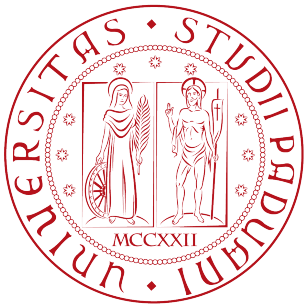
Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Magistrale in Scienze Statistiche



Copula VAR models with applications to genetic networks

Relatore:
Prof.ssa Alessandra Brazzale

Tutor all'estero:
Prof. Ernst Wit

Studentessa:
Lucia Benetazzo

Matricola N.
1105767

Anno Accademico 2016/2017

Contents

List of Figures

Introduction

Collecting observations of gene expression levels over time has become cheaper and faster in the last decades, due to significant developments in DNA microarray technology. Time-series data provide a deeper insight on the biological process under study. A time-course experiment implies that each sample is followed over time and observations are taken at different time steps. It is of interest to unravel the dependencies and the relationships among genes, both on a contemporary basis and the time-delayed ones. This can be achieved with a gene regulatory network (GRN), a visual instrument that explains in an intuitive way the gene interactions. GRNs can be useful to biologists to understand the underlying biological process. Sima et al. (2009) present a survey of study on models for GRNs for time series data.

There are two major approaches to the study of networks, according to whether observations are on the *links* or on the *vertices* of the network. The former, *network models* (e.g., Exponential Random Graph Models) are usually applied to social networks, for instance. The latter approach, which will be used in this thesis, can be referred to as *graphical models*. A thorough overview on graphical models can be found in Lauritzen (1996).

Most of the graphical models literature focuses on Gaussian Graphical Models (GGM), where the observed data are assumed to be normally distributed. The Time Series Chain Graphical Model (TSCGM) by Abegaz and Wit (2013), which extends GGMs to the time series framework, imposes the same assumption. GGMs for time series data have also been tackled by Dahlhaus and Eichler (2003) and Gao and Tian (2010). However, often the normal assumption is coarse as gene expression levels are usually not normally distributed. This work can be considered as an extension of the TSCGM, since we aim to tackle the issue of non-normality using the Gaussian copula. Our approach only imposes a multivariate normal assumption for a set of latent variables which are in a one-to-one

correspondence with the set of observed variables. The relationship between the latent and the observed process is the Gaussian copula. The latter allows to model multivariate associations and dependencies separately from the univariate marginal distributions of the observed variables. Nelsen (2007) covers all fundamental aspects of copulas. The Gaussian copula can be thought of as a transformation of the observed variables, which involves the unknown marginal distributions. We propose various nonparametric methods to estimate the unknown distributions. In this way, we are able to retrieve the latent Gaussian process without making any assumption on the distribution of the observed variables, except that they have to be continuous.

We use a likelihood approach, assuming that the latent process follows a Markovian dynamics, which can be translated into a vector autoregressive process of order 1 (VAR(1)). However, the method can be easily extended to account for higher order processes. The aim is to estimate the autoregressive matrix and the variance covariance matrix, which provide all the information to create the graphs for delayed and contemporaneous interactions among the variables, respectively.

GRNs are known to be sparse: this implies that the true graph will have very few edges compared to the full graph. To account for this, we add a *lasso* penalty to the likelihood in order to obtain sparse estimates. The estimation is accomplished via a two-stage iterative procedure taking the idea from the Multivariate Regression with Covariance Estimation (MRCE) method by Rothman et al. (2010). The regularization parameters of the lasso penalty have to be tuned: model selection criteria like AIC, BIC or extended BIC are used to that purpose.

We applied our method to two datasets of gene expression levels from:

- the plant *Arabidopsis thaliana*. The goal is to understand the interactions among genes whose expressions are related to the day/night cycle of the plant;
- the bacteria *Neisseria gonorrhoeae*, aiming to recover the dependence structure of the genes involved in the transcriptional repressor FarR.

The structure of the thesis is as follows. In Chapter 1 we give an overview of the models and methods on which we refer to develop the proposed model, such as graphical models, TSCGM, Gaussian copula and lasso. Chapter 2 is devoted to describing the theory of the proposed Copula VAR Model. In Chapter 3, inference and estimation methods can

be found. Chapter 4 regards the model selection criteria. In Chapter 5, simulations and applications to real data are presented. In the appendix, the **R** code is provided.

Chapter 1

Background

1.1 Graphical models

A graph is a pair $G = (V, E)$, where V is a finite set of *vertices* and the set of *edges* E is a subset of the set $V \times V$ of ordered pairs of distinct vertices. Edges $(\alpha, \beta) \in E$ with both (α, β) and (β, α) in E are called *undirected*, whereas an edge (α, β) with its *opposite* (β, α) not in E is called *directed*. If the graph has only undirected edges it is an *undirected* graph and if all edges are directed, the graph is said to be *directed*. A basic feature of the notion of a graph is that it is a visual object. It is conveniently represented by a picture, where a dot is used for a vertex and a line or an arrow between vertices is used for an undirected or directed edge, respectively (Lauritzen, 1996).
We will connect the graph syntax to describe conditional independence relationships for a random vector $\boldsymbol{X}$.
Formally, if X, Y, Z are continuous random variables which admit a joint distribution, we say that X *is conditionally independent of* Y *given* Z, and write $X \perp\!\!\!\perp Y \mid Z$, if

$$f_{XY \mid Z}(x, y \mid z) = f_{X \mid Z}(x \mid z) f_{Y \mid Z}(y \mid z).$$

A graph G is called a *conditional independence graph* with respect to $\boldsymbol{X}$, if

- the components of $\boldsymbol{X}$ can be associated with the vertices of V, i.e. $\boldsymbol{X} = (X_\alpha)_{\alpha \in V}$;
- $X_\alpha \perp\!\!\!\perp X_\beta \mid V \setminus \{\alpha, \beta\} \Leftrightarrow (\alpha, \beta) \notin E$: the absence of an edge between X_α and X_β corresponds with the conditional independence of these two random variables given

the remaining variables.

The latter is referred to as the *pairwise Markov property.*

1.2 Time series chain graphical models

Dahlhaus and Eichler (2003) and Abegaz and Wit (2013) introduced a Time Series Chain Graphical Model (TSCGM) to model dynamic interactions among variables.
Suppose that we have longitudinal data, for example microarray data, in which n replications indexed by $i = 1, \ldots, n$ of continuous measurements across p genes indexed by $j = 1, \ldots, p$ are repeated T times. That is, x_{ijt} is the j-th gene expression level at time t for the i-th replicate; it represents the i-th realization of the univariate random variable X_{jt}. The vector $\mathbf{x}_{it} = (x_{i1t}, \ldots, x_{ipt})$ contains the observations of the p gene expression levels for the i-th individual at time t. Moreover, $\mathbf{x}_{it}$ is a realization of the p-variate random variable $\mathbf{X}_t$.

The aim is to model both contemporaneous and dynamic (delayed) relationships among the genes. This is accomplished with a time series chain graph $G = (V, E)$, where V is a finite set of *vertices* and the set of *edges* E is a subset of the set $V \times V$ of ordered pairs of distinct vertices. The time series chain graph is based on the partitioning of V into a number of blocks $\bigcup_{t=1}^{T} V_t$. In this context, interpretation of links differs between within and across time steps. Links within a time step, V_t, are undirected as in an ordinary graphical model and represent the contemporaneous interactions among genes. Links across time steps are directed and point from the previous, V_{t-1}, to the current time step, V_t; they represent dynamic or delayed interactions between genes in time (Abegaz and Wit, 2013). Those delayed interactions follow the Markov property: the vector of gene expressions at time t only depends on that of time $t-1$, although extension to a Markov property of order $d \geq 2$ is straightforward. Let $\mathbf{X}_t = (X_{1t}, \ldots X_{pt})$ be a p-variate random variable associated to the nodes in V_t. Hence, the joint probability density of $\mathbf{X}_1, \ldots, \mathbf{X}_T$ can be decomposed as

$$f(\mathbf{X}_1, \ldots, \mathbf{X}_T) = f(\mathbf{X}_1) f(\mathbf{X}_2 \mid \mathbf{X}_1) \times \cdots \times f(\mathbf{X}_T \mid \mathbf{X}_{T-1}).$$

It is assumed that $\mathbf{X}_t$ follows a Markovian dynamics, which can be translated into a vector

autoregressive process of order 1 (VAR(1)):

$$\mathbf{X}_t = \Delta \mathbf{X}_{t-1} + \boldsymbol{\epsilon}_t. \tag{1.1}$$

In the Gaussian TSCGM setting, the following normality assumption is made:

$$\mathbf{X}_t \mid \mathbf{X}_{t-1} \sim \mathcal{N}_p(\Delta \mathbf{X}_{t-1}, \Omega^{-1}). \tag{1.2}$$

According to Dahlhaus and Eichler (2003), the matrices Δ and Ω contain all the information about the dependencies among the variables. In particular, given the time series chain graph $G = (V, E)$ where E includes both direct and undirected edges, the directed edges in the graph reflect the recursive structure of the time series, summarized in Δ:

$$(\alpha, \beta) \in V_{t-1} \times V_t \Leftrightarrow \Delta_{\alpha\beta} \neq 0,$$

i.e. non-zero entries in Δ are equivalent to directed edges. Likewise, contemporaneous interactions (undirected edges) are related to non-zero entries of the concentration matrix, also known as precision matrix, Ω:

$$(\alpha, \beta) \in V_t \times V_t \Leftrightarrow \Omega_{\alpha\beta} \neq 0.$$

However, in this thesis we want to tackle the issue of non normality: our aim is to make no assumption about the distribution of gene expressions. Particularly, in order to relax the Gaussian assumption on $\mathbf{X}_t \mid \mathbf{X}_{t-1}$, we will exploit the *Gaussian copula*.

1.3 The Gaussian copula

Copulas are functions that join or "couple" multivariate distribution functions to their one-dimensional marginal distribution functions. Alternatively, copulas are multivariate distribution functions whose one-dimensional margins are uniform on the interval $(0, 1)$ (Nelsen, 2007).
Consider a collection of random variables $(X_1, \ldots, X_p)$ with marginal Cumulative Distribution Functions (CDFs) $F_j(x_j) = \mathbb{P}[X_j \leq x_j]$, for $j = 1, \ldots, p$: let us suppose that those marginal CDFs are continuous. Consider also the joint distribution function $F(x_1, \ldots, x_p) =$

$\mathbb{P}[X_1 \leq x_1, \ldots, X_p \leq x_p]$. By applying the CDF to each component, the random vector

$$(U_1, \ldots, U_p) = (F_1(X_1), \ldots, F_p(X_p))$$

has uniformly distributed marginals in $(0, 1)$. The copula of $(X_1, \ldots, X_p)$ is defined as the joint cumulative distribution function of $(U_1, \ldots, U_p)$:

$$C(u_1, \ldots, u_p) = \mathbb{P}[U_1 \leq u_1, \ldots, U_p \leq u_p]. \tag{1.3}$$

In other words, each ordered vector $(x_1, \ldots, x_p)$ of real numbers leads to a point $(F_1(x_1), \ldots, F_p(x_p))$ in $(0, 1)^p$, and this ordered vector in turn corresponds to a number $F(x_1, \ldots, x_p)$ in $(0, 1)$. This correspondence, which assigns the value of the joint distribution function to each ordered vector of values of the individual distribution functions, is indeed a function: such functions are copulas (Nelsen, 2007).
The copula C contains all information on the dependence structure between the components of $(X_1, \ldots, X_p)$ while the marginal cumulative distribution functions F_j contain all information on the marginal distributions. The formula in (1.3) for the copula function can be rewritten as follows:

$$C(u_1, \ldots, u_p) = \mathbb{P}[X_1 \leq F_1^{-1}(u_1), \ldots, X_p \leq F_p^{-1}(u_p)].$$

Sklar's theorem (Sklar, 1959) elucidates the role that copulas play in the relationship between multivariate distribution functions and their univariate margins. It provides the theoretical foundation for the application of copulas and justifies the importance of copulas in modeling the distribution of multivariate random variables. Sklar's theorem states that every multivariate cumulative distribution function $F(x_1, \ldots, x_p) = \mathbb{P}[X_1 \leq x_1, \ldots, X_p \leq x_p]$ of a random vector $(X_1, \ldots, X_p)$ can be expressed in terms of its marginals $F_j(x) = \mathbb{P}[X_j \leq x]$ and a copula C as follows:

$$F(x_1, \ldots, x_p) = C\left(F_1(x_1), \ldots, F_p(x_p)\right).$$

If $F_1, \ldots, F_p$ are all continuous (as in our case), then C is unique.
The information in the joint distribution is decomposed into those in the marginal distributions and that in the copula function, where the latter captures the dependence structure between the variables.

Given correlation matrix P, we define the *Gaussian copula* as follows:

$$F(x_1, \ldots, x_p) = \Phi_P \left(\Phi^{-1}(F_1(x_1)), \ldots, \Phi^{-1}(F_1(x_p)) \right),$$

where Φ_P is the p-variate Gaussian CDF with mean $\mathbf{0}_p$ and covariance matrix P and Φ^{-1} is the univariate standard Gaussian quantile function. Thus, the corresponding copula is

$$C(u_1, \ldots, u_p) = \Phi_P \left(\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_p) \right).$$

Copulas are popular in high-dimensional statistical applications as they provide the theoretical framework in which multivariate associations and dependencies can be modeled separately from the univariate distributions of the observed variables (Dobra and Lenkoski, 2009).

1.4 Penalized inference and lasso

Genetic networks are known to be sparsely connected. In order to account for this, a penalty function can be subtracted to the the objective function to maximize (the likelihood, in our case). One can think of the penalization term as

$$p_{\lambda,q}(\boldsymbol{\beta}) = \lambda \sum_{k=1}^{p} |\beta_k|^q, \tag{1.4}$$

where $\boldsymbol{\beta} = (\beta_1, \ldots \beta_p)$ are the parameters under consideration. $q = 1$ corresponds to shrinking the coefficients by their l_1 norm; this method is called *lasso*. $q = 2$ results in the l_2 norm, and the method is known as *ridge*. The value $q = 0$ coincides with the variable subset selection in the context of regression, as the penalty simply counts the number of non-zero parameters.

The choice of the shrinkage method should reflect the need to set some coefficient exactly at zero. The lasso has this special property: constraining the coefficients by their l_1 norm induces sparsity in the estimates. This is not true with the ridge penalty, for instance (Tibshirani, 2015). In fact, with $q > 1$, $|\beta_k|^q$ is differentiable at zero, and so does not share the ability of lasso for setting coefficients exactly at zero (i.e., it does not provide sparse estimates). On the other hand, the case $q = 1$ is the smallest q such that the constraint region is convex: nonconvex constraint regions, given by $q < 1$, make the optimization

problem computationally difficult (Friedman et al., 2001). For this reasons we choose the lasso penalty, as it is a good compromise between sparsity and computational feasibility (being a convex problem).
Note that the shrinkage methods as the ones mentioned above reduce the variance of our estimate, at the expense of (likely) introducing bias.
The lasso penalty is not the only possibility. In the context of convex optimization, the following *elastic-net* penalty introduced by Zou and Hastie (2005) might also be used:

$$p_\lambda^{\text{EN}}(\boldsymbol{\beta}) = \lambda \sum_{k=1}^{p} (\alpha \beta_k^2 + (1-\alpha)|\beta_k|),$$

which is a compromise between lasso and ridge. The parameter α determines the mix of the penalties. Elastic-net preserves the sparsity property of lasso.
Shifting to nonconvex problems, one can modify the lasso penalty function so that larger coefficients are not excessively penalized. In fact, the lasso penalty increases linearly in the magnitude of its argument: as a result, it produces substantial biases in the estimates for large coefficients. The *Smoothly Clipped Absolute Deviation* (*SCAD*) penalty was introduced by Fan and Li (2001) to address this issue. In SCAD, the penalization term $p_\lambda(\boldsymbol{\beta})$ in (1.4) is replaced by $\sum_{k=1}^{p} p_\lambda^{\text{SCAD}}(\beta_k)$, where

$$p_\lambda^{\text{SCAD}}(\beta_k) = \begin{cases} \lambda|\beta_k| & \text{if } |\beta_k| \leq \lambda, \\ -\left[\frac{|\beta_k|^2 - 2a\lambda|\beta_k| + \lambda^2}{2(a-1)}\right] & \text{if } \lambda \leq |\beta_k| \leq a\lambda, \\ \frac{(a+1)\lambda^2}{2} & \text{if } |\beta_j| \leq a\lambda. \end{cases}$$

SCAD has been largely used it the context of genetic networks estimation: see, for instance, Fan et al. (2009) and Abegaz and Wit (2013). For further in-depth analysis of nonconvex penalty functions, see Breheny and Huang (2011) and Loh and Wainwright (2013).

Many algorithms are available to generate the whole path of lasso solutions.
The *Least Angle Regression* (LARS), introduced by Efron et al. (2004), is a computationally efficient algorithm for fitting linear regression by successive orthogonalization. An interesting property is that a simple modification of the LARS algorithm implements the entire sequence path of lasso, maintaining the same order of magnitude of computations as the least squares fit (Azzalini and Scarpa, 2012).
Another interesting algorithm is the *graphical lasso* (glasso) introduced by Friedman et al.

(2008), which was designed to estimate sparse graphs by a lasso penalty applied to the inverse covariance matrix. To the same purpose, Hsieh et al. (2014) introduced the *QUadratic approximation for sparse Inverse Covariance estimation* (QUIC) algorithm. The latter is based on Newton's method and employs a quadratic approximation. In their paper, Hsieh et al. (2014) demonstrate the considerable improvements in performance compared to alternative methods like glasso: therefore, in section (3.3), we will use QUIC to estimate the precision matrix.

Chapter 2

Copula VAR Model (CVM)

Going back to section (1.2), let us consider the longitudinal data $\mathbf{x}_{it} = (x_{i1t}, \ldots, x_{ipt})$, and the corresponding p-variate random variables $\mathbf{X}_t = (X_{1t}, \ldots, X_{pt})$, for $i = 1, \ldots, n$, $t = 1, \ldots, T$.
Our aim is to extend the TSCGM, not making any assumption on the distribution of the observed variables $\mathbf{X}_t$: therefore, the assumptions in (1.1) and (1.2) do not hold anymore.

Instead, we assume that the multivariate dependence pattern among the variables $X_{1t}, \ldots, X_{pt}$ is given by the Gaussian copula with covariance matrix Σ_t with dimensions $p \times p$:

$$F(X_{1t}, \ldots, X_{pt}) = \Phi_{\Sigma_t}\left(\Phi^{-1}(F_{1t}(X_{1t})), \ldots, \Phi^{-1}(F_{pt}(X_{pt})\right), \tag{2.1}$$

where:

- Φ_{Σ_t} is the CDF of a multivariate Gaussian distribution with covariance matrix Σ_t, $\mathcal{N}_p(\mathbf{0}, \Sigma_t)$;
- $\Sigma_t = \Gamma^{t-1}\Theta^{-1}(\Gamma^{t-1})^\top + \Gamma^{t-2}\Theta^{-1}(\Gamma^{t-2})^\top + \cdots + \Theta^{-1}$, where Θ is positive definite and Γ any matrix, both of them with dimension $p \times p$;
- Φ^{-1} is the quantile function of the univariate standard normal $\mathcal{N}(0, 1)$;
- F_{jt} is the marginal CDF of gene j at time t, $\forall\ j = 1, \ldots, p$, $t = 1, \ldots, T$.

The Gaussian copula model can also be constructed by introducing a p-variate latent variable $\mathbf{Z}_t = (Z_{1t}, \ldots, Z_{pt})$. As in Dobra and Lenkoski (2009), let us also consider the

p-variate variable $\mathbf{Z}_t^* = (Z_{1t}^*, \ldots, Z_{pt}^*)$ which is nothing but the unscaled version of $\mathbf{Z}_t$. $\mathbf{Z}_t$ and $\mathbf{Z}_t^*$ follow a dynamics similar to that in (1.1):

$$\mathbf{Z}_t^* = \Gamma \mathbf{Z}_{t-1} + \boldsymbol{\eta}_t, \quad \text{where } \boldsymbol{\eta}_t \sim \mathcal{N}_p(\mathbf{0}, \Theta^{-1}). \tag{2.2}$$

Given a covariance matrix Σ_t, the marginal distribution of $\mathbf{Z}_t^*$ is

$$\mathbf{Z}_t^* \sim \mathcal{N}_p(\mathbf{0}, \Sigma_t).$$

For example, if $\mathbf{Z}_1 \sim \mathcal{N}_p(\mathbf{0}, \Theta^{-1})$, then $\mathbf{Z}_2^* \sim \mathcal{N}_p(\mathbf{0}, \Theta^{-1} + \Gamma\Theta^{-1}\Gamma^\top)$.
We consider the unit variance scaling of $\mathbf{Z}_t^*$:

$$Z_{jt} = \frac{Z_{jt}^*}{\sqrt{\sigma_{jt}}}, \quad \text{for } j = 1, \ldots, p, \quad t = 1, \ldots, T,$$

where σ_{jt} is the j-th diagonal element of Σ_t. As a consequence, the marginal distribution of the latent variables Z_{jt} is $\mathcal{N}(0, 1)$, $\forall\ j = 1, \ldots, p$, $t = 1, \ldots, T$.
The latent variables in $\mathbf{Z}_t$ are related to the observed variables $\mathbf{X}_t$ as the following one-to-one transformation:

$$\begin{aligned} \mathbf{Z}_t &= g(\mathbf{X}_t) = (g_1(X_{1t}), \ldots, g_p(X_{pt})) \\ Z_{jt} &= g_j(X_{jt}) = \Phi^{-1}(F_{jt}(X_{jt})), \quad \text{for } j = 1, \ldots, p, \quad t = 1, \ldots, T, \end{aligned} \tag{2.3}$$

where X_{jt} is the variable which represents the gene expression level for gene j at time t.
To sum up, we consider what follows:

$$\begin{cases} \mathbf{Z}_t^* &= \Gamma \mathbf{Z}_{t-1} + \boldsymbol{\eta}_t, \quad \boldsymbol{\eta}_t \sim \mathcal{N}_p(\mathbf{0}, \Theta^{-1}) \\ Z_{jt} &= \frac{Z_{jt}^*}{\sqrt{\sigma_{jt}}}, \qquad\qquad \sigma_{jt} = Var(Z_{jt}^*) \\ X_{jt} &= F_{jt}^{-1}(\Phi(Z_{jt})). \end{cases}$$

for $j = 1 \ldots, p$, $t = 1, \ldots, T$. Figure (2.1) clarifies this pattern.

It is possible to define the model in alternative ways, which avoid the explicit introduction of $\mathbf{Z}_t^*$:

- make the Γ and Θ time-dependent, accounting for the appropriate rescaling;

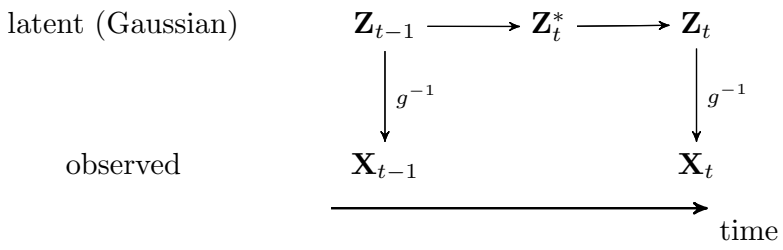


Figure 2.1: Diagram which represents the relationship between the observed process and the latent normal process through the *Gaussian copula* transformation g.

- make the copula transformation time-dependent: this can be accomplished by considering $\Phi^{-1}_{\sigma_{jt}}$, the CDF of a univariate normal distribution with mean 0 and variance equal to σ_{jt}, instead of Φ^{-1} in (2.1).

Our aim is to infer the graphical structure defined by the latent process $\mathbf{Z}_t$. This structure is contained in the non-zero pattern of the matrices Θ and Γ. In particular:

- non-zero entries of Θ represent contemporaneous interactions and will be represented by undirected links *within* a time step;
- non-zero entries of Γ correspond to direct links *between* time steps and explain dynamic interactions between genes across time.

Figure (2.2) shows an example of this structure.

2.1 Full Likelihood

The joint probability density function of $\mathbf{X}_t \mid \mathbf{X}_{t-1}$ is given by

$$f(\mathbf{X}_t|\mathbf{X}_{t-1},\Gamma,\Theta) = (2\pi)^{-p/2}|\Theta|^{1/2}\exp\left\{-\frac{1}{2}\Big[g(\mathbf{X}_t)-\Gamma g(\mathbf{X}_{t-1})\Big]^\top\Theta\Big[g(\mathbf{X}_t)-\Gamma g(\mathbf{X}_{t-1})\Big]\right\}\prod_{j=1}^{p} d_{jt}.$$

In this formulation, $d_{jt} = \Big|\phi^{-1}\big(F_{jt}(X_{jt})\big)f_{jt}(X_{jt})\Big|$ is the absolute value of the derivative of the transformation (2.3) where:

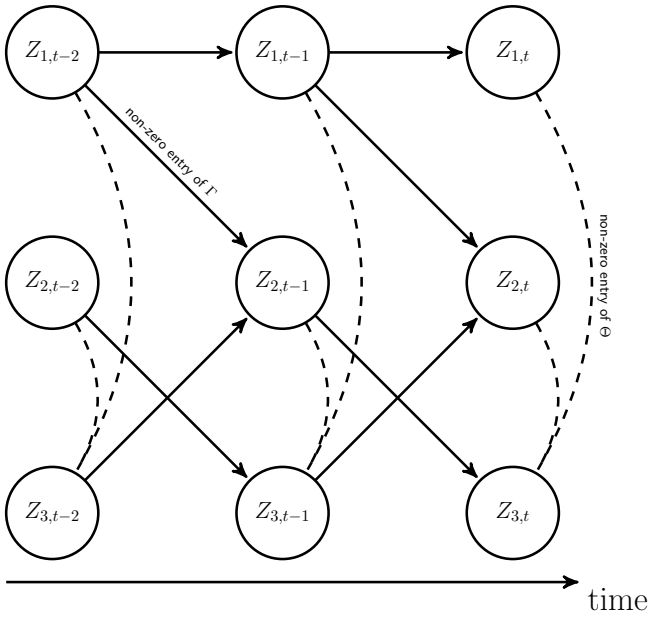


Figure 2.2: Example of graphical structure of a Copula VAR model of order 1 for $p = 3$ variables (genes) and $T = 3$ time points. Directed edges represent non-zero entries of Γ and undirected edges (dashed links) characterize the non-zero entries of Θ.

- ϕ^{-1} is the quantile density function of the univariate normal distribution $\mathcal{N}(0, 1)$. The explicit formula can be obtained by implicit differentiation:

$$\phi^{-1}(x) = \frac{d}{dx}\Phi^{-1}(x) = \frac{1}{\phi\big(\Phi^{-1}(x)\big)},$$

  where ϕ is the standard normal density function;

- F_{jt} is the unknown marginal CDF of gene j at time t;

- f_{jt} is the unknown marginal density function of gene j at time t.

The log-likelihood for n replicates each at T time steps is defined as

$$\ell(\Gamma, \Theta) = -\frac{npT}{2}\log(2\pi) + \frac{nT}{2}\log\big|\Theta\big| - \frac{nT}{2}\operatorname{Tr}(S_\Gamma\Theta) + D, \tag{2.4}$$

where:

$$
\begin{aligned}
D &= \sum_{i=1}^{n}\sum_{j=1}^{p}\sum_{t=1}^{T} \log\Big|\phi^{-1}\big(F_{jt}(x_{ijt})\big) f_{jt}(x_{ijt})\Big|, \\
S_\Gamma &= \frac{1}{nT}\sum_{i=1}^{n}\sum_{t=1}^{T} (g(\mathbf{x}_{it}) - \Gamma g(\mathbf{x}_{i,t-1}))(g(\mathbf{x}_{it}) - \Gamma g(\mathbf{x}_{i,t-1}))^\top \\
&= C_x - C_{xx\_}\Gamma^\top - \Gamma C_{xx\_}^\top + \Gamma C_{x\_}\Gamma^\top, \\
C_x &= \frac{1}{nT}\sum_{i=1}^{n}\sum_{t=1}^{T} g(\mathbf{x}_{it}) g(\mathbf{x}_{it})^\top, \\
C_{xx\_} &= \frac{1}{nT}\sum_{i=1}^{n}\sum_{t=1}^{T} g(\mathbf{x}_{it}) g(\mathbf{x}_{i,t-1})^\top, \\
C_{x\_} &= \frac{1}{nT}\sum_{i=1}^{n}\sum_{t=1}^{T} g(\mathbf{x}_{i,t-1}) g(\mathbf{x}_{i,t-1})^\top.
\end{aligned}
$$

2.2 Pseudo-likelihood

Following the idea from Genest et al. (1995), to estimate the parameter matrices Γ and Θ, two approaches could be contemplated.

If valid parametric distributions are already available for the marginals f_{jt}, then it is straightforward in principle to maximize the likelihood in (2.4). The resulting estimate for the parameters would then be margin-dependent, just as the estimates of the parameters involved in the marginal distributions would be indirectly affected by the copula.

However, we aim not to make any assumption about the marginal distributions of the genes. To do so, we will contemplate nonparametric estimates for the marginals CDFs. We reduce our model parameters to the correlation and autoregressive matrices of the latent process: inference about those parameters will be margin-free. This means that we focus on the joint distribution of the latent variables $\mathbf{Z}_t$, $\forall\, t = 1, \ldots, T$, whose relationships with the observed variables $\mathbf{X}_t$ are given by (2.3). Our semiparametric estimation strategy is to plug-in a nonparametric estimation of the CDFs F_{jt} in (2.3) in order to obtain pseudo-data which will be used to estimate the matrices Γ and Θ separately from the marginal distributions.

Practically, this allows us to neglect the term D in (2.4) since, in the pseudo-likelihood

framework, D is an additive term which does not involve the parameters.
In conclusion, the pseudo-likelihood which will be considered from now on is the following:

$$\ell^{\mathrm{PS}}(\Gamma, \Theta) = -\frac{npT}{2}\log(2\pi) + \frac{nT}{2}\log\left|\Theta\right| - \frac{nT}{2}\operatorname{Tr}(S_\Gamma \Theta). \tag{2.5}$$

Chapter 3

Inference of Copula VAR Model

3.1 Nonparametric CDF estimation

In order not to make any assumption about the marginal distributions of the observed gene expression levels, the CDFs F_{jt} necessary to create the pseudo-data in (2.3) will be estimated in a nonparametric fashion.
Let $X_1, \ldots, X_n$ be a sequence of independent and identically distributed (i.i.d.) random variables with common CDF F. Then the empirical distribution function F_n is defined as

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{X_i \leq x\} \tag{3.1}$$

where $\mathbb{1}$ is the indicator function, namely $\mathbb{1}\{X_i \leq x\}$ is 1 if $X_i \leq x$ and 0 otherwise. For i.i.d. variables, the Glivenko-Cantelli theorem states that F_n converges uniformly to the real CDF F.
Liu et al. (2009) and Liu et al. (2012) use a truncated form of the ECDF, introducing an additional truncation parameter to be tuned. Chen and Fan (2006), in their copula-based semiparametric time series models, use a rescaled version of the empirical distribution function.
One obvious drawback of F_n is that even when the real F is continuous, F_n is not: the latter is a step function, with flat plateaus. Further smoothing of the CDF F estimate can be an advantage.
In this section, we propose various nonparametric method to estimate the CDFs.

3.1.1 Modified ECDF

Let us assume that the CDFs F_{jt} are constant across time, i.e. $F_{jt} = F_j$. We will use the following modified version of the ECDF, which avoids extreme values maintaining the good properties of the ECDF:

$$F_j^e(x) = \frac{1}{nT+1}\left(\sum_{t=1}^{T}\sum_{i=1}^{n}\mathbb{1}\{x_{ijt} \leq x\} + \frac{1}{2}\right), \quad \text{for } j = 1, \ldots, p. \tag{3.2}$$

3.1.2 Kernel density estimation

Kernel density estimation is a nonparametric method to estimate the density function of a random variable: it was first introduced by Rosenblatt (1956) and Parzen (1962). Following the definition by Azzalini (1981), for i.i.d. variables $X_1, \ldots, X_n$, the kernel estimator of the density f at a given point x is

$$\tilde{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} w\left(\frac{x - X_i}{h}\right) \tag{3.3}$$

where w is some bounded density function called *kernel*. w has to be symmetric, namely $w(k) = w(-k)$. The smoothing parameter h is known as the *bandwidth* and has to be carefully chosen.
The corresponding estimate for the CDF F is

$$\tilde{F}_h(x) = \frac{1}{n}\sum_{i=1}^{n} W\left(\frac{x - X_i}{h}\right)$$

where

$$W(k) = \int_{-\infty}^{k} w(u)du.$$

The choice of the kernel is not as crucial as the bandwidth. A popular choice for w is the following Gaussian kernel:

$$w(k) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{k^2}{2}\right)$$

which is the density function of the standard normal distribution.
Assuming that the CDFs are constant across time, we will use in our longitudinal framework

the following estimate:

$$\tilde{F}_j(x) = \frac{1}{nT} \sum_{t=1}^{T} \sum_{i=1}^{n} W\left(\frac{x - x_{ijt}}{h_j}\right). \tag{3.4}$$

We will use the **R** package `ks` which provides a function to estimate the CDF, `pkde`. According to the results of the simulations, the default values for the bandwidth are satisfying.

3.1.3 Dynamic kernel density estimation

We would like to take into account that the density might change over time. In order to estimate a time varying density, Harvey and Oryshchenko (2012) introduced a weighting scheme into the kernel estimator in (3.3). The set of weights that they use takes the form of an exponentially weighted moving average (EWMA). We will use the same idea, with a different weighting scheme. In our specification, weights exponentially decay to zero as the time interval becomes larger: i.e., in order to estimate the CDF in a time point for a gene, we use observations from the present, from the future and from the past for the same gene giving more importance to those observations closer in time. Hence, the CDF estimate for the j-th gene at time t is

$$\hat{F}_{jt}(x_{ijt}) = \sum_{l=1}^{T} \sum_{k=1}^{n} \Phi\left(\frac{x_{ijt} - x_{kjl}}{h_j}\right) w_{tl}, \tag{3.5}$$

where:

- $\sum_{l=1}^{T} \sum_{k=1}^{n} w_{tl} = 1$;
- $w_{tl} = \frac{1}{n} \frac{\omega^{|t-l|}}{\sum_{j=1}^{T} \omega^{|t-j|}}$, where ω is a constant which determines the decay velocity. The latter is a parameter which should be tuned: however, for the sake of simplicity, we choose $\omega = 0.925$;
- Φ is the CDF of the standard normal (which is the chosen kernel);
- h_j is the bandwidth parameter (different for each gene).

3.1.4 Bandwidth selection

In order to choose the bandwidth, a criterion to be optimized has to be chosen.
In the second chapter of his book, Scott (2015) discusses error criteria for density estimates:

what follows is a brief summary of that chapter.
The kernel density estimator is known to be *biased*, namely

$$Bias[\tilde{f}_h(x)] = E[\tilde{f}_h(x)] - f(x) \neq 0,$$

where $E[\cdot]$ denotes the expected value. When approximating parameters with biased estimators, the mean squared error (MSE) criterion is usually adopted:

$$\text{MSE}(x, h) = E[\tilde{f}_h(x) - f(x)]^2 = Var[\tilde{f}_h(x)] + Bias^2[\tilde{f}_h(x)]$$

The application in practice is difficult, since the formula depends on the unknown density function f both in the variance and the squared bias term. In addition, the MSE measures the squared deviation of the estimate $\tilde{f}_h(x)$ from the true $f(x)$ at a single point x. If we are interested in how well we estimate the entire density surface, we might use global measures of closeness of the estimate to the true curve f.
The integrated standard error ISE is a global discrepancy measure:

$$\text{ISE}(h) = \int \left[\tilde{f}_h(x) - f(x)\right]^2 dx.$$

The latter is a complicated random variable that still depends on f. Furthermore, it is a function of the particular realization on n points, therefore different samples will produce different ISE values. Let's examine the average of the ISE over these realizations, the mean integrated standard error MISE:

$$\text{MISE}(h) = E[\text{ISE}(h)] = E\left\{\int \left[\tilde{f}_h(x) - f(x)\right]^2 dx\right\}$$

which is not a random variable.
Taking a Taylor's series expansion and omitting higher order terms leaves the asymptotic mean integrated squared error (AMISE). Scott (2015), in the sixth chapter, obtains the following formula of the AMISE for the estimator in (3.3):

$$\text{AMISE} = \frac{R(w)}{nh} + \frac{1}{4}\sigma_w^4 h^4 R(f'')$$

where w is the chosen kernel, $\sigma_w^2 = \int x^2 w(x)dx$ and $R(w)$ is the squared L_2 norm: $R(w) = \int w(x)^2 dx$. The optimal bandwidth minimizing this function is

$$h = \left[\frac{R(w)}{n\sigma_w^4 R(f'')}\right]^{\frac{1}{5}}. \tag{3.6}$$

The plug-in bandwidth in (3.6) is implemented in the **R** package `kedd`. This bandwidth is supposed to be used in the classical kernel density estimator in (3.3) and may not be appropriate in the weighted version in (3.5); however, for the sake of simplicity, we use it our framework.

Once the CDFs for each gene have been estimated, the values of the latent variable, or pseudo-data, can be estimated as follows:

$$z_{ijt} = \Phi^{-1}(\hat{F}_{jt}(x_{ijt})), \quad \text{for } i = 1, \ldots, n, \quad j = 1, \ldots, p, \quad t = 1, \ldots, T.$$

3.2 Penalized likelihood

Since both Θ and Γ are supposed to be sparse, two l_1 penalties are subtracted to the pseudo-likelihood in (2.5). Hence, the objective function for optimization is defined as follows:

$$\ell^{\text{PS}}_{\text{PEN}}(\Gamma, \Theta) = -\frac{npT}{2}\log(2\pi) + \frac{nT}{2}\log\left|\Theta\right| - \frac{nT}{2}\operatorname{Tr}(S_\Gamma \Theta) - \lambda_1 \sum_{i \neq j}^{p} |\theta_{ij}| - \lambda_2 \sum_{i \neq j}^{p} |\gamma_{ij}|, \tag{3.7}$$

where θ_{ij} and γ_{ij} are the entries of Θ and Γ. λ_1 and λ_2 are the *tuning parameters*: the bigger the value of the tuning parameters, the more shrinkage is applied to the estimated parameters. Note that the l_1 penalty is only applied to the off-diagonal elements of both Θ and Γ, following the idea from Yuan and Lin (2007). Penalizing the inverse of the variance coefficients is unnatural; it just results in shrinking the scale of the variables, which is not our aim. As regards the diagonal of Γ, we assume that it is biologically plausible that a gene influences itself across time, therefore we do not penalize the corresponding coefficients.

3.3 Joint sparse estimation of parameter matrices

The optimization problem that gives sparse estimates of Θ and Γ is the solution of

$$\left(\hat{\Theta}, \hat{\Gamma}\right) = \underset{\Theta,\Gamma}{\operatorname{argmax}}\left\{ \log\left|\Theta\right| - \operatorname{Tr}(S_\Gamma \Theta) - \lambda_1 \sum_{i \neq j}^{p} |\theta_{ij}| - \lambda_2 \sum_{i \neq j}^{p} |\gamma_{ij}| \right\}. \tag{3.8}$$

As in Abegaz and Wit (2013), the estimation is accomplished via a two stage iterative procedure. We adopt the algorithm by Rothman et al. (2010), called *Multivariate Regression with Covariance Estimation* (MRCE). The method assumes predictors are not random; however, the resulting formulas for the estimates are the same with random predictors, as in our case. Despite the optimization problem in (3.8) is not convex, solving for either Γ or Θ with the other fixed is convex.
Solving (3.8) for Θ with Γ fixed yields the following optimization problem:

$$\hat{\Theta}_\Gamma = \underset{\Theta}{\operatorname{argmax}}\left\{ \log\left|\Theta\right| - \operatorname{Tr}(S_\Gamma \Theta) - \lambda_1 \sum_{i \neq j}^{p} |\theta_{ij}| \right\} \quad \text{with } \Gamma \text{ fixed.} \tag{3.9}$$

The latter is exactly the problem of estimating sparse undirected graphical models through the use of lasso regularization applied to the inverse covariance matrix, glasso, considered by Friedman et al. (2008). The *QUadratic approximation for sparse Inverse Covariance estimation* (QUIC) algorithm by Hsieh et al. (2014) implemented in the **R** package `QUIC` is used to solve (3.9).
Solving (3.8) for Γ with Θ fixed yields the following optimization problem:

$$\hat{\Gamma}_\Theta = \underset{\Gamma}{\operatorname{argmax}}\left\{ - \operatorname{Tr}(S_\Gamma \Theta) - \lambda_1 \sum_{i \neq j}^{p} |\gamma_{ij}| \right\} \quad \text{with } \Theta \text{ fixed,} \tag{3.10}$$

which is convex if Θ is positive definite. Rothman et al. (2010) propose a solution computed using cyclical-coordinate descent analogous to that used for solving the single output lasso problem (Friedman et al., 2007). The algorithm is implemented in the function `rblasso` from the **R** package `MRCE`.
The algorithm that we use to jointly estimate sparse Γ and Θ iteratively is shown in algorithm 1.

Algorithm 1 Iterative estimate

Initialization of maximum number of iterations I_{max} and *threshold* to determine convergence
$\hat{\Gamma}_0 \leftarrow$ diagonal matrix with dimensions $p \times p$
$k \leftarrow 0$
repeat
    $S_{\hat{\Gamma}_k} \leftarrow C_x - C_{xx\_}\hat{\Gamma}_k^\top - \hat{\Gamma}_k C_{xx\_}^\top + \hat{\Gamma}_k C_{x\_}\hat{\Gamma}_k^\top$
    $k \leftarrow k + 1$
    if $k = 1$ **then**
        estimate $\hat{\Theta}_k$ with *QUIC*, $S_{\hat{\Gamma}_{k-1}}$ fixed
        force $\hat{\Theta}_k$ to be a correlation matrix
        $\hat{\Sigma}_k \leftarrow \hat{\Theta}_k^{-1}$
        estimate $\hat{\Gamma}_k$ with *rblasso*, $\hat{\Theta}_k$ fixed
    if $k > 1$ **then**
        estimate $\hat{\Theta}_k$ with *QUIC* using $\hat{\Theta}_{k-1}$ as a "warm" start, $S_{\hat{\Gamma}_{k-1}}$ fixed
        force $\hat{\Theta}_k$ to be a correlation matrix
        $\hat{\Sigma}_k \leftarrow \hat{\Theta}_k^{-1}$
        estimate $\hat{\Gamma}_k$ with *rblasso* using $\hat{\Gamma}_{k-1}$ as initial value, $\hat{\Theta}_k$ fixed
    if $\hat{\Gamma}_k - \hat{\Gamma}_{k-1} <$ *threshold* **then** Convergence = true
until Convergence = true or $k > I_{\max}$

Chapter 4

Model Selection

The sparsity of the estimate is controlled by the tuning parameters λ_1 and λ_2: the latter have to be carefully chosen. This can be accomplished by choosing the couple of parameters $(\lambda_1^{\text{opt}}, \lambda_2^{\text{opt}})$ which optimize a model selection criterion.

First of all, a range of values for λ_1 and λ_2 to choose from has to be set. Friedman et al. (2010), in the lasso framework where just one parameter λ has to be tuned, suggest to choose a range $(\lambda^{\min}, \lambda^{\max})$ where $\lambda^{\max}$ is the smallest value for which the entire vector of estimated parameters are zero. The choice of $\lambda^{\min}$ and the number of grid points N is less rigorous: the minimum value for the tuning parameter is chosen as $\lambda^{\min} = \epsilon\lambda^{\max}$. Typical values are $\epsilon = 0.001$ and $N = 100$. The sequence of N values is constructed starting from $\lambda^{\max}$, decreasing to $\lambda^{\min}$ on the log scale.
Abegaz and Wit (2013) used the same strategy, modified to fit the longitudinal context with two tuning parameters, and implemented it in the package `sparseTCSGM`. We applied the latter strategy to the copula VAR model; however, simulation studies suggested that this method selects ranges of values for λ_1 and λ_2 too large, which lead to too sparse estimates. Therefore, in the simulations and applications we choose a range of parameters to choose from empirically. The choice was made so as to comprise the two extreme cases: we included values large enough to give completely sparse estimates (Γ and Θ diagonal), and small enough not to give sparse estimates. This information was retrieved from the lasso coefficients paths plots (see figure (5.1) in Chapter 5). We end up with two vectors of parameters to choose from with length $N = 50$: $(\lambda_1^{\min}, \ldots, \lambda_1^{\max})$ and $(\lambda_2^{\min}, \ldots, \lambda_2^{\max})$.

Secondly, a model selection criterion should be chosen. In the framework of model selection,

cross-validation is the gold standard. In order to perform K-fold cross-validation, one needs first of all to split the data into K parts. For the k-th part, one should fit the model to the other $k-1$ parts of the data, and calculate the loss function of the fitted model when fitting the k-th part of the data. This has to be done for $k = 1, \ldots, K$; then, the K loss functions have to be combined (Friedman et al., 2001). The case $K = n$ is called *leave-one-out* cross-validation. In our case, the deviance can be considered as the loss function. Let us denote with $\hat{\Gamma}_{\boldsymbol{\lambda}}^{-k}$ and $\hat{\Theta}_{\boldsymbol{\lambda}}^{-k}$ the estimated parameter matrices using the $k-1$ parts of the data, given a couple of tuning parameters $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$. $\ell^k(\hat{\Gamma}_{\boldsymbol{\lambda}}^{-k}, \hat{\Theta}_{\boldsymbol{\lambda}}^{-k})$ is the pseudo-likelihood in (2.5) calculated on the k-th part of the data and on the parameters estimated on the $k-1$ remaining parts of the data. We define the deviance for the k-th part as follows:

$$\mathcal{D}^k(\boldsymbol{\lambda}) = -2\ell^k(\hat{\Gamma}_{\boldsymbol{\lambda}}^{-k}, \hat{\Theta}_{\boldsymbol{\lambda}}^{-k}).$$

The cross-validation criterion is the following:

$$\mathrm{CV}(\boldsymbol{\lambda}) = \sum_{k=1}^{K} \mathcal{D}^k(\boldsymbol{\lambda}). \tag{4.1}$$

The latter criterion is computed on a tuning parameters matrix constructed as follows:

$$\begin{bmatrix} (\lambda_1^{\min}, \lambda_2^{\min}) & \ldots & (\lambda_1^{\min}, \lambda_2^{\max}) \\ \vdots & \ddots & \vdots \\ (\lambda_1^{\max}, \lambda_2^{\min}) & \ldots & (\lambda_1^{\max}, \lambda_2^{\max}) \end{bmatrix}$$

Finally, the couple of tuning parameters which minimizes the criterion in (4.1) is chosen as optimal.

In the longitudinal framework, particular attention should be paid to how the data are split. In fact, when applying cross-validation, we are assuming that the K parts into which the data are split are independent from each other. This is maintained if we split with respect to the n individuals; however, sometimes, n is very small and this kind of cross-validation is unfeasible. It is possible to apply cross-validation splitting with respect to the time dimension: leave-one-out cross-validatory methods for dependent data were proposed by Burman et al. (1994) and Racine (2000). The idea is to reduce the training set by removing some observations preceding and following the observation in the test set. This results in leaving a gap between the test sample and the training samples, on both sides

of the test sample in order to achieve approximate independence between the training and the test data.
However, in our case, even if we are dealing with dependent observations, in the likelihood we consider conditional distributions. Once we condition on the previous observation in time, the conditional distribution is independent from the previous one:

$$f(\mathbf{X}_t \mid \mathbf{X}_{t-1} = \mathbf{x}_{t-1}, \Gamma, \Theta) \perp\!\!\!\perp f(\mathbf{X}_{t-1} \mid \mathbf{X}_{t-2} = \mathbf{x}_{t-2}, \Gamma, \Theta).$$

Therefore, in our case, splitting can be done both with respect to the sample size n and the time dimension T.

Cross-validation is computationally burdensome: it requires the fitting of K models for each element of the tuning parameters matrix. Less computationally expensive model selection criteria are available. These include:

- the Bayesian Information Criterion (BIC) by Schwarz (1978), which is defined as follows:

$$\begin{aligned}\mathrm{BIC}(\boldsymbol{\lambda}) =& -2\ell^{\mathrm{PS}}(\hat{\Gamma}_{\boldsymbol{\lambda}}, \hat{\Theta}_{\boldsymbol{\lambda}}) + \log(nT)(a_n/2 + b_n + p) \\ =& npT\log(2\pi) - nT\log\left|\hat{\Theta}_{\boldsymbol{\lambda}}\right| + nT\,\mathrm{Tr}(S_{\hat{\Gamma}_{\boldsymbol{\lambda}}}\hat{\Theta}_{\boldsymbol{\lambda}}) + \log(nT)(a_n/2 + b_n + p),\end{aligned}$$

  where a_n is the number of non-zero off-diagonal elements of $\hat{\Theta}_{\boldsymbol{\lambda}}$ and b_n is the number of non-zero elements of $\hat{\Gamma}_{\boldsymbol{\lambda}}$;

- the following Akaike Information Criterion (AIC) developed by Akaike (1998):

$$\begin{aligned}\mathrm{AIC}(\boldsymbol{\lambda}) =& -2\ell^{\mathrm{PS}}(\hat{\Gamma}_{\boldsymbol{\lambda}}, \hat{\Theta}_{\boldsymbol{\lambda}}) + 2(a_n/2 + b_n + p) \\ =& npT\log(2\pi) - nT\log\left|\hat{\Theta}_{\boldsymbol{\lambda}}\right| + nT\,\mathrm{Tr}(S_{\hat{\Gamma}_{\boldsymbol{\lambda}}}\hat{\Theta}_{\boldsymbol{\lambda}}) + 2(a_n/2 + b_n + p).\end{aligned}$$

These criteria were introduced for different purposes. AIC aims to minimize the Kullback-Leibler divergence between the distribution of the true model and that of the estimated one. On the other hand, BIC attempts to select a model that maximizes the posterior model probability, given the data. Hence, they have different properties: in general, AIC is optimal for prediction accuracy while BIC is consistent in selecting the true graph (Yang, 2005). However, the consistency of BIC is intended in the setting of a fixed number of variables p and growing sample size. When $p > n$, this does not hold. Foygel and Drton (2010)

introduced the following Extended Bayesian Information Criterion (EBIC) for graphical models:

$$\mathrm{BIC}_\gamma = -2\ell^{\mathrm{PS}}(\hat{\Gamma}_{\boldsymbol{\lambda}}, \hat{\Theta}_{\boldsymbol{\lambda}}) + \log(nT)(a_n/2 + b_n + p) + 4(a_n/2 + b_n + p)\gamma \log p.$$

The drawback of this method is that it includes an additional parameter γ to be tuned. Note that when $\gamma = 0$, the classical BIC is recovered. Numerical results in Foygel and Drton (2010) demonstrate that positive values of γ lead to improved graph inference when the number of variables and the sample size are of comparable size. Upon suggestion from the author, we fix γ at 0.5.

Chapter 5

Results

5.1 Simulations

In order to assess the performance of the copula VAR model and to compare it with the TSCGM, we conducted some simulation studies. We stress the fact that the TSCGM was designed specifically for normal data. The aim of the simulations was to establish whether:

- CVM performs comparably to the TCSGM in case of normal data;
- CVM performs better than the TSCGM in case of non normal data;
- which nonparametric estimate of the CDF performs better.

The first step in the simulation process is to generate the sparse autoregressive matrix and precision matrix. Algorithms (2) and (3) explain in detail the method we used, which is similar to that in Yin and Li (2011). In the simulations, we choose as sparsity level 1/3, which means that one third of the elements of both Γ and Θ are different from zero.
Once we have the true coefficient matrices Θ and Γ, we simulate the longitudinal data according to the following distributions:

- Normal;
- Log normal;
- Exponential;
- Exponential with time-varying rate.

Algorithm 2 Generation of sparse Θ

Initialization of the sparsity level desired, *spars*
function ThetaGenerate(*spars*)
    $\Theta^O \leftarrow 0$ matrix with dimensions $p \times p$
    for each element of Θ^O, θ^O_{ij}, $i, j = 1, \ldots, p$ **do**
        tmp1 $\leftarrow 1$ with probability *spars*, 0 otherwise
        if tmp1=0 **then**
            $\theta^O_{ij} \leftarrow 0$
        else if tmp1=1 **then**
            tmp2 $\leftarrow 0$ or 1 with equal probability
            if tmp2=0 **then**
                $\theta^O_{ij} \leftarrow$ random uniform in the interval $(-1, -0.5)$
            else if tmp2=1 **then**
                $\theta^O_{ij} \leftarrow$ random uniform in the interval $(0.5, 1)$
    Force Θ^O to be positive definite
    $\theta^O_{ii} \leftarrow 1$, $\forall i = 1, \ldots, p$
    $\Sigma^O \leftarrow (\Theta^O)^{-1}$
    $\Sigma^N \leftarrow$ corresponding correlation matrix of Σ^O
    $\Theta^N \leftarrow (\Sigma^N)^{-1}$
    return Θ^N

Algorithm 3 Generation of sparse Γ

Initialization of the sparsity level desired, *spars*
function GammaGenerate(*spars*, Θ)
    $\Gamma \leftarrow 0$ matrix with dimensions $p \times p$
    mv $\leftarrow \min(\theta_{ij}), \forall \theta_{ij} \neq 0, i, j = 1, \ldots, p$
    for each element of Γ, γ_{ij}, $i, j = 1, \ldots, p$ **do**
        tmp1 $\leftarrow 1$ with probability *spars*, 0 otherwise
        if tmp1=0 **then**
            $\gamma_{ij} \leftarrow 0$
        else if tmp1=1 **then**
            tmp2 $\leftarrow 0$ or 1 with equal probability
            if tmp2=0 **then**
                $\gamma_{ij} \leftarrow$ random uniform in the interval $(-1, -\text{vm})$
            else if tmp2=1 **then**
                $\gamma_{ij} \leftarrow$ random uniform in the interval $(\text{vm}, 1)$
    $\gamma_{ii} \leftarrow 0.1$, $\forall i = 1, \ldots, p$
    return Γ

5.1.1 Simulation setting

We first simulate the latent process $\mathbf{z}_t$, for $t = 1, \ldots, T$. The first time component $\mathbf{z}_1$ is generated as a realization of a $\mathcal{N}_p(\mathbf{0}, \Theta^{-1})$. The following unscaled time component is generated as $\mathbf{z}_2^* = \Gamma \mathbf{z}_1 + \boldsymbol{\eta}_2$, where $\boldsymbol{\eta}_2 \sim \mathcal{N}_p(\mathbf{0}, \Theta^{-1})$. Then, each component of $\mathbf{z}_2^*$ is scaled in order to have unit variance: $z_{j2} = \frac{z_{j2}^*}{\sqrt{\sigma_j}}$,for $j = 1, \ldots, p$, where σ_j is the j-th diagonal element of $\Sigma = \Theta^{-1} + \Gamma\Theta^{-1}\Gamma^\top$. We obtain in this way $\mathbf{z}_2$. This is repeated for $t = 1, \ldots, T$, which gives a time series dataset for p variables (genes): $(\mathbf{z}_1, \ldots, \mathbf{z}_T)$. The process is repeated n times to obtain n i.i.d. replicates.
For the normal simulation, the latter is considered as observed and used directly. For the non normal simulations, a further step is necessary. When generating $\mathbf{z}_t$, each component is transformed according to the inverse of the Gaussian copula transformation: $x_{jt} = F^{-1}(\Phi(z_{jt}))$. F can be any continuous CDF; we choose the exponential distribution with rate 1 and the log normal distribution with mean and variance of the logarithm equal to 0 and 1, respectively. Moreover, we simulated exponential data with time-changing rate: at the first time point the rate is $\gamma = 0.5$, then it increases as $\gamma = t/2$, for $t = 2, \ldots, T$.
Once we obtain the simulated data, we apply both the TSCGM and the CVM with various CDF nonparametric estimators. "CVM ecdf" refers to the ECDF estimator in (3.2); "CVM kde" is the kernel estimator in (3.4); "CVM kde t.v." corresponds to the time-varying estimator in (3.5). We apply the lasso penalty in all cases. Figure (5.1) shows an example of the lasso coefficient paths for the estimated parameters.

5.1.2 Simulation results

Figures (5.2), (5.3), (5.4), (5.5) show the ROC curves for both the CVM and the TSCGM in various settings, obtained with just one simulation per each case. The performance of the model has also been assessed using specificity (Spe=TN/(TN + FP)), sensitivity (Sen=TP/(TP + FN)), and F1 score (F1=2TP/(2TP + FP + FN)), where TP, TN, FP, and FN are the numbers of true positives, true negatives, false positives, and false negatives in identifying the non-zero elements in the matrices Γ and Θ. The results did not change considerably according to whether the AIC, BIC or extended BIC criterion was used; therefore, we only report the results for the BIC criterion. The results are presented in tables (5.1), (5.2), (5.3), (5.4) for different settings, and they refer to only one simulation per each case. The first row of each table represents the results of the TSCGM when the tuning parameters are chosen with the BIC criterion. "Oracle" indicates the results ob-

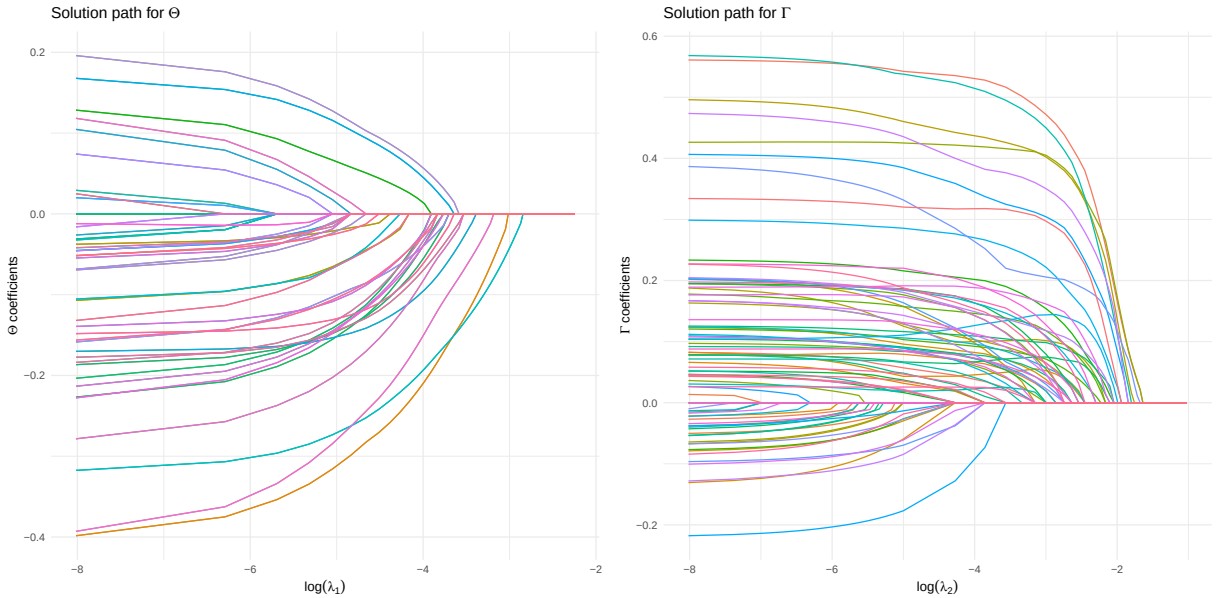


Figure 5.1: Lasso coefficient paths for the CVM: plot of coefficient profiles for Θ (left) and Γ (right), as a function of $\log(\lambda_1)$ $\log(\lambda_2)$, respectively. Log normal simulated data with $n = 20$, $t = 10$, $p = 10$.

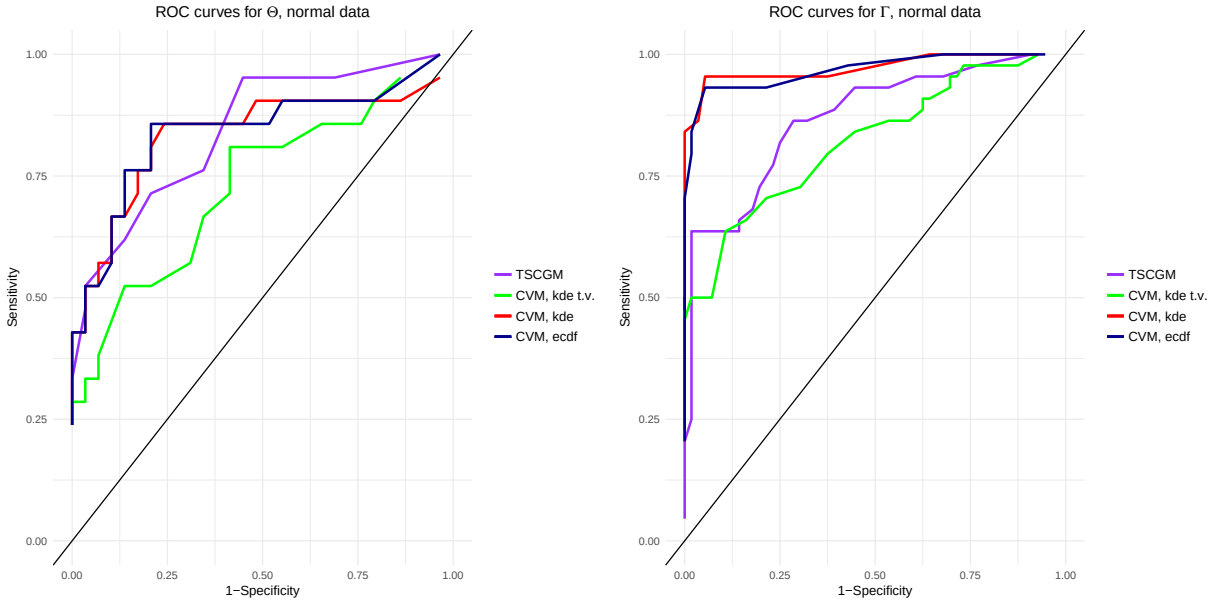


Figure 5.2: ROC curves for Θ (left) and Γ (right), Normal simulated data with $n = 20$, $t = 10$, $p = 10$.

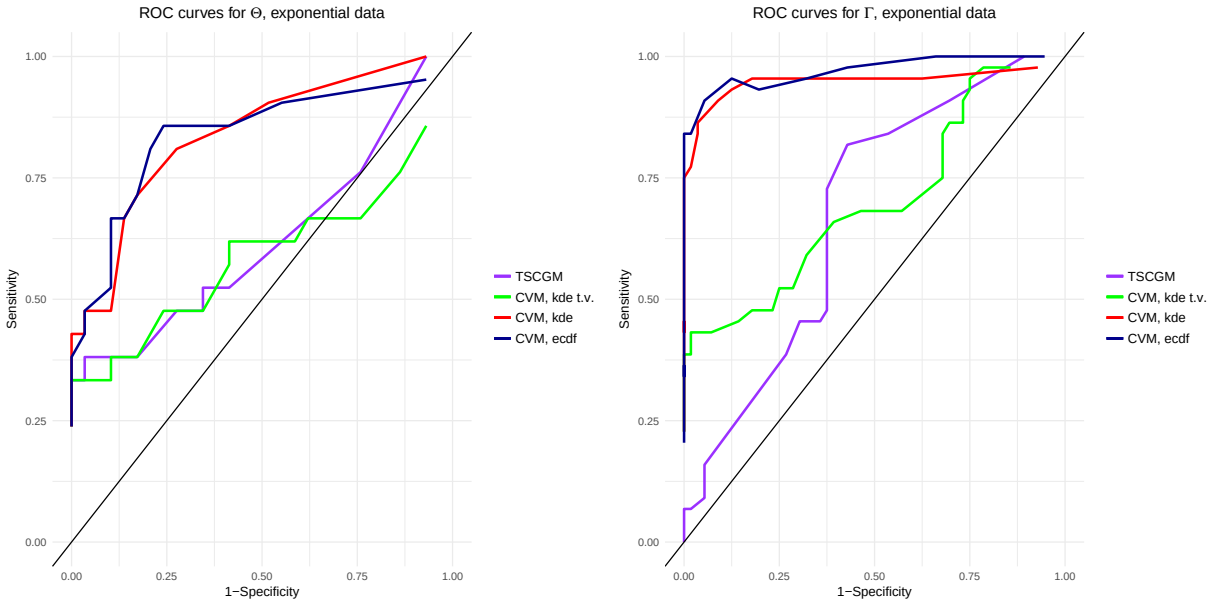


Figure 5.3: ROC curves for Θ (left) and Γ (right), Exponential simulated data with $n = 20$, $t = 10$, $p = 10$.

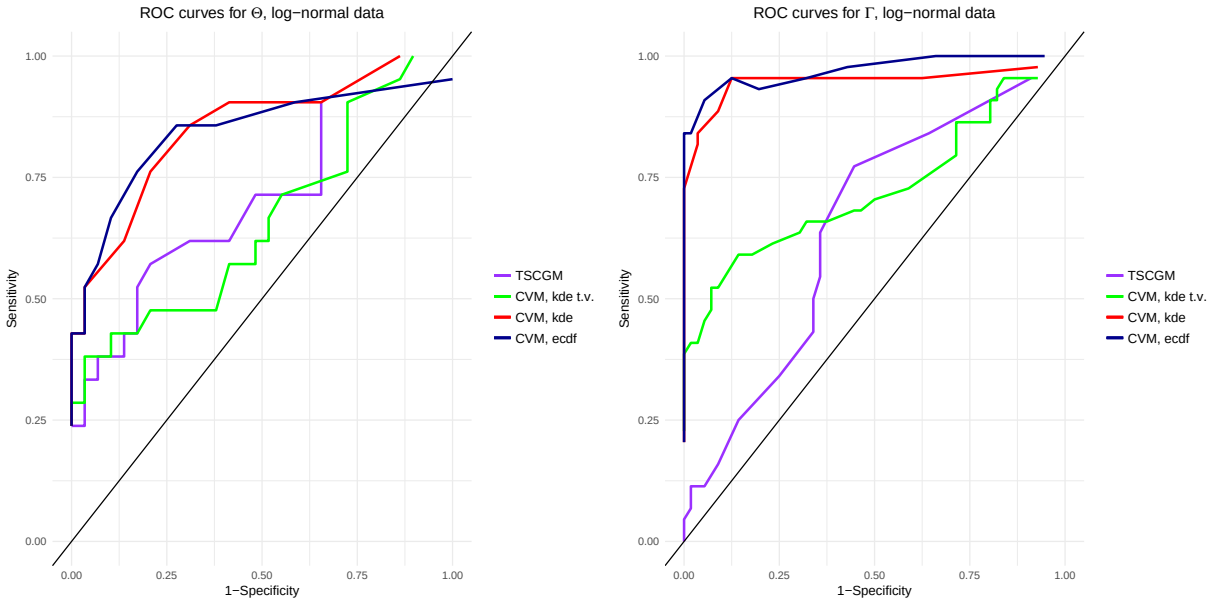


Figure 5.4: ROC curves for Θ (left) and Γ (right), Log normal simulated data with $n = 20$, $t = 10$, $p = 10$.

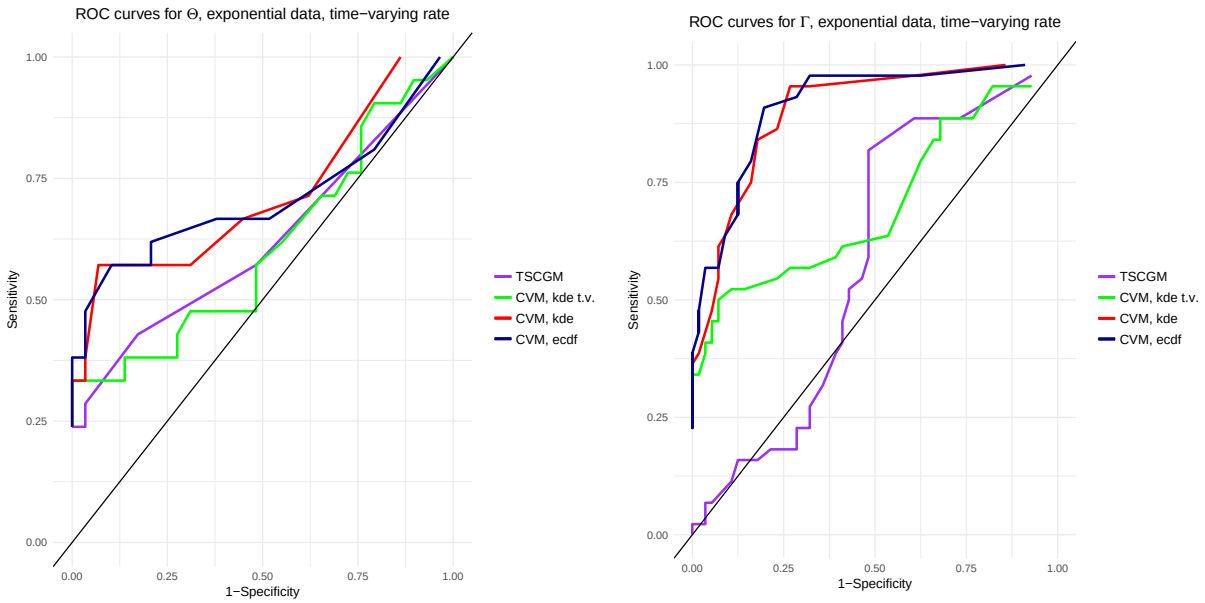


Figure 5.5: ROC curves for Θ (left) and Γ (right), Exponential simulated data with time-varying rate, $n = 20$, $t = 10$, $p = 10$.

tained applying the CVM and choosing the tuning parameters which provide the highest F1 score. The results in the tables suggest that sometimes the BIC criterion did not accurately select the true structure of the data when the CVM is applied, especially for Θ. However, the oracle results show that the method works and it improves the existing TSCGM. Of course, choosing the tuning parameters that provide the highest F1 score is not feasible with real data, as the true structure is not known. Therefore, an efficient model selection criterion has to be sought. The ROC curves show that, in general, the CVM with non time-varying estimators perform better than the TSCGM and the CVM with the time-varying kernel estimator. In the time-varying exponential rate simulation, the time-varying kernel estimator still performs worse than the non time-varying estimators. It is worthwhile to note that, also in case of normal data, the proposed method performs comparably to the TSCGM. In addition, both the CVM and the TSCGM estimate better the Γ matrix rather than the Θ matrix in all settings.

Θ	Spe	Sen	F1
TSCGM, BIC	0.86	0.62	0.68
CVM kde t.v., BIC	1.00	0.24	0.39
CVM kde t.v., oracle	0.59	0.81	0.68
CVM kde, BIC	0.48	0.91	0.69
CVM kde, oracle	0.76	0.86	0.78
CVM ecdf, BIC	0.45	0.91	0.68
CVM ecdf, oracle	0.79	0.86	0.80

Γ	Spe	Sen	F1
TSCGM, BIC	0.95	0.71	0.80
CVM kde t.v., BIC	1.00	0.23	0.37
CVM kde t.v., oracle	0.89	0.64	0.72
CVM kde, BIC	0.98	0.82	0.89
CVM kde, oracle	0.95	0.96	0.94
CVM ecdf, BIC	0.96	0.82	0.88
CVM ecdf, oracle	0.95	0.93	0.93

Table 5.1: Results from one simulation where $n = 20$, $t = 10$, $p = 10$. Normal data.

Θ	Spe	Sen	F1
TSCGM, BIC	0.83	0.38	0.47
CVM kde t.v., BIC	0.03	0.95	0.56
CVM kde t.v., oracle	0.59	0.62	0.57
CVM kde, BIC	0.48	0.91	0.69
CVM kde, oracle	0.72	0.81	0.74
CVM ecdf, BIC	0.41	0.91	0.67
CVM ecdf, oracle	0.76	0.86	0.78

Γ	Spe	Sen	F1
TSCGM, BIC	0.66	0.59	0.58
CVM kde t.v., BIC	1.00	0.23	0.37
CVM kde t.v., oracle	0.21	0.97	0.66
CVM kde, BIC	0.96	0.86	0.91
CVM kde, oracle	0.96	0.86	0.91
CVM ecdf, BIC	0.79	0.93	0.85
CVM ecdf, oracle	0.95	0.91	0.92

Table 5.2: Results from one simulation where $n = 20$, $t = 10$, $p = 10$. Exponential data.

Θ	Spe	Sen	F1
TSCGM, BIC	0.90	0.38	0.50
CVM kde t.v., BIC	0.00	1.00	0.57
CVM kde t.v., oracle	0.28	0.91	0.62
CVM kde, BIC	0.14	1.00	0.63
CVM kde, oracle	0.69	0.86	0.75
CVM ecdf, BIC	0.41	0.91	0.67
CVM ecdf, oracle	0.72	0.86	0.77

Γ	Spe	Sen	F1
TSCGM, BIC	0.66	0.52	0.53
CVM kde t.v., BIC	1.00	0.23	0.37
CVM kde t.v., oracle	0.86	0.59	0.67
CVM kde, BIC	0.88	0.93	0.89
CVM kde, oracle	0.88	0.96	0.90
CVM ecdf, BIC	0.95	0.91	0.92
CVM ecdf, oracle	0.95	0.91	0.92

Table 5.3: Results from one simulation where $n = 20$, $t = 10$, $p = 10$. Log normal data.

Θ	Spe	Sen	F1
TSCGM, BIC	0.97	0.24	0.37
CVM kde t.v., BIC	0.00	1.00	0.59
CVM kde t.v., oracle	0.21	0.91	0.60
CVM kde, BIC	0.14	1.00	0.62
CVM kde, oracle	0.93	0.57	0.69
CVM ecdf, BIC	0.03	1.00	0.60
CVM ecdf, oracle	0.90	0.57	0.67

Γ	Spe	Sen	F1
TSCGM, BIC	0.50	0.64	0.56
CVM kde t.v., BIC	1.00	0.23	0.37
CVM kde t.v., oracle	0.32	0.89	0.65
CVM kde, BIC	0.82	0.77	0.77
CVM kde, oracle	0.73	0.96	0.83
CVM ecdf, BIC	0.80	0.91	0.84
CVM ecdf, oracle	0.80	0.91	0.84

Table 5.4: Results from one simulation where $n = 20$, $t = 10$, $p = 10$. Exponential data with time-varying rate.

5.2 Applications to gene expression data

In this section, the application of the CVM to time course gene expression datasets is shown. In particular, we considered the study of genes involved in the circadian regulation in a plant and the genes related to the transcriptional repressor farR in a bacteria. When representing the dynamic networks in figures (5.8) and (5.11), the self loops have been omitted for a more clear representation.

5.2.1 Application to *Arabidopsis thaliana* dataset

A. thaliana is a small flowering plant which is a widely used model for plant biology; it was the first plant to have its genome sequenced. We used a dataset that can be found in GEO data repository (https://www.ncbi.nlm.nih.gov/geo/, GEO accession: GSE3416). The gene expression levels were measured with Affymetrix microarrays. Data were collected with an interval of 4 h: the $T = 6$ time points are 0 h, 4 h, 8 h, 12 h, 16 h, and 20 h. $n = 3$ biological replicates were analyzed. The aim of the experiment was to understand the plant's circadian cycle and how the transcript levels of leaf-expressed genes change in a normal day-night cycle. We focused on the contemporaneous and dynamic interactions among the genes involved in circadian regulation. Figure (5.6) shows the circadian clock model known in biology. We considered the subset of nine genes that are known in the literature to be involved in circadian regulation, as in Abegaz and Wit (2013), Grzegorczyk and Husmeier (2010) and Grzegorczyk and Husmeier (2011). Those genes can be divided in two groups, according to whether their expression peaks in the morning or in the evening: *morning genes*, including LHY, CCA1, PRR9, and PRR5 and *evening genes*, which include TOC1, ELF4, ELF3, GI, and PRR3. The longitudinal data are shown in figure (5.7).

We then applied the CVM tuning the lasso parameters with BIC. We used the non time-varying kernel estimator. The range of parameters to choose from was $(0.01, 1)$ with length 30, for both λ_1 and λ_2. BIC chose $\lambda_1^{opt} = 0.01$ and $\lambda_2^{opt} = 0.22$. Figure (5.8) represents the estimated networks. The density of the estimated dynamic network is 0.47 while the density of the contemporaneous one is 0.26.

5.2.2 Application to *Neisseria gonorrhoeae* dataset

Neisseria is a bacteria responsible for the sexually transmitted disease gonorrhea. We are interested in detecting interactions among the 60 genes related to the transcriptional

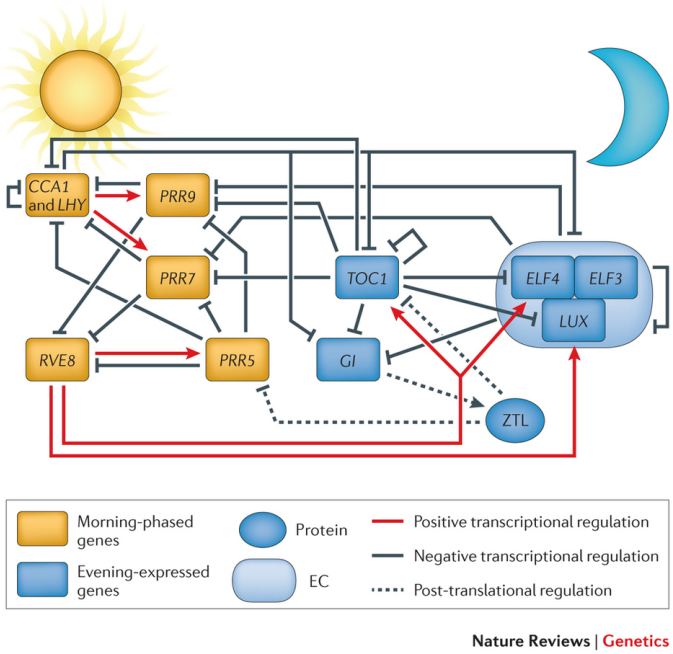


Figure 5.6: Model of the Arabidopsis thaliana circadian clock, from Greenham and McClung (2015).

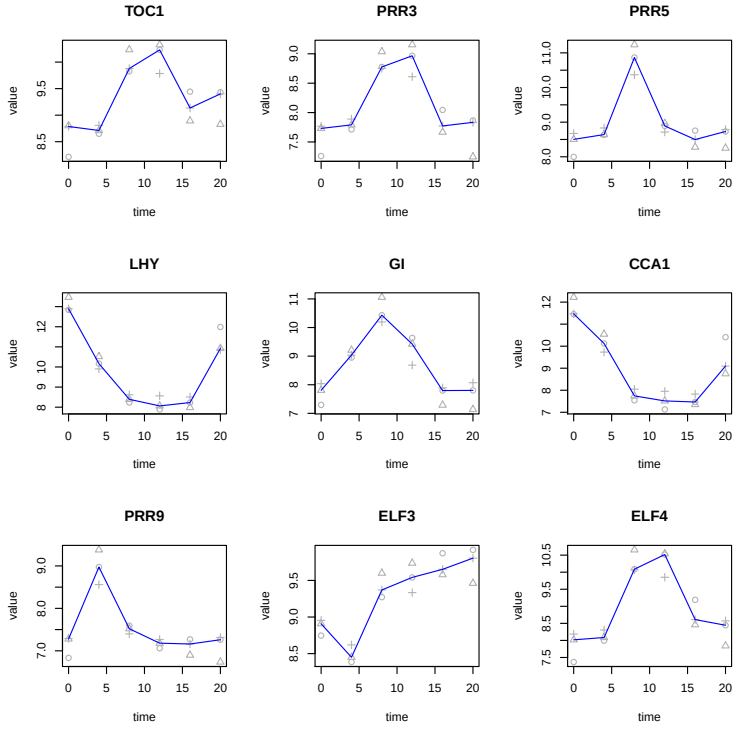


Figure 5.7: Time series of the 9 genes considered for *A. thaliana*. The different shapes (triangle, circle and plus) represent the replicates.

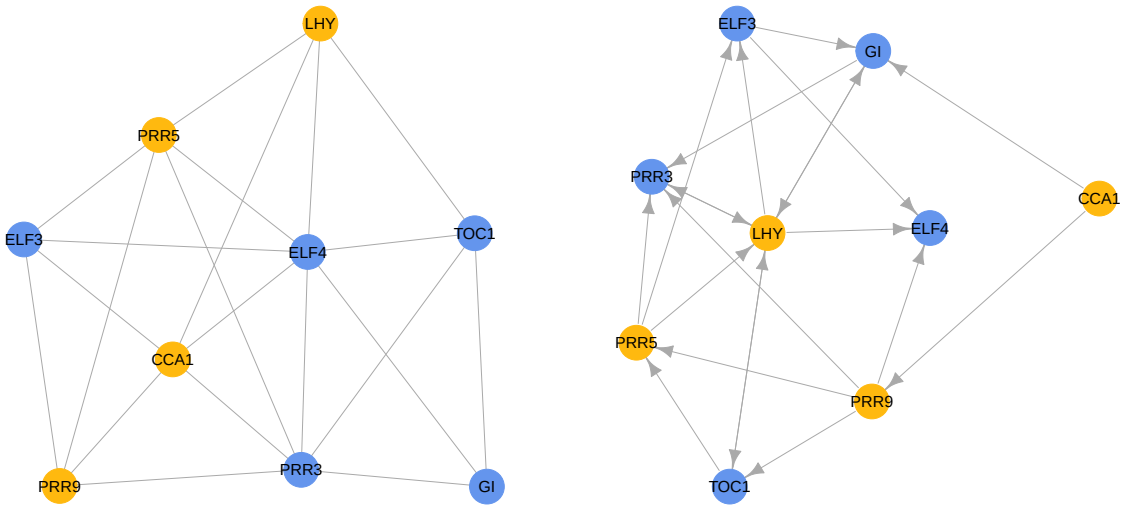


Figure 5.8: Contemporaneous (left) and dynamic (right) networks corresponding to the estimates of Θ and Γ, respectively, from *A. thaliana*. The dynamic network refers to 4-hour relations. Morning genes are represented by yellow vertices, evening genes by blue vertices.

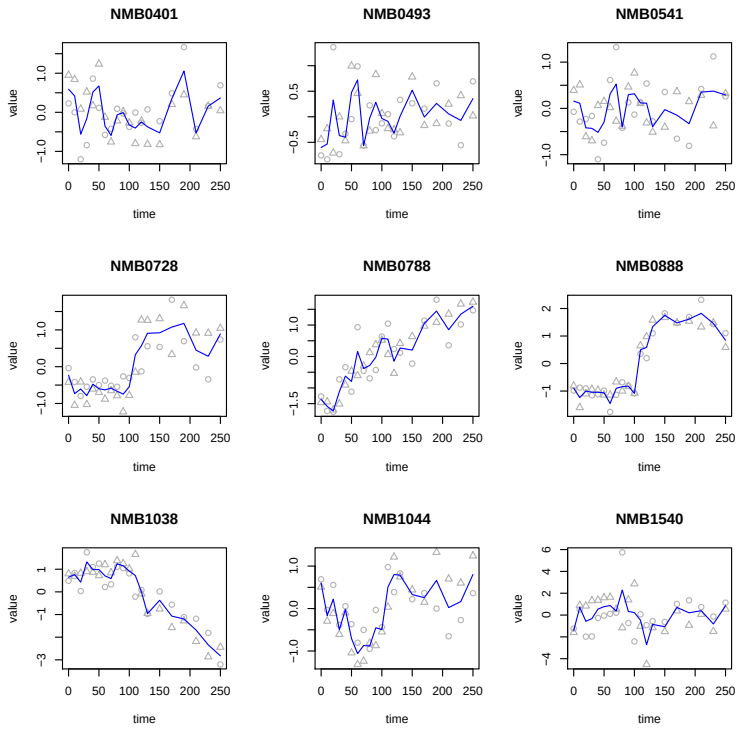


Figure 5.9: Time series of 9 genes among the 60 genes from *Neisseria*. The different shapes (triangle and circle) represent the replicates.

repressor FarR. In molecular genetics, a transcriptional repressor is a DNA- or RNA-binding protein that inhibits the expression of one or more genes.
The data available concern $n = 2$ biological replicates observed across $T = 20$ time points with intervals of 10 hours. Figure (5.9) shows the time series of 9 genes from the dataset. We applied the CVM with non time-varying kernel estimator and, as in the previous example, we first tried to tune the lasso parameters with BIC. From the range $(0.01, .8)$ with length 20, BIC choose $\lambda_1^{opt} = \lambda_2^{opt} = 0.8$, resulting in too sparse estimates, i.e. $\hat{\Theta}$ and $\hat{\Gamma}$ almost diagonal. Figures (5.10) and (5.11) represent the resulting networks when $\lambda_1 = 0.15$ and $\lambda_2 = 0.1$ are chosen in order to obtain networks with density similar to those known to biologists (densities: 0.06 for the contemporaneous network, 0.09 for the dynamic network).

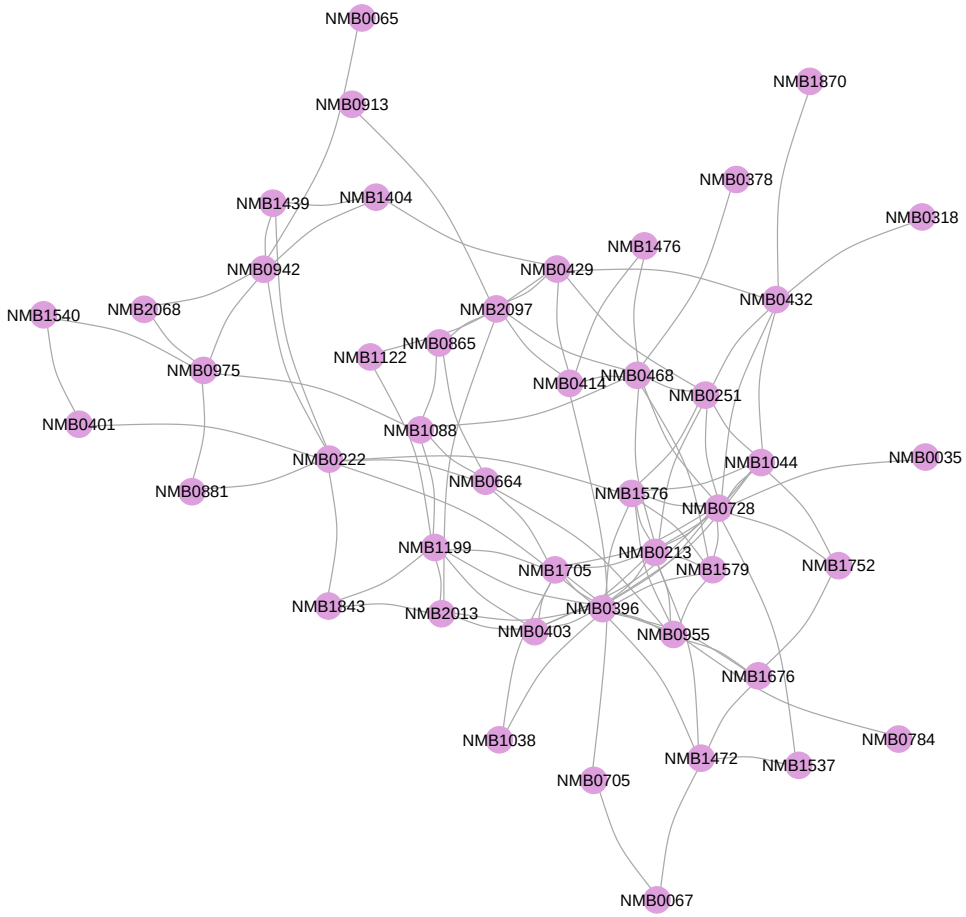


Figure 5.10: Contemporaneous network corresponding to the estimate of Θ from *Neisseria*.

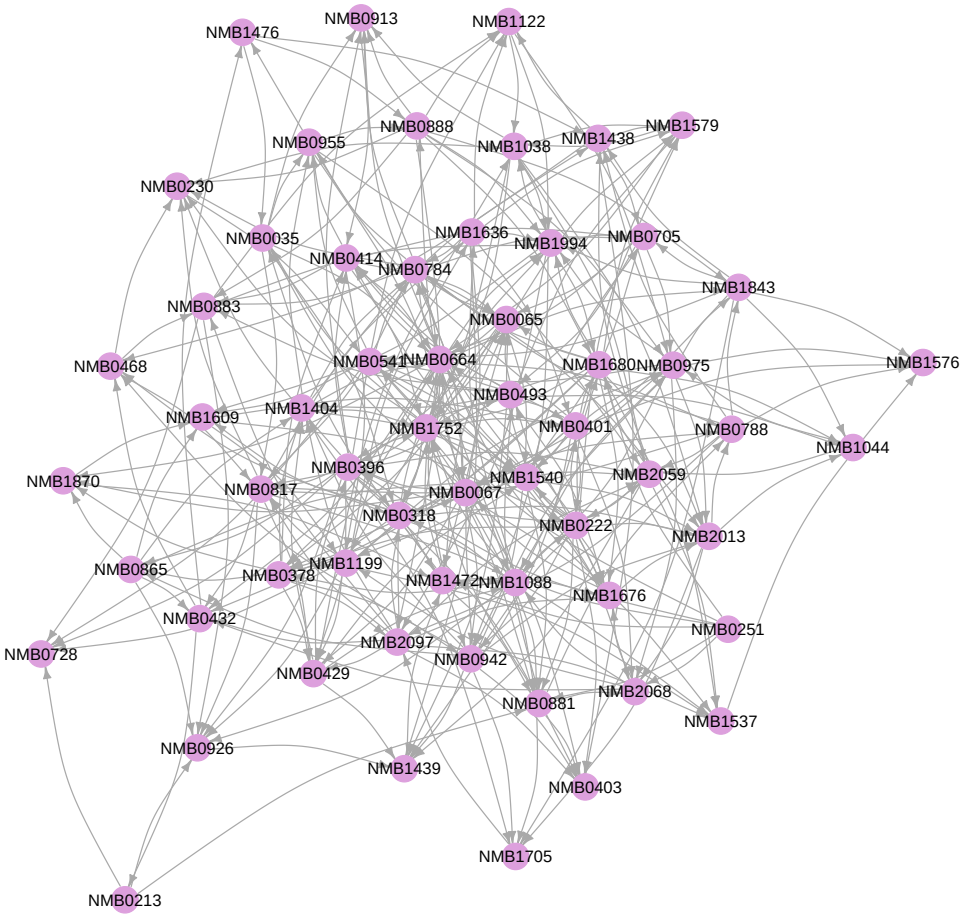


Figure 5.11: Dynamic network of 10-hour relations among genes. It corresponds to the estimate of Γ from *Neisseria*.

Conclusion

In this thesis, we developed an extension of the TSCGM proposed by Abegaz and Wit (2013) for non normal data. Applications of the proposed method can be to longitudinal genetic dataset as well as any generic longitudinal dataset, on the condition that the measured variables are continuous.

We assumed that there exist a latent normally distributed VAR process, related to the observed process as a one-to-one transformation called Gaussian copula. Since this transformation involves the unknown marginal distributions, various nonparametric methods to estimate the latter have been proposed. This allowed us to estimate the latent process and treat it as observed. We used a pseudo-likelihood approach, in that we approximate the full likelihood neglecting the derivative of the Gaussian copula transformation: this is the standard approach when the Gaussian copula is used, as there is little information in it anyway. The parameters to estimate are the covariance and autoregressive matrices of the latent process. Non-zero entries of the autoregressive matrix correspond to edges in the dynamic network, which explains delayed interactions among the variables. Likewise, non-zero entries of the covariance matrix are related to edges in the undirected network, which shows contemporaneous relations. We suppose that those networks are sparse: to account for this, two lasso penalties are added to the pseudo-likelihood. The estimation is achieved with a two-stage iterative procedure using the MRCE algorithm by Rothman et al. (2010). The tuning parameters of the lasso penalties have to be chosen: we considered the AIC, BIC and extended BIC criteria.

Simulation studies have been conducted to assess the performance of the proposed method compared to the TSCGM in case of normal and non normal data. The results showed that, in the evaluated settings, the proposed method outperforms the TSCGM when the data are non normal, and performs comparably to the TSCGM in case of normal data. However, the model selection criteria considered are sometimes not accurate in selecting

the true structure of the network when the proposed model is applied.

More extensive simulation studies could be done in future research to seek a more accurate model selection criterion, cross-validation being a possible choice. The Kullback–Leibler Cross Validation (KLCV) criterion by Vujačić et al. (2015) might also be considered, which is a computationally fast alternative to cross-validation. However, KLCV was designed for graphical models with a single penalty; in order to be applied to the copula VAR model, an extension of the method is needed to account for the additional penalty for the autoregressive matrix.

So far we have made the rather strict assumption of considering the autoregressive and covariance matrices constant across time. A possible topic of further research might be to allow those matrices to change over time, identifying the changes with changepoint detection.

Another limiting assumption that we made is that the variables have to be continuous; extensions of the model could allow for discrete variables. In that case, the extended rank likelihood approach by Hoff (2007) is suggested and has been extensively applied when dealing with discrete variables in Gaussian copula graphical models (Dobra and Lenkoski, 2009; Abegaz and Wit, 2015).

Appendix

R code

In what follows, the **R** code to apply the proposed copula VAR model to a longitudinal dataset is presented. The functions for the simulations are also provided.
Many functions are modifications of those in the package `SparseTSCGM` by Fentaw Abegaz.

```
# required packages:

library(QUIC)
library(longitudinal)
library(MRCE)
library(SparseTSCGM)
library(glasso)
library(network)
library(compositions)
library(ks)
```

The following functions compute the the Gaussian copula transformation with various nonparametric methods.

```
# modified ecdf

my.ecdf <- function (x)
{
  x <- sort(x)
  n <- length(x)
```

```
  if (n < 1)
    stop("'x' must have 1 or more non-missing values")
  vals <- unique(x)
  rval <- approxfun(vals, (cumsum(tabulate(match(x, vals))) + .5)/(
     n + 1),
                    method = "constant", f = 0, ties = "ordered")
  class(rval) <- c("ecdf", "stepfun", class(rval))
  assign("nobs", n, envir = environment(rval))
  attr(rval, "call") <- sys.call()
  rval
}

# univariate time-varying kernel estimation, Gaussian kernel

weight <- function(x, y, omega)
{
  if((omega > 1) | (omega <= 0)) stop("omega must be in (0, 1]\n")
  w.tmp <- omega^abs(x[2,] - y[2])
  w.sum <- sum(w.tmp)
  w.tmp/w.sum
}

my.kde <- function(x, omega, time.vec, bw)
{
  x2 <- rbind(x, time.vec)
  cdfest <- apply(x2, 2, function(y) sum(pnorm(abs(y[1] - x2[1,])/
     bw)*weight(x2, y, omega)))
  list(Fhatj=t(cdfest))
}

# Gaussian copula transformation

fp <- function(x, npest, omega=.925)
{
  p <- ncol(x)
  nT <- nrow(x)
```

```
  out2 <- array(NA, dim=c(nT, p))
  time.vec <- rep(attr(x, "time"), attr(x, "repeats"))
  for(i in 1:p)
  {
    if(npest=="tvkde")
    {
      h.tmp <- kedd::h.amise(x[,i], kernel="gaussian")
      result <- my.kde(x=x[,i], omega=omega, time.vec=time.vec, bw=
         h.tmp$h)
      out2[,i] <- result$Fhatj
    }
    else if(npest=="kde")
    {
      fhat <- kde(x=x[,i], binned=TRUE)
      out2[,i] <- pkde(x[,i], fhat)
    }
    else if(npest=="ecdf")
    {
      for(j in 1:nT)
      {
        out2[j,i] <- my.ecdf(x[,i])(x[j,i])
      }
    }
  }
  list(Fhat=out2)
}

transf.cvm <- function(stimade, p_num, time, n_obs)
{
  xy.Fhat <- array(NA, c(time, p_num, n_obs))
  for (i in 1:n_obs)
  {
    for (t in 1:time)
    {
      cc <- 1 + (t - 1) * n_obs + (i - 1)
      xy.Fhat[t, , i] <- stimade$Fhat[cc, ]
```

```
    }
  }
  out.z <- array(NA, c(time, p_num, n_obs))
  set.seed(1234)
  for(i in 1:n_obs)
  {
    for(t in 1:time)
    {
      for(j in 1:p_num)
      {
        out.z[t, j, i] <- qnorm(xy.Fhat[t, j, i])
      }
    }
  }
  out.z
}

pre.cvm <- function(xy.z=xy.z, time=time, model=c("ar1","ar2"))
{
  model = match.arg(model)
  if (model=="ar1") {
    X <- xy.z[1:time-1,,,drop=FALSE]
    Y <- xy.z[2:time,,,drop=FALSE]
  }
  else if (model=="ar2") {
    t1=time-1
    t2=time-2
    Y <- round(xy.z[3:time,,,drop=FALSE],3)
    X1 <- round(xy.z[2:t1,,,drop=FALSE],3)
    X2<- round(xy.z[1:t2,,,drop=FALSE],3)
    X <- abind(X1, X2,along = 2 )
  }
  T <- dim(Y)[1]
  p <- dim(X)[2]
  n <- dim(Y)[3]
  q <- dim(Y)[2]
```

```
  xtyi <- array(NA, c(p,q,n))
  xtxi <- array(NA, c(p,p,n))
  ytyi <- array(NA, c(q,q,n))
  for(i in 1:n){
    XX <- X[,,i]
    YY <- Y[,,i]
    XX2 <- X[,,i]^2
    YY2 <- Y[,,i]^2
    xtyi[,,i]=crossprod(XX,YY)
    xtxi[,,i]=crossprod(XX)
    ytyi[,,i]=crossprod(YY)
  }
  xty=apply(xtyi, c(1,2), sum) # C_xx_
  xtx=apply(xtxi, c(1,2), sum) # C_x_
  yty=apply(ytyi, c(1,2), sum) # C_x
  xtxt=apply(xtxi, c(1,2), sum)/(n*T)
  xtx2=(n*T)*colMeans(apply(XX2, c(1,2), sum))
  yty2=(n*T)*colMeans(apply(YY2, c(1,2), sum))

  out.data <- list(xty=xty, xtx=xtx, yty=yty, xtxt=xtxt, xtx2=xtx2,
                   yty2=yty2,p=p, T=T, n=n, q=q)
  return(out.data)
}
```

`compute.cvm` is the function for the two-stage iterative estimation method.

```
compute.cvm <- function(stimade, model = c("ar1", "ar2"), T=T, n=n,
                        p=p, q=q, time=time, tps=tps, xty = xty,
                        xtx = xtx, yty = yty, xtxt = xtxt,
                        xtx2 = xtx2, yty2 = yty2, lam1=lam1,
                        lam2=lam2, optimality = c("NULL","bic","bic_
                           ext","aic"), setting=setting)
{
  nlam=(n*T)*lam2
  lam11 <- lam1*(1-diag(q))
  old.B <- diag(p)
```

```
k=0
mab =  sum(sum(abs(old.B)))

while (1) {
  k = k + 1
  samp.cov = (yty - t(xty) %*% old.B - t(old.B) %*%
                xty + t(old.B) %*% xtx %*% old.B)/(n * T)
  if (k == 1) {
    g.out = QUIC(S = samp.cov, rho = lam11, tol = 1e-04,
                 msg = 0, maxIter = 10000)
    old.om.i <- cov2cor(g.out$W)
    old.om <- solve(old.om.i)
    old.om <- round(old.om, digits=8)
  }
  if (k > 1) {
    old1.om = old.om
    old1.om.i = old.om.i
    g.out1 = QUIC(S = samp.cov, rho = lam11, tol = 1e-04,
                  msg = 0, maxIter = 10000, X.init = old1.om,
                  W.init = old1.om.i)
    old.om.i <- cov2cor(g.out1$W)
    old.om <- solve(old.om.i)
    old.om <- round(old.om, digits=8)
  }
  if (!is.numeric(old.om))
    old.om <- diag(q)
  xtyom = (xty %*% old.om)
  wt1 <- matrix(lam2, nrow = p, ncol = q)
  rho2 <- wt1 * (n * T)
  diag(rho2) <- 0
  warmstart = 1
  if (k == 1)
    warmstart = 0
  B = MRCE:::rblasso(s = xtx, m = xtyom, om = old.om, nlam = rho2
     ,
```

```
                tol = 1e-05, sbols = mab, maxit = setting$maxit.in,
                warm = warmstart, B0 = old.B)
    bdist = sum(sum(abs(B - old.B)))
    if (!is.numeric(B))
      B <- matrix(0, p, q)
    old.B = B
    if ((bdist < setting$tol.out * mab) | (k > setting$maxit.out))
      break
    cat("Outer iterations: ", k, "\n")
    cat("lambda1=", lam1, "\n")
    cat("lambda2=", lam2, "\n")
    cat("S_lambda=", as.matrix(samp.cov), "\n")
  }

  if(setting$silent ==FALSE) cat("Total outer iterations for cvm :
     ", k, "\n")
  return(list(gamma=old.B, theta=old.om))
}
```

`cvm.bic` tunes the lasso parameters with AIC, BIC or extended BIC, given a range of tuning parameters.

```
cvm.bic <- function (stimade, model = c("ar1", "ar2"), T=T, n=n,
                     p=p, q=q, time=time, tps=tps, xty = xty, xtx =
                        xtx, yty = yty, xtxt = xtxt, xtx2 = xtx2,
                        yty2 = yty2, lam1=lam1, lam2=lam2,
                        optimality = c("NULL", "bic","bic_ext","aic"
                        ), setting=setting)
{
  lam.vec.1 = lam1
  lam.vec.2 = lam2
  lamR = length(lam.vec.1) * length(lam.vec.2)
  BICh = matrix(NA, lamR, 1)
  lam1h = matrix(NA, lamR, 1)
  lam2h = matrix(NA, lamR, 1)
  uv <- 0
```

```
for (u in 1:length(lam.vec.1)) {
  for (v in 1:length(lam.vec.2)) {
    uv <- uv + 1
    outlasso_s <- compute.cvm(stimade=stimade, model = model,
                  T = T, n = n, p = p, q = q,
                  time=time, tps=tps, xty = xty,
                  xtx = xtx, yty = yty, xtxt = xtxt,
                  xtx2 = xtx2, yty2 = yty2,
                  lam1 = lam.vec.1[u], lam2 = lam.vec.2[v],
                  optimality = "NULL", setting = setting)
    P0 = outlasso_s$theta
    PB = outlasso_s$gamma
    WS = (yty - t(xty) %*% PB - t(PB) %*% xty + t(PB) %*%
            xtx %*% PB)/(n * T)
    lik1 = determinant(P0)$modulus[1]
    lik2 <- sum(diag(P0 %*% WS))
    diag(P0) = 0
    pd0 = sum(sum(P0 != 0))
    pdB = sum(sum(PB != 0))
    LLk <- (n * T/2) * (lik1 - lik2)
    LLk0 <- (n * T/2) * (-lik2)
    if (optimality == "bic") {
      BICh[uv, 1] <- -2 * LLk + (log(n * T)) * (pd0/2 +
                                                q + pdB)
    }
    else if (optimality == "bic_ext") {
      BICh[uv, 1] <- -2 * LLk + (log(n * T)) * (pd0/2 +
                      q + pdB) + (pd0/2 + q + pdB) * 4 * 0.5 *
                      log(q + p)
    }
    else if (optimality == "aic") {
      BICh[uv, 1] <- -2 * LLk + 2 * (pd0/2 + q + pdB)
    }
    lam1h[uv, 1] <- lam.vec.1[u]
    lam2h[uv, 1] <- lam.vec.2[v]
  }
```

```
  }
  res.lasso <- cbind(lam1h, lam2h, BICh)
  bicid <- which.min(res.lasso[, 3])
  lam1.opt <- res.lasso[bicid, 1]
  lam2.opt <- res.lasso[bicid, 2]
  bicm <- res.lasso[bicid, 3]
  tmp.out = compute.cvm(stimade=stimade, model = model, T = T,
                        n = n, p = p, q = q, time=time, tps=tps,
                        xty = xty, xtx = xtx, yty = yty,
                        xtxt = xtxt, xtx2 = xtx2,
                        yty2 = yty2, lam1 = lam1.opt,
                        lam2 = lam2.opt, optimality = "NULL",
                        setting = setting)
  best.B = tmp.out$gamma
  best.theta = tmp.out$theta
  d.gamma <- tmp.out$gamma
  diag(d.gamma) <- 0
  s.gamma = sum(abs(d.gamma) > 0)/(p^2)
  d.theta <- tmp.out$theta
  diag(d.theta) <- 0
  s.theta = (0.5 * sum(abs(d.theta) > 0))/(0.5 * q * (q - 1))
  tun.ic <- res.lasso
  lam1s <- lam.vec.1
  lam2s <- lam.vec.2
  min.ic <- bicm
  colnames(tun.ic) <- c("Lambda1", "Lambda2", "IC")
  return(list(gamma = best.B, theta = best.theta,
              lam1.opt = lam1.opt, lam2.opt = lam2.opt,
              lam1.seq = lam1s, lam2.seq = lam2s, min.ic = min.ic,
              tun.ic = tun.ic, s.gamma = s.gamma,
              s.theta = s.theta))
}
```

The following function can be directly applied by the user to a longitudinal dataset, giving a range of tuning parameters. One has to choose:

- the model selection criterion among BIC, AIC or extended BIC;

- the nonparametric estimator for CDFs among empirical cumulative distribution function (ecdf), time-varying (tvkde) or non time-varying (kde) kernel estimator.

```
cvm <- function (data = data, lam1 = NULL, lam2 = NULL,
                 nlambda = NULL, model = c("ar1", "ar2"),
                 optimality = c("NULL", "bic", "bic_ext", "aic"),
                 npest=c("tvkde", "kde", "ecdf"),
                 control = list())
{
  npest = match.arg(npest)
  require(longitudinal)
  if (is.longitudinal(data) == TRUE) {
    n_obs = get.time.repeats(data)$repeats[[1]]
    tps = get.time.repeats(data)$time
    p_num = dim(data)[2]
    time = dim(data)[1]/n_obs
    stimade <- fp(data, npest)
  }
  else {
    cat("Data format is not longitudinal.", "\n")
  }
  model = match.arg(model)
  xy.z <- transf.cvm(stimade, p_num, time, n_obs)
  data.prep <- pre.cvm(xy.z = xy.z, time = time, model = model)
  xty <- data.prep$xty
  xtx <- data.prep$xtx
  yty <- data.prep$yty
  xtxt <- data.prep$xtxt
  xtx2 <- data.prep$xtx2
  yty2 <- data.prep$yty2
  if (model=="ar1")
  {
    T <- length(get.time.repeats(data)$time) - 1
  }
  else if (model=="ar2")
```

```
{
  T <- length(get.time.repeats(data)$time) - 2
}
q <- p_num
optimality = match.arg(optimality)
nobic = (length(lam1) + length(lam2) == 2)
doms = (length(lam1) + length(lam2) > 2)
if (!is.list(control))
  stop("control is not a list")
setting <- list(maxit.out = 5, maxit.in = 50, tol.out = 1e-04,
                silent = TRUE)
nmsSetting <- names(setting)
setting[(nms <- names(control))] <- control
if (length(noNms <- nms[!nms %in% nmsSetting]))
  warning("unknow names in control: ", paste(noNms, collapse = ",
      "))
if (nobic != 2 & optimality == "nosel")
  stop("Specify positive scalar values for the tuning parameters"
     )
if (doms > 2 & optimality != "nosel")
  stop("Specify vector of positive decreasing values for the
     tuning parameters")
doNULL = nobic & (optimality == "NULL")
dobic = doms & (optimality == "bic")
dobic1 = doms & (optimality == "bic_ext")
dobic3 = doms & (optimality == "aic")
gamma = NULL
theta = NULL

if (doNULL) {
  tmp.out = compute.cvm(stimade=stimade, model = model, T = T,
                        n = n_obs, p = p_num, q = q, time=time,
                        tps=tps, xty = xty, xtx = xtx, yty = yty,
                        xtxt = xtxt, xtx2 = xtx2, yty2 = yty2,
                        lam1 = lam1, lam2 = lam2,
                        optimality = "NULL", setting = setting)
```

```
}
else if (dobic) {
  tmp.out = cvm.bic(stimade=stimade, model = model, T = T,
                    n = n_obs, p = p_num, q = q, time=time,
                    tps=tps, xty = xty, xtx = xtx, yty = yty,
                    xtxt = xtxt, xtx2 = xtx2, yty2 = yty2,
                    lam1 = lam1, lam2 = lam2, optimality = "bic",
                    setting = setting)
}
else if (dobic1) {
  tmp.out = cvm.bic(stimade=stimade, model = model, T = T,
                    n = n_obs, p = p_num, q = q, time=time,
                    tps=tps, xty = xty, xtx = xtx, yty = yty,
                    xtxt = xtxt, xtx2 = xtx2, yty2 = yty2,
                    lam1 = lam1, lam2 = lam2,
                    optimality = "bic_ext", setting = setting)
}
else if (dobic3) {
  tmp.out = cvm.bic(stimade=stimade, model = model, T = T,
                    n = n_obs, p = p_num, q = q, time=time,
                    tps=tps, xty = xty, xtx = xtx, yty = yty,
                    xtxt = xtxt, xtx2 = xtx2, yty2 = yty2,
                    lam1 = lam1, lam2 = lam2, optimality = "aic",
                    setting = setting)
}
gamma = tmp.out$gamma
gamma = gamma * (1 * (abs(gamma) > 0.01))
theta = tmp.out$theta
theta = theta * (1 * (abs(theta) > 0.01))
lam1.opt = tmp.out$lam1.opt
lam2.opt = tmp.out$lam2.opt
lam1.seq = tmp.out$lam1.seq
lam2.seq = tmp.out$lam2.seq
s.gamma = tmp.out$s.gamma
s.theta = tmp.out$s.theta
tun.ic = tmp.out$tun.ic
```

```
  min.ic = tmp.out$min.ic
  if (model == "ar1") {
    colnames(gamma) <- colnames(data)
  }
  else if (model == "ar2") {
    colnames(gamma) <- colnames(data)
    rownames(gamma) <- c(colnames(data), colnames(data))
  }
  colnames(theta) <- rownames(theta) <- colnames(data)
  out = list(gamma = gamma, theta = theta, lam1.opt = lam1.opt,
             lam2.opt = lam2.opt, lam1.seq = lam1.seq, lam2.seq =
                lam2.seq,
             min.ic = min.ic, tun.ic = tun.ic, s.gamma = s.gamma,
             s.theta = s.theta)
  class(out) = "sparse.tscgm"
  return(out)
}
```

The following function generates sparse autoregressive and covariance matrices as in algorithms (2) and (3).

```
gen.sparse.tg <- function(n.var, sparsity=NULL)
{
  if(is.null(sparsity))
    sparsity <- 1/n.var
  # Theta
  out1 <- matrix(NA, nrow=n.var, ncol=n.var)
  for(i in 1:(n.var))
  {
    for(j in 1:n.var)
    {
     tmp  <- rbinom(1, size=1, prob=sparsity)
     if(tmp==0)
       out1[i, j] <- tmp
     else
     {
       tmp2 <- rbinom(1, size=1, prob=.5)
```

```
      if(tmp2==0)
        out1[i, j] <- runif(1, -1, -.5)
      else
        out1[i, j] <- runif(1, .5, 1)
    }
  }
}
for(k in 1:n.var)
{
  if(any(!(out1[k,-k]==rep(0, n.var - 1))))
    out1[k,-k] <- out1[k, -k] / (1.5*sum(abs(out1[k, -k])))
}
out1 <- Matrix::forceSymmetric(out1)
diag(out1) <- 1
sigma.old <- solve(out1)
sigma.new <- cov2cor(sigma.old)
prec <- solve(sigma.new)
prec <- round(prec, digits=8)

# Gamma
vm <- min(abs(prec[prec!=0]))
out2 <- matrix(NA, nrow=n.var, ncol=n.var)
for(i in 1:(n.var))
{
  for(j in 1:n.var)
  {
    tmp  <- rbinom(1, size=1, prob=sparsity)
    if(tmp==0)
      out2[i, j] <- tmp
    else
    {
      tmp2 <- rbinom(1, size=1, prob=.5)
      if(tmp2==0)
        out2[i, j] <- runif(1, -1, -vm)
      else
        out2[i, j] <- runif(1, vm, 1)
```

```
      }
    }
  }
  gamma <- out2
  diag(gamma) <- .1
  list(theta=as.matrix(prec), gamma=gamma)
}
```

The following function simulates exponential or log-normal longitudinal data as described in section (5.1). If normal data are needed, it is sufficient to pick the object `zt`; otherwise, one has to choose between exponential or lognormal distribution through the argument `distr`. If exponential data with time-varying rate are needed, one needs to set `rate=t/2`, for example.

```
sim.nonn <- function (time = time, n.obs = n.obs, n.var = n.var,
                      seed = NULL, prob0 = NULL, prec = NULL,
                      gamma1 = NULL, distr=c("exponential",
                      "lognormal"), ...)
{
  distr = match.arg(distr)
  t = time
  n = n.obs
  d = n.var
  if (is.numeric(seed))
    r = 0
  else {
    seed = 123
    r = round(runif(1), 4) * 10000
  }
  mu <- rep(0, d)
  true_theta <- prec
  sigma1 <- round(solve(true_theta), digits=7)
  true_gamma <- gamma1
  B11 <- true_gamma
  varmarg <- sigma1 + B11%*%sigma1%*%t(B11)
  cdiag <- diag(varmarg)
  xtn <- array(NA, c(t, d, n))
```

```
  xtt <- array(NA, c(t, d, 1))
  for (i in 1:n) {
    z0 <- mvtnorm::rmvnorm(1, mu, sigma1, method = "svd")
    x0 <- rep(NA, d)
    for(k in 1:d)
    {
      z0[k] <- z0[k]/(sqrt(cdiag[k]))
      if(distr=="exponential")
        x0[k] <- qexp(pnorm(z0[k]), ...)
      else if(distr=="lognormal")
        x0[k] <- qlnorm(pnorm(z0[k]), ...)
    }
    for (j in 1:t) {
      et <- mvtnorm::rmvnorm(1, mu, sigma1, method = "svd")
      zt <- z0 %*% B11 + et
      xt <- rep(NA, d)
      for(k in 1:d)
      {
        zt[k] <- zt[k]/(sqrt(cdiag[k]))
        if(distr=="exponential")
          xt[k] <- qexp(pnorm(zt[k]), ...)
        else if(distr=="lognormal")
          xt[k] <- qlnorm(pnorm(zt[k]), ...)
      }
      xtt[j, , ] <- xt
      z0 <- zt
    }
    xtn[, , i] <- round(xtt, 3)
  }
  xy = matrix(aperm(xtn, c(3, 1, 2)), ncol = d)
  data1 <- as.longitudinal(xy, repeats = n)
  return(list(data1 = data1, theta = true_theta,
              gamma = true_gamma, sigma = sigma1))
}
```

References

Abegaz, Fentaw and Wit, Ernst (2013). Sparse time series chain graphical models for reconstructing genetic networks. Biostatistics: 586–599.

— (2015). Copula Gaussian graphical models with penalized ascent Monte Carlo EM algorithm. Statistica Neerlandica, 69 (4): 419–441.

Akaike, Hirotogu (1998). "Information theory and an extension of the maximum likelihood principle". In: *Selected Papers of Hirotugu Akaike*. Springer, pp. 199–213.

Azzalini, Adelchi (1981). A note on the estimation of a distribution function and quantiles by a kernel method. Biometrika, 68 (1): 326–328.

Azzalini, Adelchi and Scarpa, Bruno (2012). Data analysis and data mining: An introduction. OUP USA.

Breheny, Patrick and Huang, Jian (2011). Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. The annals of applied statistics, 5 (1): 232.

Burman, Prabir, Chow, Edmond, and Nolan, Deborah (1994). A cross-validatory method for dependent data. Biometrika, 81 (2): 351–358.

Chen, Xiaohong and Fan, Yanqin (2006). Estimation and model selection of semiparametric copula-based multivariate dynamic models under copula misspecification. Journal of econometrics, 135 (1): 125–154.

Dahlhaus, Rainer and Eichler, Michael (2003). Causality and graphical models in time series analysis. Oxford Statistical Science Series: 115–137.

Dobra, Adrian and Lenkoski, Alex (2009). *Copula Gaussian graphical models*. Tech. rep. Technical report, Department of Statistics, University of Washington.

Efron, Bradley, Hastie, Trevor, Johnstone, Iain, Tibshirani, Robert, et al. (2004). Least angle regression. The Annals of statistics, 32 (2): 407–499.

Fan, Jianqing and Li, Runze (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American statistical Association, 96 (456): 1348–1360.

Fan, Jianqing, Feng, Yang, and Wu, Yichao (2009). Network exploration via the adaptive LASSO and SCAD penalties. The annals of applied statistics, 3 (2): 521.

Foygel, Rina and Drton, Mathias (2010). "Extended Bayesian information criteria for Gaussian graphical models". In: *Advances in neural information processing systems*, pp. 604–612.

Friedman, Jerome, Hastie, Trevor, and Tibshirani, Robert (2001). The elements of statistical learning. Vol. 1. Springer series in statistics New York.

Friedman, Jerome, Hastie, Trevor, Höfling, Holger, Tibshirani, Robert, et al. (2007). Pathwise coordinate optimization. The Annals of Applied Statistics, 1 (2): 302–332.

Friedman, Jerome, Hastie, Trevor, and Tibshirani, Robert (2008). Sparse inverse covariance estimation with the graphical lasso. Biostatistics, 9 (3): 432–441.

Friedman, Jerome, Hastie, Trevor, and Tibshirani, Rob (2010). Regularization paths for generalized linear models via coordinate descent. Journal of statistical software, 33 (1): 1.

Gao, Wei and Tian, Zheng (2010). Latent ancestral graph of structure vector autoregressive models. Journal of Systems Engineering and Electronics, 21 (2): 233–238.

Genest, Christian, Ghoudi, Kilani, and Rivest, L-P (1995). A semiparametric estimation procedure of dependence parameters in multivariate families of distributions. Biometrika: 543–552.

Greenham, Kathleen and McClung, C Robertson (2015). Integrating circadian dynamics with physiological processes in plants. Nature Reviews. Genetics, 16 (10): 598.

Grzegorczyk, Marco and Husmeier, Dirk (2010). Improvements in the reconstruction of time-varying gene regulatory networks: dynamic programming and regularization by information sharing among genes. Bioinformatics, 27 (5): 693–699.

— (2011). Non-homogeneous dynamic Bayesian networks for continuous data. Machine Learning, 83 (3): 355–419.

Harvey, Andrew and Oryshchenko, Vitaliy (2012). Kernel density estimation for time series data. International Journal of Forecasting, 28 (1): 3–14.

Hoff, Peter D (2007). Extending the rank likelihood for semiparametric copula estimation. The Annals of Applied Statistics: 265–283.

Hsieh, Cho-Jui, Sustik, Mátyás A, Dhillon, Inderjit S, and Ravikumar, Pradeep (2014). QUIC: quadratic approximation for sparse inverse covariance estimation. Journal of Machine Learning Research, 15 (1): 2911–2947.

Lauritzen, Steffen L (1996). Graphical models. Vol. 17. Clarendon Press.

Liu, Han, Lafferty, John, and Wasserman, Larry (2009). The nonparanormal: Semiparametric estimation of high dimensional undirected graphs. Journal of Machine Learning Research, 10 (Oct): 2295–2328.

Liu, Han, Han, Fang, Yuan, Ming, Lafferty, John, Wasserman, Larry, et al. (2012). High-dimensional semiparametric Gaussian copula graphical models. The Annals of Statistics, 40 (4): 2293–2326.

Loh, Po-Ling and Wainwright, Martin J (2013). "Regularized M-estimators with nonconvexity: Statistical and algorithmic theory for local optima". In: *Advances in Neural Information Processing Systems*, pp. 476–484.

Nelsen, Roger B (2007). An introduction to copulas. Springer Science & Business Media.

Parzen, Emanuel (1962). On estimation of a probability density function and mode. The annals of mathematical statistics, 33 (3): 1065–1076.

Racine, Jeff (2000). Consistent cross-validatory model-selection for dependent data: hv-block cross-validation. Journal of econometrics, 99 (1): 39–61.

Rosenblatt, Murray et al. (1956). Remarks on some nonparametric estimates of a density function. The Annals of Mathematical Statistics, 27 (3): 832–837.

Rothman, Adam J, Levina, Elizaveta, and Zhu, Ji (2010). Sparse multivariate regression with covariance estimation. Journal of Computational and Graphical Statistics, 19 (4): 947–962.

Schwarz, Gideon et al. (1978). Estimating the dimension of a model. The annals of statistics, 6 (2): 461–464.

Scott, David W (2015). Multivariate density estimation: theory, practice, and visualization. John Wiley & Sons.

Sima, Chao, Hua, Jianping, and Jung, Sungwon (2009). Inference of gene regulatory networks using time-series data: a survey. Current genomics, 10 (6): 416–429.

Sklar, M (1959). Fonctions de répartition à n dimensions et leurs marges. Université Paris 8.

Tibshirani, Ryan (2015). *Sparsity and the Lasso*.

Vujačić, Ivan, Abbruzzo, Antonino, and Wit, Ernst (2015). A computationally fast alternative to cross-validation in penalized Gaussian graphical models. Journal of Statistical Computation and Simulation, 85 (18): 3628–3640.

Yang, Yuhong (2005). Can the strengths of AIC and BIC be shared? A conflict between model indentification and regression estimation. Biometrika, 92 (4): 937–950.

Yin, Jianxin and Li, Hongzhe (2011). A sparse conditional gaussian graphical model for analysis of genetical genomics data. The annals of applied statistics, 5 (4): 2630.

Yuan, Ming and Lin, Yi (2007). Model selection and estimation in the Gaussian graphical model. Biometrika, 94 (1): 19–35.

Zou, Hui and Hastie, Trevor (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67 (2): 301–320.

Acknowledgements

I would like to express my gratitude to my supervisors, Ernst Wit and Alessandra Brazzale, who guided me through this project making it a fruitful as well as enjoyable experience. I am really thankful to them for giving me the possibility to work on this thesis in Groningen, the Netherlands, and participate to many academic activities which made me broaden my perspective.

I will never be grateful enough to my family, Franca, Celestino and Fabio, for supporting me through all this years and for always having done everything they could for me.

There are no words to describe how much my friends are vital to my existence. Therefore, I will just say thanks to Caterina, Roberta, Silvia, Alvise and Andrée, for coloring every single day of mine.

Thanks to my beloved Tito mates: Fabio, Federica, Francesco, Martina and Valentina, for being wonderful housemates before, and even better friends after leaving the house.

Thanks to Mattia, Alice and Edoardo, with whom I shared the tutoring experience, for revealing themselves wonderful and incredibly inspiring friends.