = 0.25,
+    lags = c(1, 2, 3))
```

We visualize the $68 \times 68 \times 3$ interaction parameters of this VAR model in `rs_mvar$wadj` in three separate network plots in Figure 8, one for each lag. We provide code to reproduce Figure 8 from the package example data set `restingstate_data` in the online supplementary materials and on the Github repository `https://github.com/jmbh/mgmDocumentation`.

For the lag of size one, many coefficients are nonzero. In contrast, for the lags of size two and three only few coefficients are nonzero. For a typical fMRI data analysis, this could mean that it is sufficient to fit a VAR model of lag 1 in order to reduce the variance for further analyses.

Similarly to MGMs, the function `resample()` can be used to obtain bootstrapped sampling distributions for the parameters of the mVAR model.

3.3. Time-varying mixed graphical models

In this section we show how to estimate a time-varying MGM, how to compute predictions from it and how to visualize it.

Estimating time-varying mixed graphical models

We fit a time-varying MGM to gene expression data used by Gibberd and Nelson (2017), who took a subset of the data presented by Arbeitman *et al.* (2002). Specifically, we model $p = 150$ gene expressions related to the immune system of D. melanogaster (the fruit fly) measured at $n = 67$ time points across its whole life span. Since $p > n$, this is an example of a high-dimensional estimation problem. Figure 9 (top panel) shows that the 67 measurements are distributed unequally across the time interval.

Estimating the type of time-varying models introduced in Section 2.5 requires the specification of a bandwidth parameter σ that reflects how many time points are combined locally for estimation. The bandwidth parameter σ is the standard deviation of the Gaussian distribution that defines the weighting function (see Section 2.5). The empirical time points are normalized to the interval $[0, 1]$ and the Gaussian weighting function is defined on this interval. This allows some intuition about which σ is appropriate. For example, $\sigma = 2$ implies weights that are close to uniform on the interval $[0, 1]$ and therefore gives similar results as the stationary model. This intuition allows to specify a *candidate* σ-sequence. We select σ with the function `bwSelect()`, which computes prediction errors on leave-out sets for all candidate σ-values and selects the σ^* that has the lowest mean error. In the next paragraph we describe this approach in detail.

The function `bwSelect()` fits time varying models on an equally spaced sequence between the time points j and $n - F + j - 1$ of length J with $j \in \{1, 2, \dots, F\}$, where F is the number of folds (times the procedure is repeated) while leaving out (weighting to zero) the time point at which the model is estimated. In a second step, the data at this time point are predicted with the time-varying model and an error measure is computed (RMSE for non-categorical, 0/1-loss for categorical). This procedure is repeated F times. Then we take the arithmetic mean over J estimation points, p variables and F folds. If $J = \frac{n}{F}$, this procedure is equal to a time-stratified F-fold cross-validation scheme. We allow to specify $J < \frac{n}{F}$ to save computational cost. J is specified by the argument `bwFoldsize` and F is specified by the argument `bwFolds`. Selecting the ratio between `bwFoldsize` and n corresponds to the problem of selecting the number of folds in cross-validation (see, e.g., Friedman, Hastie, and Tibshirani 2001).

For the present illustration we select `bwFolds = 5` and `bwFoldsize = 5` to keep the computation time short. We provide the candidate σ-sequence $\{0.1, 0.2, 0.3, 0.4\}$. And we provide all arguments for the time-varying MGM. This is because we repeatedly fit the type of model we want as our final model (then with fixed σ). We provide the time points of measurements `fruitfly_data$timevector` via the argument `timepoints` (see Figure 2 in Section 2.5 for an explanation of why one has to provide the time points if they are not equally spaced). Finally, we specify the class of time-varying model `modeltype = "mgm"` and pass the arguments `k`, `threshold` and `ruleReg`, to `tvmgm()` (see Section 3.1 on `mgm()` for a description of these arguments).

```
R> set.seed(1)
R> p <- ncol(fruitfly_data$data)
R> bw_tvmgm <- bwSelect(data = fruitfly_data$data, type = rep("g", p),
+    level = rep(1, p), bwSeq = c(0.1, 0.2, 0.3, 0.4), bwFolds = 5,
+    bwFoldsize = 5, timepoints = fruitfly_data$timevector,
+    modeltype = "mgm", k = 2, threshold = "none", ruleReg = "OR")
```

Note that the code above takes about 3 hours to run.

We would like to know which candidate bandwidth minimized the average prediction error. This information is stored in `bw_tvmgm$meanError`:

```
R> round(bw_tvmgm$meanError, 3)
```

```
0.1   0.2   0.3   0.4
0.826 0.707 0.630 0.640
```

We see that $\sigma = 0.3$ minimizes the error in this data set. If the smallest/largest candidate σ minimized the prediction error, it is advisable to extend the candidate σ sequence to smaller/larger values.

After obtaining a reasonable bandwidth for this data set, we can estimate the final time-varying MGM. The estimation points are specified on the unit interval $[0, 1]$ to which the provided time scale is normalized internally. We choose 20 equally spaced time points across the time series by setting `estpoints = seq(0, 1, length.out = 20)`. Finally, we specify the above obtained bandwidth with `bandwidth = 0.3` and set a random seed to ensure reproducibility.

```
R> set.seed(1)
R> fit_tvmgm <- tvmgm(data = fruitfly_data$data, type = rep("g", p),
+    level = rep(1, p), timepoints = fruitfly_data$timevector,
+    estpoints = seq(0, 1, length.out = 20), k = 2, bandwidth = 0.3,
+    threshold = "none", ruleReg = "OR")
```

The output list in the fit object `fit_tvmgm` is similar to the list returned by `mgm()`. The difference is that all parameter matrices are now 3-dimensional arrays, with an additional dimension for the estimated time points $t^e \in \mathcal{E}$. For instance, the edge parameters of the pairwise MGM estimated at the third estimation point $t^e = 3$ are stored in the matrix `fit_tvmgm$pairwise$wadj[, , 3]`. For a a detailed description of all output provided in `fit_tvmgm`, see the help file `?tvmgm`.

Making predictions from time-varying mixed graphical models

When making predictions with time-varying MGMs, in principle we would need to estimate the time-varying model at the maximum resolution, that is, at every time point. However, this would be computationally expensive: For example, for a time-series of $n = 1000$ time points, we would need to fit 1000 models in order to compute predictions. The `predict` method in **mgm** provides two different options in order to compute predictions and nodewise errors across time, without requiring to estimate n models.

The first option, `tvMethod = "weighted"`, computes predictions for each of the n time points from *all* models $t^e \in \mathcal{E}$. It then computes a weighted average over the predictions of all models at each time point. The weight is equal to the weight of the kernel function at t for the respective model estimated at t^e. The second option is `tvMethod = "closestModel"`, which for each time point determines the closest estimation point t^e, and then uses this model for prediction. Accordingly, local nodewise errors are calculated only from the closest model.

Note that if one estimates n models at equally spaced time points, this method corresponds to the above described situation of estimating a time-varying model for each time point.

In order to compute predictions we call the `predict()` function and provide the data, the fit object and the desired method to compute predictions. Here we pick `tvMethod = "weighted"`:

```
R> pred_tvmgm <- predict(object = fit_tvmgm, data = fruitfly_data$data,
+    tvMethod = "weighted")
```

The output object `pred_tvmgm` is a list containing the function call `pred_tvmgm$call`, the predicted values `pred_tvmgm$predicted` and `pred_tvmgm$probabilities` (in the case of categorical variables) computed by the method `tvMethod = "weighted"`. `pred_tvmgm$true` contains the true data matrix and `pred_tvmgm$errors` contains an array of local nodewise error, where the third dimension indicates the estimation points.

Visualizing time-varying mixed graphical models

Figure 9 displays several aspects of the time-varying MGM estimated on the fruit fly data above. The top panel shows the number of edges (solid line) estimated across the time series of 67 measurements, which decreases across the time series. This can be explained by the small number of measurements available at the end of the time series (see red dashes on the time arrow). To make this explicit, we plot $n_{\sigma=0.3,t^e}$, the used sample size at a given estimation point (see Section 2.5). We see that extremely few data points are available in the end of the time series, resulting in a very low sensitivity to detect edges. The lower panel shows the undirected network at the 2nd, 6th and 13th estimation point out of 20 equally spaced estimation points across the whole time series (blue dashes).

While we can interpret the MGM at each estimation in context of the local n_{σ,t^e}, it is difficult to interpret changes over time, because the sensitivity of the algorithm decreases towards the end of the time series (because less data is available) and hence it is unclear whether edges in the end of the time series are absent in the true model or whether the sensitivity of the algorithm was too low to detect them. This highlights the importance of collecting data with a roughly constant sampling frequency. We provide code to exactly reproduce Figure 9 from the package example data set `fruitfly_data` in the online supplementary materials and on the Github repository `https://github.com/jmbh/mgmDocumentation`.

Sampling from time-varying mixed graphical models

The function `tvmgmsampler()` allows to sample from a time-varying MGM. The function input is identical to the input to `mgmsampler()`, the sampling function of the stationary MGM described in Section 3.1, except that the arguments `thresholds`, `sds` and `interactions` have an additional dimension for time. The number of indices in this additional time dimension defines the length of the time series. Thus, a separate model is specified for each time point in the time series. For details see `?tvmgmsampler`.

Bootstrap sampling distributions

Similarly to stationary MGMs, the function `resample()` can be used to obtain bootstrapped sampling distributions for the parameters of the time-varying MGM. The only difference

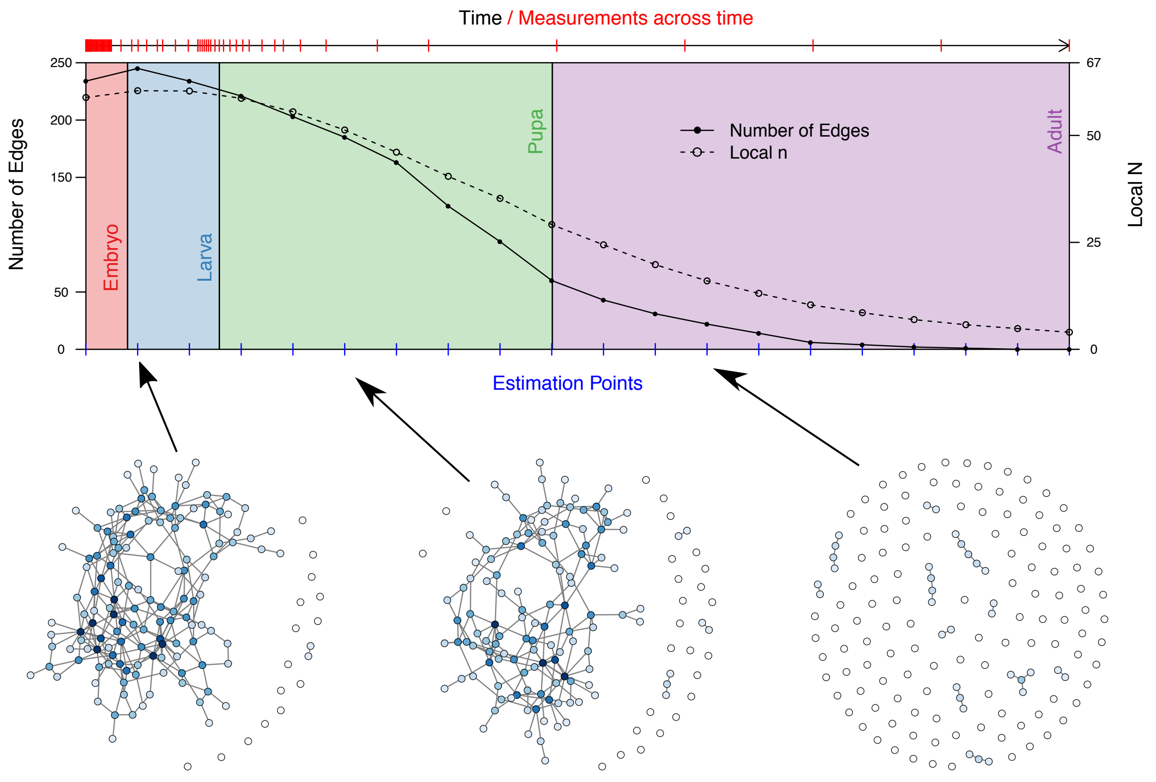


Figure 9: Top panel: the number of estimated edges (solid line) and the local sample size $n_{\sigma=0.3,t^e}$ (dashed line) at each estimation point. The red dashes indicate the available measurement on the true time scale. The four colored areas indicate the four stages of the life cycle of the fruit fly. Bottom panel: the undirected network plottet at three different estimation points 2, 6, 13 (with 20 estimation points equally distributed across the 67 time points).

is that we use a block-bootstrapping scheme to ensure that data points remain reasonably distributed across time. The number of blocks can be specified with the argument `blocks` in the `resample()` function. The larger the number of blocks, the more evenly distributed the bootstrap samples are across the time interval and the higher the similarity between bootstrap samples. Since even distribution across time and low similarity across bootstrap samples is desirable, the number of blocks controls this trade-off. For more details see the help file `?resample`.

3.4. Time-varying mixed VAR models

We illustrate how to fit a time-varying mixed VAR model on a symptom time series with 51 variables measured on 1478 time points during 238 consecutive days from an individual diagnosed with major depression (Wichers, Groot, Psychosystems, ESM Group, and EWS Group 2016). The measured variables include questions regarding mood, self-esteem, social interaction, physical activity, events and symptoms of depression (see also legend in Figure 10). During the measured time interval, a double-blind medication dose reduction was carried out, consisting of a baseline period, the dose reduction, and two post assessment periods (see Figure 10, the points on the time line correspond to the two dose reductions). For a detailed description of this data set see Kossakowski, Groot, Haslbeck, Borsboom, and Wichers (2017).

Estimating time-varying mixed VAR models

We provide the data, the type (continuous and categorical), and the levels for each variable, all of which are contained in the data list `symptom_data` (automatically loaded with **mgm**), similarly to specifying `mvar()`. Next, we provide the day number with `dayvar` and the number of notification on each day with `beepvar`. Alternatively, one could manually compute a single vector that indicates the consecutiveness of measurements and provide it via the argument `consec`. We provide this information because the measurements in this data set are not consecutive, both because of the day-night break in which no measurements are taken and because of randomly missing measurement points. If we did not provide this information, the resulting parameters would represent a mixture of effects across different lags and are therefore not interpretable anymore. We explained this in detail in Section 2.5. The function `tvmvar()` uses this information to fit the model only on rows of the time series for which sufficient previous measurements are available (1 for lag 1, 2 for lag 2, etc.).

In order to fit a time-varying MGM we need to choose an appropriate bandwidth parameter σ, which determines how many observations close in time we combine in order to estimate a local model (see Section 2.5). In Section 3.3, we provided an explanation of how to use `bwSelect()` to select an appropriate σ using a time-stratified cross-validation scheme. Here we choose $\sigma = 0.2$.

We specify a lag of order 1 and via the argument `estpoints` we specify that we would like to estimate the model at 20 equally spaced time intervals throughout the time series. We specify the sequence of estimation points on the unit interval $[0, 1]$, to which the provided time scale is normalized internally. Finally, we set thresholding `threshold = "none"` and set a random seed to ensure reproducibility.

```
R> set.seed(1)
R> fit_tvmvar <- tvmvar(data = symptom_data$data, type = symptom_data$type,
+    level = symptom_data$level, beepvar = symptom_data$data_time$beepno,
+    dayvar = symptom_data$data_time$dayno, lags = 1,
+    estpoints = seq(0, 1, length.out = 20), bandwidth = 0.2,
+    threshold = "none", saveData = TRUE)
```

Note that the code above takes about 15 minutes to run.

The output of `tvmvar()` is similar to the output of `?mvar` described in Section 3.2. The difference is that all entries have now an additional dimension for estimation points. For example, the entry of the parameter array `fit_tvmvar$wadj[4, 9, 2, 15]` indicates the cross lagged effect of 9 on 4 over the second specified lag in `lags` at the 15th estimation point. The array `fit_tvmvar$signs` has the same dimension and specifies the signs of the parameters in `fit_tvmvar$wadj`, if defined. For a discussion of when a sign is defined for an edge-parameter see Section 3.1. The object `fit_tvmvar$intercepts` contains a list with time-varying thresholds/intercepts and `fit_tvmvar$tvmodels` contains the models at each of the $|\mathcal{E}|$ estimation points.

We provided the day and notification number of each measurement and `tvmvar()` used this information to only include measurements in the model for which sufficient previous measurements are available. By executing the model object in the console, we get the number of measurements that were actually used for estimation:

```
R> fit_tvmvar

  mgm fit-object
  Model class:  Time-varying mixed Vector Autoregressive (tv-mVAR) model
  Lags:  1
  Rows included in VAR design matrix:  876 / 1476 ( 59.35 %)
  Nodes:  48
  Estimation points:  20
```

We see that for 876 of 1476 measurement points the previous measurement (requirement of lag 1) is available and were therefore used for estimation. If we included lags with higher order the number of usable measurements would become smaller.

Making predictions from time-varying mixed VAR models

In order to compute predictions from the mixed VAR model we have to choose between the two options `tvMethod = "weighted"` and `tvMethod = "closestModel"`. For a discussion of these two methods see Section 3.3 or the help file `?predict.mgm`. Next to the fit object we provide the data and information about the consecutiveness of measurements to `predict()`:

```
R> pred_tvmvar <- predict(object = fit_tvmvar, data = symptom_data$data,
+    tvMethod = "weighted", beepvar = symptom_data$data_time$beepno,
+    dayvar = symptom_data$data_time$dayno)
```

The output object `pred_tvmvar` is a list containing the function call `pred_tvmgm$call`, the predicted values `pred_tvmgm$predicted` and `pred_tvmgm$probabilities` (in the case of categorical variables) computed by the method `tvMethod = "weighted"`. `pred_tvmgm$true` contains the true data matrix, which is useful in the case of VAR models, when not all rows in the original data matrix are necessarily used to fit the VAR model (see previous section). Finally, `pred_tvmgm$errors` is an array of local nodewise errors, where the third dimension indexes estimation points. For instance, `pred_tvmgm$errors[, , 9]` contains the nodewise errors for estimation point 9.

Visualizing time-varying mixed VAR models

Figure 10 displays some aspects of the time varying mixed VAR model estimated in the previous section. In the top row of Figure 10 we depict a network plot of the VAR(1) parameters at the estimation points 2, 6, and 16. Green edges indicate positive relationships and red edges indicate negative relationships. Grey edges indicate that no sign is defined, because the edge-weight is a function of several parameters, which is the case for interactions including categorical variables (see Section 3.1). The width of edges is proportional to the absolute value of the edge-weight. It is evident from the three network plots that the model changes considerably over time which suggests that a stationary model is not appropriate for these data. The second row depicts six autoregressive or cross-lagged effects across the

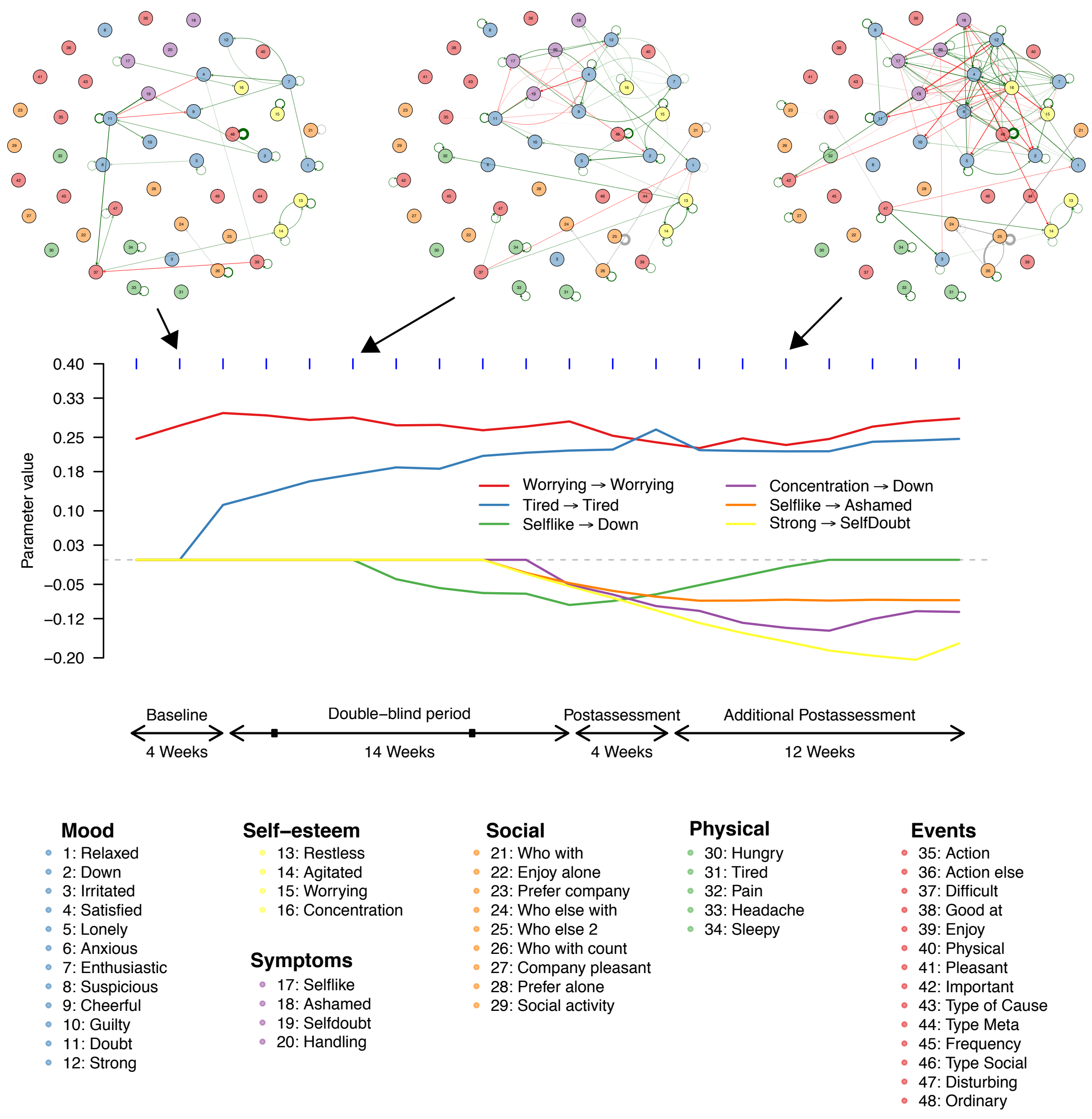


Figure 10: Top row: network visualization of VAR(1) parameters at the estimation points 2, 6, and 16. Green edges indicate positive relationships, red edges indicate negative relationships and grey edges indicate that no sign is defined. The color of the nodes corresponds to the group the variable belongs to (see legend); second row: six autoregressive (e.g., $\textit{Worrying}^{t-1} \rightarrow \textit{Worrying}^{t}$) or cross-lagged effects (e.g., $\textit{Selflike}^{t-1} \rightarrow \textit{Down}^{t}$) depicted as a function of time.

measured time interval. We see that parameters change considerably over time, for instance the autoregressive effect of *Tired* is strong at the beginning of the time series and decreases almost monotonously until the end of the measured time interval.

We provide code to exactly reproduce Figure 10 from the example data set `symptom_data` in the online supplementary materials and on the Github repository `https://github.com/jmbh/mgmDocumentation`.

Sampling from time-varying mixed VAR models

The function `tvmvarsampler()` allows to sample from a time-varying mVAR model. The function input is identical to the input to `mgmsampler()`, the sampling function of the stationary mVAR described in Section 3.2, except that the arguments `thresholds`, `sds` and `coefarray` have an additional dimension for time. The number of indices in this additional time dimension defines the length of the time series. Thus, a separate model is specified for each time point in the time series. For details see `?tvmvarsampler`.

Similarly to time-varying MGMs, the function `resample()` allows to obtain bootstrapped sampling distributions for the parameters of time-varying mixed VAR models.

4. Concluding comments

We presented the **mgm** package which allows to fit stationary and time-varying mixed graphical models and stationary and time-varying mixed vector autoregressive models. In addition to the estimation functions, we provide methods to compute predictions and nodewise errors and assess the stability of estimates via resampling. Furthermore, flexible sampling functions for all model classes allow the user to evaluate the performance of the estimation algorithms in a given situation via simulations. Finally, we provided fully reproducible code examples that illustrate how to use the software package.

The **mgm** package is under continuous development. We aim to add functions that allow one to inspect higher order interactions in an accessible way. We plan to implement different ways to select tuning parameters (λ penalization parameter, α elastic net parameter, σ bandwidth parameter), for instance with stability-selection (Meinshausen and Bühlmann 2010; Liu, Roeder, and Wasserman 2010). And we will implement other estimators than ℓ_α-penalized regression, which might be more appropriate in some situations. Finally, since all estimation algorithms are based on sequential regressions, considerable performance gains can be made by parallelizing the estimation algorithms.

Acknowledgments

We would like to thank three anonymous reviewers for their detailed and constructive feedback, and Fabian Dablander and Oisín Ryan for their comments on an earlier version of this manuscript.

References

Agresti A (2003). *Categorical Data Analysis*, volume 482. John Wiley & Sons.

Albert R, Barabasi AL (2002). "Statistical Mechanics of Complex Networks." *Reviews of Modern Physics*, **74**(1), 47–97. `doi:10.1103/revmodphys.74.47`.

Arbeitman MN, Furlong EEM, Imam F, Johnson E, Null BH, Baker BS, Krasnow MA, Scott MP, Davis RW, White KP (2002). "Gene Expression During the Life Cycle of Drosophila Melanogaster." *Science*, **297**(5590), 2270–2275. `doi:10.1126/science.1072152`.

Banerjee O, El Ghaoui L, d'Aspremont A (2008). "Model Selection through Sparse Maximum Likelihood Estimation for Multivariate Gaussian or Binary Data." *Journal of Machine Learning Research*, **9**, 485–516.

Begeer S, Wierda M, Venderbosch S (2013). *Allemaal Autisme, Allemaal Anders. Rapport NVA Enquete 2013 [All Autism, All Different. Dutch Autism Society Survey 2013].* Bilthoven. NVA.

Borsboom D, Cramer AOJ (2013). "Network Analysis: An Integrative Approach to the Structure of Psychopathology." *Annual Review of Clinical Psychology*, **9**(1), 91–121. doi:10.1146/annurev-clinpsy-050212-185608.

Bühlmann P, Kalisch M, Meier L (2014). "High-Dimensional Statistics with a View Toward Applications in Biology." *Annual Review of Statistics and Its Application*, **1**, 255–278. doi:10.1146/annurev-statistics-022513-115545.

Casas I, Fernandez-Casal R (2020). ***tvReg**: Time-Varying Coefficients Linear Regression for Single and Multiple Equations.* R package version 0.5.0, URL https://CRAN.R-project.org/package=tvReg.

Chen S, Witten DM, Shojaie A (2015). "Selection and Estimation for Mixed Graphical Models." *Biometrika*, **102**(1), 47–64. doi:10.1093/biomet/asu051.

Chen X, He Y (2018). "Inference of High-Dimensional Linear Models with Time-Varying Coefficients." *Statistica Sinica*, **28**(1), 255–276. doi:10.5705/ss.202015.0202.

Deserno MK, Borsboom D, Begeer S, Geurts HM (2017). "Multicausal Systems Ask for Multicausal Approaches: A Network Perspective on Subjective Well-Being in Individuals with Autism Spectrum Disorder." *Autism*, **21**(8), 960–971. doi:10.1177/1362361316660309.

Efron B (1992). "Bootstrap Methods: Another Look at the Jackknife." In *Breakthroughs in Statistics*, pp. 569–593. Springer-Verlag.

Efron B, Tibshirani RJ (1994). *An Introduction to the Bootstrap.* CRC Press.

Epskamp S (2018). ***graphicalVAR**: Graphical VAR for Experience Sampling Data.* R package version 0.2.2, URL https://CRAN.R-project.org/package=graphicalVAR.

Epskamp S, Borsboom D, Fried EI (2018). "Estimating Psychological Networks and Their Accuracy: A Tutorial Paper." *Behavior Research Methods*, **50**(1), 195–212. doi:10.3758/s13428-017-0862-1.

Epskamp S, Cramer AOJ, Waldorp LJ, Schmittmann VD, Borsboom D (2012). "**qgraph**: Network Visualizations of Relationships in Psychometric Data." *Journal of Statistical Software*, **48**(4), 1–18. doi:10.18637/jss.v048.i04.

Epskamp S, Deserno MK, Bringmann LF (2019). ***mlVAR**: Multi-Level Vector Autoregression.* R package version 0.4.4, URL https://CRAN.R-project.org/package=mlVAR.

Foygel R, Drton M (2010). "Extended Bayesian Information Criteria for Gaussian Graphical Models." In *Advances in Neural Information Processing Systems*, pp. 604–612.

Foygel R, Drton M (2015). "High-Dimensional Ising Model Selection with Bayesian Information Criteria." *Electronic Journal of Statistics*, **9**(1), 567–607. doi:10.1214/15-ejs1012.

Friedman J, Hastie T, Tibshirani R (2001). *The Elements of Statistical Learning*. Springer-Verlag.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics*, **9**(3), 432–441. doi:10.1093/biostatistics/kxm045.

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22. doi:10.18637/jss.v033.i01.

Friedman J, Hastie T, Tibshirani R (2019). ***glasso**: Graphical Lasso – Estimation of Gaussian Graphical Models*. R package version 1.11, URL https://CRAN.R-project.org/package=glasso.

Friedman N, Linial M, Nachman I, Pe'er D (2000). "Using Bayesian Networks to Analyze Expression Data." *Journal of Computational Biology*, **7**(3–4), 601–620. doi:10.1089/106652700750050961.

Fruchterman TMJ, Reingold EM (1991). "Graph Drawing by Force-Directed Placement." *Software: Practice and Experience*, **21**(11), 1129–1164. doi:10.1002/spe.4380211102.

Ghazalpour A, Doss S, Zhang B, Wang S, Plaisier C, Castellanos R, Brozell A, Schadt EE, Drake TA, Lusis AJ, Horvath S (2006). "Integrating Genetic and Network Analysis to Characterize Genes Related to Mouse Weight." *PLoS Genetics*, **2**(8), e130. doi:10.1371/journal.pgen.0020130.

Gibberd AJ, Nelson JDB (2016). "Estimating Dynamic Graphical Models from Multivariate Time-Series Data: Recent Methods and Results." In A Douzal-Chouakria, J Vilar, PF Marteau (eds.), *AALTD 2015: Advanced Analysis and Learning on Temporal Data*, volume 9785 of *Lecture Notes in Computer Science*, pp. 111–128. Springer-Verlag, Cham. doi:10.1007/978-3-319-44412-3_8.

Gibberd AJ, Nelson JDB (2017). "Regularized Estimation of Piecewise Constant Gaussian Graphical Models: The Group-Fused Graphical Lasso." *Journal of Computational and Graphical Statistics*, **26**(3), 623–634. doi:10.1080/10618600.2017.1302340.

Hamilton JD (1994). *Time Series Analysis*. 1st edition. Princeton University Press, Princeton.

Harrell Jr FE (2020). ***Hmisc**: Harrell Miscellaneous*. R package version 4.3-1, URL https://CRAN.R-project.org/package=Hmisc.

Haslbeck JMB (2020). ***mgm**: Estimating Time-Varying k-Order Mixed Graphical Models*. R package version 1.2-9, URL https://CRAN.R-project.org/package=mgm.

Haslbeck JMB, Borsboom D, Waldorp LJ (2020). "Moderated Network Models." *Multivariate Behavioral Research*. doi:10.1080/00273171.2019.1677207. Forthcoming.

Haslbeck JMB, Bringmann LF, Waldorp LJ (2019). "A Tutorial on Estimating Time-Varying Vector Autoregressive Models." arXiv:1711.05204 [stat.AP], URL http://arxiv.org/abs/1711.05204.

Haslbeck JMB, Waldorp LJ (2015). "Structure Estimation for Mixed Graphical Models in High-Dimensional Data." arXiv:1510.05677 [stat.AP], URL https://arxiv.org/abs/1510.05677.

Haslbeck JMB, Waldorp LJ (2018). "How Well Do Network Models Predict Future Observations? On the Importance of Predictability in Network Models." *Behavior Research Methods*, **50**, 853–861. doi:10.3758/s13428-017-0910-x.

Hastie T, Tibshirani R, Wainwright M (2015). *Statistical Learning with Sparsity*. CRC Press.

Huang S, Li J, Sun L, Ye J, Fleisher A, Wu T, Chen K, Reiman E (2010). "Learning Brain Connectivity of Alzheimer's Disease by Sparse Inverse Covariance Estimation." *NeuroImage*, **50**(3), 935–949. doi:10.1016/j.neuroimage.2009.12.120.

Immer A, Gibberd A (2017). "**GraphTime**: A Package for Dynamic Graphical Model Estimation." https://github.com/GlooperLabs/GraphTime.

Jiang H, Fei X, Liu H, Roeder K, Lafferty J, Wasserman L, Li X, Zhao T (2019). ***huge**: High-Dimensional Undirected Graph Estimation*. R package version 1.3.4, URL https://CRAN.R-project.org/package=huge.

Kolar M, Song L, Ahmed A, Xing EP (2010). "Estimating Time-Varying Networks." *The Annals of Applied Statistics*, **4**(1), 94–123. doi:10.1214/09-aoas308.

Kolar M, Xing EP (2012). "Estimating Networks with Jumps." *Electronic Journal of Statistics*, **6**, 2069–2106. doi:10.1214/12-ejs739.

Kolar M, Xing EP (2013). "Sparsistent Estimation of Time-Varying Discrete Markov Random Fields." arXiv:05677 [stat.ML], URL https://arxiv.org/abs/0907.2337.

Koller D, Friedman N (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Kossakowski J, Groot P, Haslbeck JMB, Borsboom D, Wichers M (2017). "Data from 'Critical Slowing Down as a Personalized Early Warning Signal for Depression'." *Journal of Open Psychology Data*, **5**(1). doi:10.5334/jopd.29.

Lauritzen SL (1996). *Graphical Models*. Number 17 in Oxford Statistical Science Series. Clarendon Press, Oxford.

Liu H, Roeder K, Wasserman L (2010). "Stability Approach to Regularization Selection (Stars) for High Dimensional Graphical Models." In *Advances in Neural Information Processing Systems*, pp. 1432–1440.

Loh PL, Wainwright MJ (2012). "Structure Estimation for Discrete Graphical Models: Generalized Covariance Matrices and Their Inverses." In *Advances in Neural Information Processing Systems*, pp. 2087–2095.

McNally RJ, Robinaugh DJ, Wu GW, Wang L, Deserno MK, Borsboom D (2015). "Mental Disorders as Causal Systems a Network Approach to Posttraumatic Stress Disorder." *Clinical Psychological Science*, **3**(6), 836–849. doi:10.1177/2167702614553230.

Meinshausen N, Bühlmann P (2006). "High-Dimensional Graphs and Variable Selection with the Lasso." *The Annals of Statistics*, **34**(3), 1436–1462. doi:10.1214/009053606000000281.

Meinshausen N, Bühlmann P (2010). "Stability Selection." *Journal of the Royal Statistical Society B*, **72**(4), 417–473. doi:10.1111/j.1467-9868.2010.00740.x.

Monti RP (2014). "**pySINGLE**." https://github.com/piomonti/pySINGLE.

Monti RP, Hellyer P, Sharp D, Leech R, Anagnostopoulos C, Montana G (2014). "Estimating Time-Varying Brain Connectivity Networks from Functional MRI Time Series." *NeuroImage*, **103**, 427–443. doi:10.1016/j.neuroimage.2014.07.033.

Nelder JA, Baker RJ (1972). "Generalized Linear Models." *Encyclopedia of Statistical Sciences*. doi:10.1002/0471667196.ess0866.

Nicholson W, Matteson D, Bien J (2019). ***BigVAR**: Dimension Reduction Methods for Multivariate Time Series*. R package version 1.0.6, URL https://CRAN.R-project.org/package=BigVAR.

Novomestky F (2012). ***matrixcalc**: Collection of Functions for Matrix Calculations*. R package version 1.0-3, URL https://CRAN.R-project.org/package=matrixcalc.

Pfaff B (2008a). *Analysis of Integrated and Cointegrated Time Series with R*. 2nd edition. Springer-Verlag, New York. doi:10.1007/978-0-387-75967-8.

Pfaff B (2008b). "VAR, SVAR and SVEC Models: Implementation within R Package **vars**." *Journal of Statistical Software*, **27**(4), 1–32. doi:10.18637/jss.v027.i04.

Ravikumar P, Wainwright MJ, Lafferty JD (2010). "High-Dimensional Ising Model Selection Using L1-Regularized Logistic Regression." *The Annals of Statistics*, **38**(3), 1287–1319. doi:10.1214/09-aos691.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Schmittmann VD, Jahfari S, Borsboom D, Savi AO, Waldorp LJ (2015). "Making Large-Scale Networks from fMRI Data." *PloS ONE*, **10**(9), e0129074. doi:10.1371/journal.pone.0129074.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464. doi:10.1214/aos/1176344136.

Song L, Kolar M, Xing EP (2009). "KELLER: Estimating Time-Varying Interactions between Genes." *Bioinformatics*, **25**(12), i128–i136. doi:10.1093/bioinformatics/btp192.

Tao Q, Huang X, Wang S, Xi X, Li L (2016). "Multiple Gaussian Graphical Estimation with Jointly Sparse Penalty." *Signal Processing*, **128**, 88–97. doi:10.1016/j.sigpro.2016.03.009.

Trip DSL, Van Wieringen WN (2018). "A Parallel Algorithm for Penalized Learning of the Multivariate Exponential Family from Data of Mixed Types." arXiv:1812.02401 [stat.ME], URL https://arxiv.org/abs/1812.02401.

Van Borkulo C, Epskamp S (2016). ***IsingFit**: Fitting Ising Models Using the ELasso Method.* R package version 0.3.1, URL https://CRAN.R-project.org/package=IsingFit.

Van Borkulo CD, Borsboom D, Epskamp S, Blanken TF, Boschloo L, Schoevers RA, Waldorp LJ (2014). "A New Method for Constructing Networks from Binary Data." *Scientific Reports*, **4**(5918). doi:10.1038/srep05918.

Van Rossum G, *et al.* (2011). *Python Programming Language.* URL https://www.python.org/.

Wainwright MJ, Jordan MI (2008). "Graphical Models, Exponential Families, and Variational Inference." *Foundations and Trends® in Machine Learning*, **1**(1–2), 1–305. doi:10.1561/2200000001.

Wan YW, Allen GI, Baker Y, Yang E, Ravikumar P, Liu Z (2015). ***XMRF**: Markov Random Fields for High-Throughput Genetics Data.* R package version 1.0, URL https://CRAN.R-project.org/package=XMRF.

Warnes GR, Bolker B, Lumley T (2018). ***gtools**: Various R Programming Tools.* R package version 3.8.1, URL https://CRAN.R-project.org/package=gtools.

Wichers M, Groot PC, Psychosystems, ESM Group, EWS Group (2016). "Critical Slowing Down as a Personalized Early Warning Signal for Depression." *Psychotherapy and Psychosomatics*, **85**(2), 114–116. doi:10.1159/000441458.

Wickham H (2019). ***stringr**: Simple, Consistent Wrappers for Common String Operations.* R package version 1.4.0, URL https://CRAN.R-project.org/package=stringr.

Wild B, Eichler M, Friederich HC, Hartmann M, Zipfel S, Herzog W (2010). "A Graphical Vector Autoregressive Modelling Approach to the Analysis of Electronic Diary Data." *BMC Medical Research Methodology*, **10**(1), 28. doi:10.1186/1471-2288-10-28.

Yang E, Baker Y, Ravikumar P, Allen G, Liu Z (2014). "Mixed Graphical Models via Exponential Families." In *Artificial Intelligence and Statistics*, pp. 1042–1050.

Yang E, Ravikumar P, Allen GI, Liu Z (2015). "Graphical Models via Univariate Exponential Family Distributions." *Journal of Machine Learning Research*, **16**(1), 3813–3847.

Yang E, Ravikumar PK, Allen GI, Liu Z (2013). "On Poisson Graphical Models." In *Advances in Neural Information Processing Systems*, pp. 1718–1726.

Zhao T, Liu H, Roeder K, Lafferty J, Wasserman L (2012). "The Huge Package for High-Dimensional Undirected Graph Estimation in R." *Journal of Machine Learning Research*, **13**(1), 1059–1062.

Zhou S, Lafferty J, Wasserman L (2010). "Time Varying Undirected Graphs." *Machine Learning*, **80**(2–3), 295–319. doi:10.1007/s10994-010-5180-0.

Zou H, Hastie T (2005). "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society B*, **67**(2), 301–320. doi:10.1111/j.1467-9868.2005.00503.x.

Affiliation:

Jonas M. B. Haslbeck
Psychological Methods
Nieuwe Achtergracht 129-B
Postbus 15906
1018 WT, Amsterdam, The Netherlands
E-Mail: jonashaslbeck@gmail.com
Website: http://www.jonashaslbeck.com/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
April 2020, Volume 93, Issue 8 *Submitted:* 2016-04-27
doi:10.18637/jss.v093.i08 *Accepted:* 2019-01-16