







cglasso: An R Package for Conditional Graphical Lasso Inference with Censored and Missing Values

Luigi Augugliaro 
University of Palermo

Gianluca Sottile 
University of Palermo

Ernst C. Wit 
Università della Svizzera italiana

Veronica Vinciotti 
University of Trento

Abstract

Sparse graphical models have revolutionized multivariate inference. With the advent of high-dimensional multivariate data in many applied fields, these methods are able to detect a much lower-dimensional structure, often represented via a sparse conditional independence graph. There have been numerous extensions of such methods in the past decade. Many practical applications have additional covariates or suffer from missing or censored data. Despite the development of these extensions of sparse inference methods for graphical models, there have been so far no implementations for, e.g., conditional graphical models.

Here we present the general-purpose package **cglasso** for estimating sparse conditional Gaussian graphical models with potentially missing or censored data. The method employs an efficient expectation-maximization estimation of an ℓ_1 -penalized likelihood via a block-coordinate descent algorithm. The package has a user-friendly data manipulation interface. It estimates a solution path and includes various automatic selection algorithms for the two ℓ_1 tuning parameters, associated with the sparse precision matrix and sparse regression coefficients, respectively. The package pays particular attention to the visualization of the results, both by means of marginal tables and figures, and of the inferred conditional independence graphs.

This package provides a unique and computational efficient implementation of a conditional Gaussian graphical model that is able to deal with the additional complications of missing and censored data. As such it constitutes an important contribution for empirical scientists wishing to detect sparse structures in high-dimensional data.

Keywords: cglasso, conditional Gaussian graphical models, glasso, high-dimensionality, sparsity, censoring, missing data.

1. Introduction

By being able to include covariates inside graphical models, the applicability of such *conditional graphical models* to fields such as genomics and information retrieval has increased substantially. Moreover, the increasing availability of statistical methods for sparse inference have allowed for efficient approaches in large data settings. For example, in the context of genomics, [Yin and Li \(2011\)](#) advocate the use of conditional graphical models to account for the effect of genetic variants as predictors x on the gene expression, i.e., the responses y . The idea is that the high correlation among gene expression may be explained by the effects of shared genetic variants.

Many approaches have considered the case of conditional Gaussian graphical models and have proposed efficient algorithms for inference. [Rothman, Levina, and Zhu \(2010\)](#) independently proposed the same estimator as [Yin and Li \(2011\)](#), based on an extension of the popular graphical lasso (glasso) estimator of [Friedman, Hastie, and Tibshirani \(2008\)](#). [Wang \(2015\)](#), on the other hand, proposed a new algorithm based on the idea of decomposing the initial problem into a series of simpler conditional problems involving, at each step, the conditional log-likelihood function of each response variable given all the remaining variables. Recently, other approaches have proposed two-steps procedures for fitting a sparse conditional Gaussian graphical model. [Li, Chun, and Zhao \(2012\)](#) propose to first use an initial non-sparse estimator for the conditional covariance matrix Σ using the theory of reproducing kernel Hilbert spaces, and then, in the second step of the procedure, the resulting estimate is used to formulate a glasso-type problem for the estimation of the inverse covariance matrix Θ . In contrast to this, the two-steps procedure proposed in [Yin and Li \(2013\)](#) uses, in the first step, an ℓ_1 -penalized multivariate linear regression model with independent errors for estimating the sparse coefficient regression matrix, whereas, in the second step, the precision matrix is estimated using the standard glasso estimator. Finally, [Chen, Ren, Zhao, and Zhou \(2016\)](#) proposed a two-step procedure where the scaled lasso is used in each step to select the amount of sparsity.

In large data settings, it is not uncommon to encounter various artifacts that violate the fully observed Gaussian scenario, such as missing data and saturation effects, also known as censoring. Censoring can occur when the measuring device has a limit of detection, either at the lower or the upper end. This is for example the case of data generated by reverse transcription quantitative polymerase chain reaction (RT-qPCR), a popular technology for gene expression profiling ([Derveaux, Vandesompele, and Hellemans 2010](#)). Missingness is quite common for large, automatically retrieved data, where certain entries are deemed unreliable and automatically discarded. [Augugliaro, Abbruzzo, and Vinciotti \(2020a\)](#) have looked closely at inference of a Gaussian graphical model in the case of censoring, whereas [Augugliaro, Sottile, and Vinciotti \(2020b\)](#) have extended this to the more general case of a conditional Gaussian graphical model, as implemented in the R ([R Core Team 2022](#)) package `cglasso` ([Augugliaro, Sottile, Wit, and Vinciotti 2023](#)). Missing-at-random in the context of (conditional) Gaussian graphical models has been considered by [Städler and Bühlmann \(2012\)](#) and has also been integrated within this package through the development of a unifying computationally efficient algorithm.

1.1. Related implementations

Although many packages are available to estimate sparse Gaussian graphical models, none of

them support the inclusion of covariate information. The R package **glasso** (Friedman, Hastie, and Tibshirani 2019) fits the graphical lasso model using the block-coordinate descent algorithm proposed in Friedman *et al.* (2008). An efficient implementation of this algorithm is also available in the R package **glassoFast** (Sustik, Calderhead, and Clavel 2018). In contrast to this, the package **QUIC** (Hsieh, Sustik, Dhillon, and Ravikumar 2011) implements the inferential routines via Newton’s method and coordinate descent, with the package **BigQuic** (Kunji 2022) providing an implementation specifically built for working with large datasets. The package **huge** (Jiang *et al.* 2021) provides a general framework for high-dimensional undirected graph estimation. It integrates data preprocessing, neighborhood screening via the method proposed in Meinshausen and Bühlmann (2006), penalized estimation via the graphical lasso model, and model selection techniques into a single pipeline. It also implements the nonparanormal transformation of Liu, Lafferty, and Wasserman (2009) for relaxing the normality assumption. Finally, there are packages that perform Bayesian sparse inference for graphical models. For example, the package **BDgraph** (Mohammadi and Wit 2019) provides statistical tools for Bayesian structure learning in undirected graphical models for continuous, discrete, and mixed data, whereas **BayesianGLasso** (Trainor and Wang 2017) implements the block Gibbs sampler for the Bayesian graphical lasso introduced in Wang (2012).

Particularly in the Bayesian inferential setting, there are also packages that can account for missing data, mostly of the missing-at-random type. The R package **bnlearn** (Scutari 2017) supports incomplete data with missing data, while implementing key algorithms covering all stages of Bayesian network modelling: data pre-processing, structure learning combining data and expert/prior knowledge, parameter learning, and inference (including causal inference via do-calculus). The **bnstruct** package (Franzin, Sambo, and di Camillo 2017) also provides algorithms for Bayesian network structure learning from data with missing values. Similarly, **BGGM** (Williams and Mulder 2020) can be used to fit Bayesian Gaussian graphical models and can handle datasets with missing data. However, **BGGM** has been built specifically for social and behavioral scientists and it does not support high-dimensional data. In contrast to this, **sparsebn** (Aragam, Gu, and Zhou 2019) is an R package for learning sparse Bayesian networks and other graphical models from high-dimensional data via sparse regularization. In the frequentist, but low-dimensional, setting, the **monomvn** package (Gramacy, Moler, and Turlach 2022) implements maximum likelihood estimation of the mean and covariance matrix of multivariate normal distributed data with a monotone missingness pattern. Moreover, using shrinkage regressions, the available function can handle an arbitrary amount of missing data.

Our proposed package complements this literature, by providing sparse inference for Gaussian graphical models, under the presence of additional covariate information and with possibly missing data, either generated at random or via a censoring process.

1.2. Overview of the paper

The remaining part of the paper is structured as follows. In Section 2 we introduce the conditional Gaussian graphical model and describe a sparse inference procedure for such models that is able to deal with data with both censored and missing-at-random values. In Section 3, we present the main routines implemented in the **cglasso** package through the analysis of a genomic dataset on multiple myeloma (Section 3.1) and a dataset on megakaryocyte-erythroid progenitors (Section 3.2). In Section 3.3 we present also a function for simulating data from a conditional Gaussian graphical model and two plotting method functions designed to evaluate

the statistical assumptions underlying the proposed method. The remaining sections of the paper are devoted to the technical description of the implemented functions. In particular, the functions for data manipulations are described in Section 4; routines for model fitting and model selection are presented in Section 5 and Section 6, respectively, whereas in Section 7 we discuss how to plot and analyze the fitted graphs. Finally, in Section 8 we draw some conclusions.

2. Methodological background

2.1. Conditional Gaussian graphical model

Conditional Gaussian graphical models, also known as covariate adjusted Gaussian graphical models, are a class of conditional probabilistic graphical models used to encode the dependence structure among the elements of a set of random variables conditional on a second set of random variables (Lafferty, McCallum, and Pereira 2001). Formally, let $y = (y_1, \dots, y_p)^\top$ and $x = (x_1, \dots, x_q)^\top$ be p - and q -dimensional random vectors, respectively, and let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with vertex set $\mathcal{V} = \{1, \dots, p\}$, indexing only the entries in y , and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, where $(h, k) \in \mathcal{E}$ iff there is a directed edge from the vertex h to k in \mathcal{G} . Suppose that the distribution of y conditional on x is a multivariate Gaussian distribution with probability density function defined as follows:

$$\phi(y \mid x; B, \Theta) = (2\pi)^{-\frac{p}{2}} |\Theta|^{\frac{1}{2}} \exp \left\{ -\frac{1}{2} (y - B^\top x)^\top \Theta (y - B^\top x) \right\}, \quad (1)$$

where, with a little abuse of notation, we let $x = (1, x^\top)^\top$ be the vector of predictors and $B = (\beta_0, \beta^\top)^\top$ the $(q+1) \times p$ regression coefficient matrix, with distinct columns to ensure model identifiability. The model in (1) assumes that the predictors affect the distribution of the response variables y only via the p conditional expected values and through a linear function, that is:

$$\mathbf{E}(y \mid x) = B^\top x, \quad \text{VAR}(y \mid x) = \Sigma.$$

The inverse of the variance matrix, denoted by $\Theta = (\theta_{hk})$, is called the precision matrix and its entries have a one-to-one correspondence with the partial correlation coefficients. Using standard results about the multivariate Gaussian distribution, it is possible to show that y_h and y_k are conditionally independent given x and all the remaining variables in y iff the corresponding partial correlation coefficient is zero (Lauritzen 1996). This remarkable property of the multivariate Gaussian distribution gives rise in a natural way to the notion of a conditional Gaussian graphical model, which is based on the idea of relating the factorization of the density (1) to the topological structure of the associated undirected graph \mathcal{G} .

Definition 1 *The triplet $\{y, \phi(y \mid x; B, \Theta), \mathcal{G}\}$ is said to be a conditional Gaussian graphical model if the density $\phi(y \mid x; B, \Theta)$ factorizes according to the conditional independence graph \mathcal{G} , whereby $\theta_{ij} = 0$ iff $(i, j) \notin \mathcal{E}$.*

This definition shows that estimating the edge set of the undirected graph \mathcal{G} is formally equivalent to finding non-zero entries in Θ . In a high-dimensional setting, in order to make statistical inference possible, we further assume that model (1) has a sparse representation,

meaning that only a few regression coefficients and partial regression coefficients are different from zero [Fan and Lv \(2010\)](#). This will motivate the development of a sparse inferential procedure, which will allow for simultaneous parameter estimation and model selection.

The **cglasso** package is able to deal with sparse inference of a conditional Gaussian graphical model under censoring and missing-at-random structures, which occur frequently in real data. Dealing with these artifacts can be computationally challenging in problems of high-dimensionality (in y , in x , or in both). In order to address these issues, we propose a doubly penalized estimator for a conditional Gaussian graphical model by extending [Augugliaro *et al.* \(2020b\)](#) to account simultaneously for censoring and missingness-at-random mechanisms.

2.2. Sparse inference of a conditional Gaussian graphical model

Let (y_i, x_i) with $i = 1, \dots, n$ be a set of n independent observations. We consider a number of scenarios. For observations $i \in \mathcal{O}$, in which the observation vector y_i is fully observed, the contribution to the log-likelihood is

$$\ell_i(B, \Theta) = \log \phi(y_i \mid x_i; B, \Theta).$$

For observations $i \in \mathcal{C}$, in which some entries j of y_i are censored, $j \in c_i$, either from below or from above, the contribution to the likelihood is given by the multi-dimensional integral across the censored variables,

$$\ell_i(B, \Theta) = \log \int_{D_{c_i}} \phi(y_i \mid x_i; B, \Theta) dy_{ic_i},$$

where the region $D_{c_i} = \prod_{j \in c_i} D_{ij}$ is the censoring region, with $D_{ij} = (-\infty, l_j)$ if $y_{ij} \leq l_j$ (censored from below) or $D_{ij} = (u_j, \infty)$ if $y_{ij} \geq u_j$ (censored from above). Finally, if some entries y_{ij} are missing-at-random, then it is possible to extend the definition of the censoring region to encompass such missingness in the likelihood. In particular, $D_{ij} = \mathbb{R}$ for these cases. Considering all possible cases, the relevant average observed log-likelihood function is given by

$$\bar{\ell}(B, \Theta) = \frac{1}{n} \sum_{i=1}^n \ell_i(B, \Theta). \quad (2)$$

Under a high-dimensional setting, that is $\min\{p, q\} > n$, inference about B and Θ is carried out under the assumption that these matrices have a sparse structure. To this end, [Augugliaro *et al.* \(2020b\)](#) propose to estimate the parameters of a conditional Gaussian graphical model by maximizing a new objective function whereby two specific lasso-type penalty functions are added to the average observed log-likelihood. The resulting estimator is defined as follows:

$$\{\hat{B}, \hat{\Theta}\} = \arg \max \bar{\ell}(B, \Theta) - \lambda \sum_{k=1}^p \theta_{kk} \|\beta_k\|_1 - \rho \|\Theta\|_1^- \quad (3)$$

where β_k denotes the k th column of β , $\|\beta_k\|_1 = \sum_{h,k} |\beta_{hk}|$ and $\|\Theta\|_1^- = \sum_{h \neq k} |\theta_{hk}|$. Like in the standard conditional glasso (cglasso) estimator ([Yin and Li 2011](#)), the tuning parameter λ is used to control the amount of sparsity in the estimated regression coefficient matrix whereas ρ is devoted to control the sparsity in $\hat{\Theta} = (\hat{\theta}_{hk})$ and, consequently, in the corresponding estimated conditional independence graph $\hat{\mathcal{G}} = \{\mathcal{V}, \hat{\mathcal{E}}\}$, where $\hat{\mathcal{E}} = \{(h, k) : \hat{\theta}_{hk} \neq 0\}$. When ρ

is sufficiently large, some $\hat{\theta}_{hk}$ are shrunken to zero resulting in the removal of the corresponding link in $\hat{\mathcal{G}}$; on the other hand, when ρ is equal to zero and the sample size is large enough the estimator $\hat{\Theta}$ coincides with the maximum likelihood estimator of the precision matrix, which implies a fully connected conditional independence graph.

2.3. Algorithm for fitting a conditional graphical lasso

We propose a unifying algorithm for inference of a sparse conditional Gaussian graphical model that can accommodate both the case of censoring (Augugliaro *et al.* 2020b), missingness-at-random (Städler and Bühlmann 2012) as well as the high-dimensionality of the data. To this aim, we propose an expectation-maximization (EM) algorithm for maximizing the penalized log-likelihood. In general, the EM algorithm is based on the idea of repeating expectation and maximization steps, until a convergence criterion is met. For the sake of simplicity, in the remaining part of this section, we use $\vartheta = \{B, \Theta\}$ to indicate the full set of parameters and $\hat{\vartheta}$ to denote their current estimate inside the EM. Moreover, r_{ik} indicates whether y_{ik} is observed ($r_{ik} = 0$) or not ($r_{ik} \neq 0$), with the latter case including both censoring and missingness.

Since the complete probability density function is a member of the regular exponential family, the E-step consists in computing two quantities. First, the imputed response matrix $\hat{Y} = (\hat{y}_{i,k})$ is obtained, whose entries are defined as:

$$\hat{y}_{i,k} = \begin{cases} y_{ik} & \text{if } r_{ik} = 0 \\ \mathbb{E}(y_{ik} \mid y_{ic_i} \in D_{c_i}, x_i; \hat{\vartheta}) & \text{otherwise,} \end{cases}$$

where $\mathbb{E}(\cdot \mid y_{ic_i} \in D_{c_i}, x_i; \hat{\vartheta})$ denotes the expected value operator computed with respect to the conditional Gaussian distribution of y_{ic_i} given $\{x_i, y_{io_i}\}$ and truncated over the region D_{c_i} . Secondly, it involves the matrix $\hat{C}_{yy} = \sum_{i=1}^n \hat{C}_i$, whose components have entries:

$$\hat{C}_{i,hk} = \begin{cases} y_{ih}y_{ik} & \text{if } r_{ih} = 0 \text{ and } r_{ik} = 0 \\ y_{ih}\mathbb{E}(y_{ik} \mid y_{ic_i} \in D_{c_i}, x_i; \hat{\vartheta}) & \text{if } r_{ih} = 0 \text{ and } r_{ik} \neq 0 \\ \mathbb{E}(y_{ih} \mid y_{ic_i} \in D_{c_i}, x_i; \hat{\vartheta})y_{ik} & \text{if } r_{ih} \neq 0 \text{ and } r_{ik} = 0 \\ \mathbb{E}(y_{ih}y_{ik} \mid y_{ic_i} \in D_{c_i}, x_i; \hat{\vartheta}) & \text{if } r_{ih} \neq 0 \text{ and } r_{ik} \neq 0. \end{cases}$$

From this, the working empirical covariance matrix is given by:

$$\hat{S}_{y|x}(B) = n^{-1}\{\hat{C}_{yy} - \hat{Y}^\top XB - (XB)^\top \hat{Y} + X^\top XB\}, \quad (4)$$

where X denotes the design matrix. Given the matrix (4), the M-step involves solving a new maximization problem obtained by replacing the objective function in definition (3), with the so-called penalized Q -function:

$$Q(B, \Theta) = \log \det \Theta - \text{tr}\{\Theta \hat{S}_{y|x}(B)\} - \lambda \sum_{k=1}^p \theta_{kk} \|\beta_k\|_1 - \rho \|\Theta\|_1^{-1}. \quad (5)$$

Since, for a fixed $\hat{\vartheta}$, the penalized Q -function in (5) is a bi-convex function in B and Θ , its maximization can be obtained by repeating two sub-steps until a convergence criterion is met. Given the current estimate of the precision matrix $\hat{\Theta}$, the first sub-step consists in estimating the regression coefficient matrix by solving the following maximization problem:

$$\min_B \text{tr}\{\hat{\Theta} \hat{S}_{y|x}(B)\} + \lambda \sum_{k=1}^p \hat{\theta}_{kk} \|\beta_k\|_1, \quad (6)$$

Algorithm 1 Pseudo-code of the EM algorithm.

- 1: Let \hat{B} and $\hat{\Theta}$ be the initial estimates
 - 2: **repeat**
 - 3: let $\hat{\vartheta} = \{\hat{B}, \hat{\Theta}\}$
 - 4: update \hat{Y} , \hat{C}_{yy} and compute $\hat{S}_{y|x}(B)$ ▷ E-Step
 - 5: **repeat** ▷ M-Step
 - 6: $\hat{B} = \arg \min_B \text{tr}\{\hat{\Theta}\hat{S}_{y|x}(B)\} + \lambda \sum_{k=1}^p \hat{\theta}_{kk} \|\beta_k\|_1$
 - 7: $\hat{\Theta} = \arg \max_{\Theta \succ 0} \log \det \Theta - \text{tr}\{\Theta\hat{S}_{y|x}(\hat{B})\} - \rho \|\Theta\|_1^-$
 - 8: **until** convergence criterion is met
 - 9: **until** convergence criterion is met
-

whereas, in the second sub-step, given \hat{B} , the precision matrix is estimated by solving the sub-problem:

$$\max_{\Theta \succ 0} \log \det \Theta - \text{tr}\{\Theta\hat{S}_{y|x}(\hat{B})\} - \rho \|\Theta\|_1^-. \quad (7)$$

Algorithm 1 reports the pseudo-code of the EM algorithm proposed in Augugliaro *et al.* (2020b).

While problem (7) is a standard graphical lasso problem that can be efficiently solved using, for example, the block-coordinate descent algorithm proposed in Friedman *et al.* (2008), problem (6) is similar to that studied by Rothman *et al.* (2010) and Yin and Li (2011) in the case of no censoring. However, instead of solving this problem through a cyclic coordinate descent algorithm, we use a more efficient and easy-to-implement block-coordinate descent algorithm based on the following identity: let B_k be the k th column of the matrix B and $\hat{S}_{y|x}(B_k)$ be the working empirical covariance matrix seen as function of B_k while the remaining columns are held fixed to the current estimates. Then the minimization problem

$$\min_{B_k} \text{tr}\{\hat{\Theta}\hat{S}_{y|x}(B_k)\} + \lambda \hat{\theta}_{kk} \|\beta_k\|_1,$$

is equivalent to

$$\min_{B_k} \frac{1}{n} \|\tilde{Y}_k - XB_k\|^2 + \lambda \|\beta_k\|_1,$$

where \tilde{Y}_k is a vector with i th element $\tilde{y}_{i,k} = \hat{y}_{i,k} + \hat{\theta}_{kk}^{-1} \sum_{h \neq k}^p \hat{\theta}_{hk} \{\hat{y}_{i,h} - \mathbf{x}_i^\top \hat{B}_h\}$. Algorithm 2 reports the pseudo-code of the method proposed to solve sub-problem (6), which is called the *multi-lasso algorithm*. Convergence to a global minimizer follows from the fact that the trace term in sub-problem (6) is a convex and differentiable function and the penalty function can be decomposed as a sum of p convex functions (Tseng 2001). Furthermore, we underline that the computational efficiency of the proposed multi-lasso algorithm can be improved using the results given in Witten, Friedman, and Simon (2011). Finally, as discussed also in Augugliaro *et al.* (2020a), the computational burden needed to compute the mixed moments in \hat{C}_{yy} can be significant when the estimated precision matrix is dense. For this reason, we propose an approximate EM algorithm defined by replacing the matrix \hat{C}_{yy} with the approximation suggested in Guo, Levina, Michailidis, and Zhu (2015).

Before closing this section, we discuss the convergence of the coordinate descent algorithm used to maximize the penalized Q -function (5). As studied by Yin and Li (2011), the algorithm always converges to a stationarity point. In the classical setting, where the sample size is

Algorithm 2 Pseudo-code of the multi-lasso algorithm.

- 1: Let $\hat{\Theta}$ be the current estimate of the precision matrix
 - 2: **repeat**
 - 3: **for** $k = 1 \dots p$ **do**
 - 4: compute: $\tilde{y}_{i,k} = \hat{y}_{i,k} + \hat{\theta}_{kk}^{-1} \sum_{h \neq k} \hat{\theta}_{hk} \{\hat{y}_{i,h} - \mathbf{x}_i^\top \hat{B}_h\}$
 - 5: compute: $\hat{B}_k = \arg \min_{B_k} \frac{1}{n} \|\tilde{Y}_k - X B_k\|^2 + \lambda \|\beta_k\|_1$
 - 6: replace the k th column of the matrix \hat{B} with \hat{B}_k
 - 7: **end for**
 - 8: **until** convergence criterion is met
-

large enough, the global optimality is always assured, while, in a high-dimensional setting, the convergence properties of the coordinate descent algorithm closely depend on the ratio between sample size and the number of nonzero estimates or, in other terms, on the values of the tuning parameters used. When λ and ρ are large enough with respect to the sample size, the algorithm converges to the unique stationary point, whereas when the two tuning parameters are too close to zero, there are potentially many stationary points due to the high-dimensional nature of the parameter space. This behavior is not surprising and shared by many algorithms developed for inference in the case of high-dimensionality. In that regard, a key role will be played by the arguments `lambda.min.ratio` and `rho.min.ratio` of the implemented `cglasso()` function (see Section 5.1), as they are designed to define the smallest λ and ρ value used throughout the fitting process.

3. Package structure and usage

The package `cglasso` is available under the general public license (GPL ≥ 2) from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=cglasso>, and can be installed and loaded into the current R session using the following code:

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

The proposed package provides an integrated set of core routines for manipulation, simulation, visualization and analysis of datasets drawn from a conditional Gaussian graphical model, possibly featuring censored and/or missing values. According to their role in the analysis, the implemented functions have been classified into four distinct groups. Figure 1 provides a graphical representation of the overall structure of the proposed package.

The first group of functions, called *data manipulation*, contains the main function used to retrieve the internal representation of a dataset drawn from a conditional Gaussian graphical model, with possibly censored and missing values, as well as a set of specific method and accessor functions. The next group, called *model fitting*, is devoted to the fitting step of the implemented models and to the analysis of the corresponding results. This part of the package also contains functions for prediction and imputation of the censored and missing data. The third group, called *model selection*, concerns the model fit evaluation. It contains the main goodness-of-fit functions and a set of functions for evaluating and comparing the fitted models using graphs and summary statistics. Finally, the last group, called *network analysis*, is

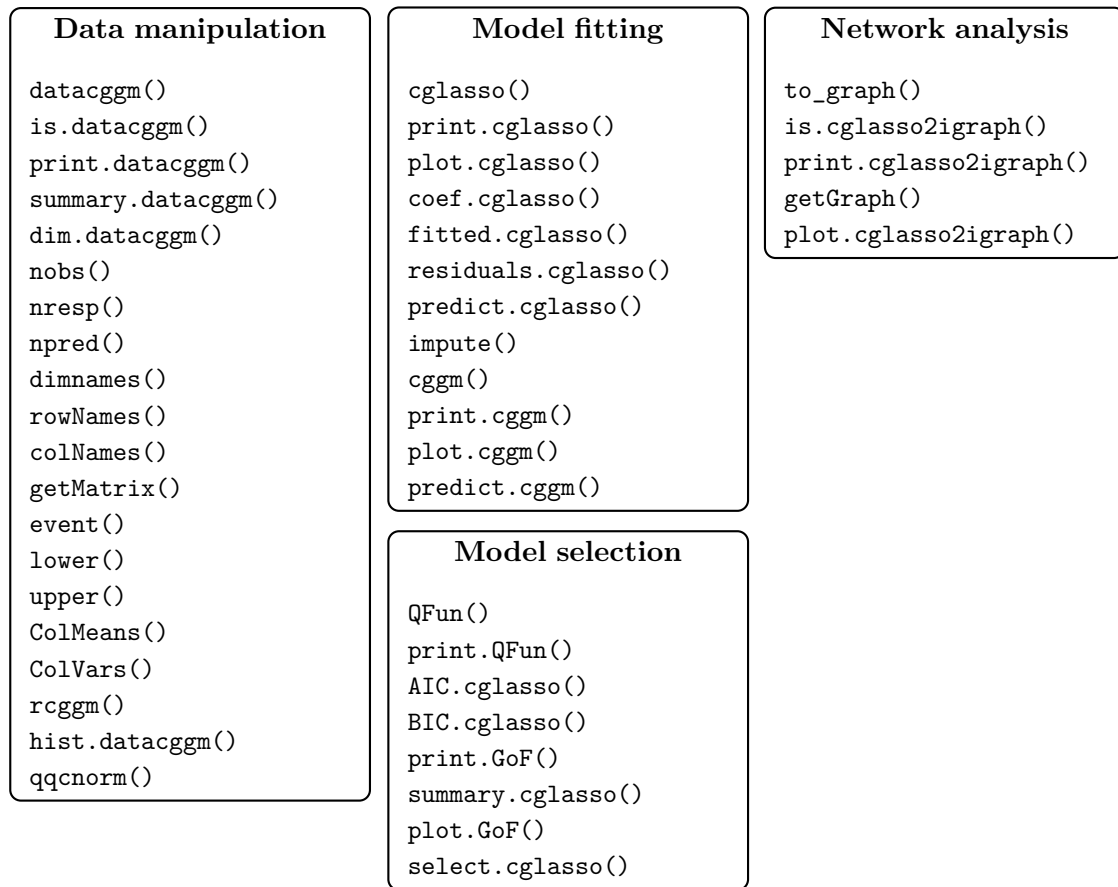


Figure 1: Overall structure of the **cglasso** package: The implemented functions are partitioned into four disjoint groups, graphically depicted as rectangles.

devoted to the statistical analysis of the fitted graph. To this end, the main function of this group builds upon existing functions from the R package **igraph** (Csardi and Nepusz 2006), augmenting it with a set of specific plotting method functions. We highlight that **cglasso** package provides also an utility function, named **ShowStructure()**, aimed to provide insight into the package structure. This function returns a graph depicting the proposed package's overall structure. Nodes and edges belonging to a specific group of functions are drawn with a specific color and the main functions are marked using bold text. We refer to the help-page for further details and examples.

Before going into the technical details of the implemented functions, we summarize the key features of the proposed package by performing a real data analysis on two datasets, which we distinguish in the next two sub-sections by the presence or not of covariates, respectively, followed by a sub-section where we present a function to simulate data from the model and two plotting functions to evaluate the theoretical assumptions of the implemented methods. The formal definition of the implemented functions, the technical description of their arguments as well as further examples are included in the next section.

3.1. Network inference from multiple myeloma data

Dataset description

We start the description of the main functions implemented in the proposed package using a subset of the data originally studied in [Gutiérrez *et al.* \(2010\)](#). The dataset is called `MM` and can be loaded inside the current R session by:

```
R> data("MM", package = "cglasso")
```

The aim of the study was to investigate the expression level of a set of 49 MicroRNAs (miRNAs), measured on a sample of 64 patients with multiple myeloma (MM). miRNAs are endogenous small non-coding RNAs, approximately 22 nucleotides in length, that play a regulatory role in gene expression by mediating mRNA cleavage or translation expression. In this study, patients were selected to represent the most relevant recurrent genetic abnormalities in MM.

`MM` is an object of class ‘`list`’ with entries named `Y` and `X`. Matrix `Y` contains the measured cycle-threshold for $p = 49$ miRNAs, whereas `X` is a ‘`data.frame`’ containing the expression level of 15 endogenous internal reference genes, called housekeeping and typically used in the normalization of RT-qPCR data, and a categorical variable encoding the genetic abnormalities considered in the study. RT-qPCR data are typically right-censored ([Augugliaro *et al.* 2020b](#)) and, in this study, the right censoring value is equal to 40 for each response variable. This value represents the limit of detection of the measuring device used.

Data manipulation

We start the analysis of the `MM` dataset using the function `datacggm()`, which represents the main function of the group named *data manipulation* as it has been designed to represent, in a comprehensive way, a dataset with censored and/or missing values drawn from a conditional Gaussian graphical model. We refer the reader to Section 4.1 for the formal definition and description of the arguments of this function.

Below we run `datacggm()` specifying the response matrix, that is the matrix containing the cycle-thresholds (argument named `Y`), the predictor matrix, that is the ‘`data.frame`’ containing the housekeepings’ expression levels and categorical variable encoding the genetic abnormalities (argument named `X`), and the right censoring value (argument named `up`):

```
R> MM.datacggm <- datacggm(Y = MM$Y, X = MM$X, up = 40)
```

As can be confirmed running the check function `is.datacggm()`, function `datacggm()` returns an R object with attribute `class` set to ‘`datacggm`’ so that a coherent set of accessor and method functions can be used to handle all the features of this R object easily (see Section 4.2 and 4.3, for a complete description of these functions). For example, we can extract the two matrices enclosed in `MM.datacggm` by the accessor function `getMatrix()`:

```
R> Y <- getMatrix(MM.datacggm, name = "Y")
R> X <- getMatrix(MM.datacggm, name = "X")
```

Similarly, we can recover the upper and lower censoring values by the accessor functions `upper()` and `lower()`.

Since an object of class ‘`datacggm`’ encloses information on two different matrices, we have extended the standard R method functions typically used to recover or set the attributes of a single matrix to this case. As a general rule, all the implemented extensions return a list whose components contain the required attribute of the matrix Y and/or X, respectively. As an example, below we show the output from the call to the function `dim()`:

```
R> dim(MM.datacggm)
```

```
$Y
```

```
[1] 64 49
```

```
$X
```

```
[1] 64 16
```

Similarly, one can use the function `dimnames()`, along with the `$` operator, to recover the `dimnames` attribute of the matrices Y and X, respectively:

```
R> dimnames(MM.datacggm)$Y />
+   lapply(head)
```

```
[[1]]
```

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

```
[[2]]
```

```
[1] "miR_193b" "miR_23a" "miR_362" "miR_629" "miR_328" "miR_550"
```

```
R> dimnames(MM.datacggm)$X />
+   lapply(head)
```

```
[[1]]
```

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

```
[[2]]
```

```
[1] "RNU44_1" "RNU44_2" "RNU44_3" "RNU44_4" "RNU44_5" "RNU44_6"
```

then, the functions `rowNames()` and `colNames()` can be used to rename rows and columns, respectively:

```
R> rowNames(MM.datacggm)$Y <- paste0("u", seq_len(nobs(MM.datacggm)))
R> rowNames(MM.datacggm)$X <- paste0("u", seq_len(nobs(MM.datacggm)))
R> colNames(MM.datacggm)$Y <- paste0("Y", seq_len(nresp(MM.datacggm)))
R> colNames(MM.datacggm)$X <- c(paste0("X", seq_len(15L)), "CyAb")
```

We point out that the information about the sample size, the number of response variables and the number of predictors can be recovered also by the accessor functions `nobs()`, `nresp()` and `npred()`, respectively.

In some applications, such as genomics, the matrices Y and X can be very large, both in terms of the number of rows and columns. Thus, we have implemented a specific printing method function to avoid the generation of excessive output on screen:

```
R> print(MM.datacggm, n = 5L)
```

```
Printing 'datacggm' object
```

```
Y: 64 x 49 matrix
```

```

      Y1    Y2    Y3    Y4    Y5    Y6    Y7    Y8    Y9    Y10   Y11
u1 39.3  40.0+ 40.0+ 40.0+ 40.0+ 40.0+ 40.0+ 39.3  40.0+ 40.0+ 40.0+
u2 36.2  33.0  40.0+ 36.7  40.0+ 40.0+ 34.6  40.0  36.9  36.0  31.8
u3 40.0+ 35.2  37.5  36.2  36.4  36.2  33.5  37.6  35.7  34.4  34.3
u4 40.0+ 40.0+ 40.0+ 40.0+ 35.8  36.3  36.7  37.3  39.1  35.4  35.6
u5 40.0+ 40.0+ 40.0+ 37.9  36.3  37.5  34.9  36.8  36.0  37.5  34.6
# with 59 more rows, and 38 more variables
```

```
X: 64 x 16 matrix
```

```

      X1      X2      X3      X4      X5      X6      X7
u1 -0.1279 -0.15497 -0.3445 -0.12995 -0.15511  0.14165 -0.19687
u2 -0.8331 -0.79277 -0.8505 -1.02452 -0.95223 -0.78481 -0.71611
u3 -0.7322 -0.84625 -0.7744 -1.02427 -0.86677 -1.16685 -0.97674
u4  0.6705  0.68098  0.9550  0.61373  0.63437  1.06343  0.62363
u5 -0.2508 -0.20502 -0.3107 -0.04907 -0.07696 -0.10914 -0.05496
# with 59 more rows, and 9 more variables
```

By default, only the first ten rows are printed but we use the optional argument `n` to show only the first five rows. This example also shows how the function `print()` structures the printed output into two sections which are referred to as the response and the predictor matrix, respectively. The readability of the matrix `Y` is improved by adding the symbol “+” at the end of each right censored value. In the same way, left censored values are marked with the symbol “-”.

The same rationale was used to organize the output of the function `summary()`. For example, running the `summary()` function with argument `n = 5L`, provides the summary statistics for the first 5 response and predictor variables:

```
R> summary(MM.datacggm, n = 5L)
```

```
Y:
```

```

      Lower Min. 1st Qu. Median Mean 3rd Qu. Max. Upper NA% LC% RC%
Y1 -Inf  32.8 35.9    36.4  36.6 37.5    39.8 40    0%  0%  35.94%
Y2 -Inf  32.9 35.2    36.2  36.0 36.9    38.8 40    0%  0%  29.69%
Y3 -Inf  32.9 35.2    36.5  36.3 37.6    39.1 40    0%  0%  31.25%
Y4 -Inf  33.5 35.2    36.0  36.1 37.1    39.3 40    0%  0%  31.25%
Y5 -Inf  30.6 34.5    35.5  35.4 36.3    39.4 40    0%  0%  25.00%
# with 44 more variables
```

```
X-numeric:
```

```

      Min.  1st Qu. Median  Mean    3rd Qu. Max.
```

```

X1 -2.48 -0.558 -0.11133 -5.09e-16 0.533 4.26
X2 -2.38 -0.634 -0.09684 -5.06e-16 0.497 3.94
X3 -2.35 -0.627 -0.10154 -2.93e-16 0.531 4.03
X4 -2.25 -0.577 0.00826 4.57e-16 0.639 3.78
X5 -2.45 -0.607 -0.04684 -5.19e-16 0.479 4.14
# with 10 more variables

```

X-categorical:

CyAb :

	Freq	Perc
CyAb:0	5	7.81%
CyAb:1	13	20.31%
CyAb:2	14	21.88%
CyAb:3	11	17.19%
CyAb:4	4	6.25%
CyAb:5	17	26.56%

As shown, the printed summary statistics are divided into three sections: The first section, named `Y`, reports the summary statistics for the response matrix, whereas the second and third sections, named `X-numeric` and `X-categorical`, are devoted to summarizing the numeric and categorical predictors, respectively. For each variable reported in the `Y` section, the function `summary()` reports the standard summary statistics computed using only the observed values. This section reports also the lower and upper censoring values, under the columns named `Lower` and `Upper`, respectively, and the percentage of missing-at-random, left- and right-censored values, under the columns named `NA%`, `LC%` and `RC%`, respectively.

Model fitting

The function `cglasso()` represents the main fitting function belonging to the group named *model fitting*. As we shall discuss in Section 5.1, this function has been designed to fit a broad range of `cglasso` estimators, and the appropriate version is automatically selected using the auxiliary information enclosed in the object of class `'datacggm'` that is passed to the function. Thus, in accordance with Table 4, as the object `MM.datacggm` encloses auxiliary information about a set of predictors, that is the matrix `X`, and the unobserved response values are encoded as right censored, the `cglasso()` function will fit a sequence of conditional Gaussian graphical models under censoring (Augugliaro *et al.* 2020b).

By default, these models are fitted using a decreasing sequence of ten λ values and a decreasing sequence of ten ρ values. In order to focus our discussion on the output given by the `cglasso()` function, and in order to avoid the generation of excessive output on screen, we will consider now only two λ values and two ρ values. In the next section, we will show how to conduct model fitting across a path of solutions, in order to select the optimal λ and ρ value.

The function `cglasso()` can be run by the following code:

```

R> mylambda <- c(0.30, 0.20)
R> myrho <- c(0.50, 0.40)
R> MM.mdl <- cglasso(data = MM.datacggm, lambda = mylambda, rho = myrho)

```

where the arguments `lambda` and `rho` are used to specify the two sequences of tuning parameters. We refer to Section 5.1 for the description of the full set of arguments. Function `cglasso()` returns a list with attribute `class` set to `'cglasso'`, thus, the user can use a specific printing method function to print on screen the main summary statistics of the fitted models:

```
R> MM.mdl
```

```
Call:  cglasso(data = MM.datacggm, lambda = mylambda, rho = myrho)
```

```
1 :
  lambda rho  df          N. Comp.
    0.3  0.5 465  (20.630%)      1
    0.3  0.4 482  (21.384%)      1

2 :
  lambda rho  df          N. Comp.
    0.2  0.5 490  (21.739%)      1
    0.2  0.4 500  (22.183%)      1
```

```
---
```

```
model: 'conditional censored glasso'
  nObs: 64
  nResp: 49
  nPred: 16
```

As shown above, the function `print()` organizes the output into two different parts. The first one is structured in sections, where each section refers to the sub-sequence of fitted models obtained while keeping one of the two tuning parameters fixed (λ in our example). Each section reports information about the complexity of the fitted models and the topological structure of the corresponding estimated graphs. In detail, for each pair of λ and ρ value used, the output reports the degrees-of-freedom (`df`), defined as the number of unique nonzero estimates, and the number of connected components (`N. Comp.`), that is, the number of maximal connected subgraphs in which the estimated graph can be decomposed. For example, let us consider the results obtained using $\lambda = 0.30$ and $\rho = 0.40$. The fitted model corresponding to these two values has 482 non-zero estimated parameters (about 21.384% of all model parameters) and the undirected graph representing the estimated precision matrix can not be decomposed into subgraphs since it has only one connected component. As expected, reducing the two tuning parameters will result in an increase in terms of degrees-of-freedom. Finally, the last part of the printed text reports the name of the fitted model, the sample size, the number of response variables and the number of predictors, respectively.

The group of functions devoted to the model fitting provides also a rich set of method functions for extracting the results of the fitting process. For illustration purpose only, in what follows we will explain these functions using only the default setting and we refer to the Section 5.3 for their description and further examples.

The estimated model coefficients can be extracted using the function `coef()`

```
R> coef(MM.mdl) |>
+   lapply(dim)
```

```
$B
[1] 21 49 2 2
```

```
$Sigma
[1] 49 49 2 2
```

```
$Theta
[1] 49 49 2 2
```

By default this method function returns a named list containing the estimated regression coefficient matrix (**B**), the estimated covariance matrix (**Sigma**) and, finally, the estimated precision matrix (**Theta**). These quantities are stored as ‘**array**’ where the last two dimensions are referred to the λ and ρ values used in the fitting process.

Similarly, we can extract from `MM.mdl` the fitted values, the residuals and the imputed response values via the functions `fitted()`, `residuals()` and `impute()`, respectively:

```
R> MM.mdl.fitted <- fitted(MM.mdl)
R> MM.mdl.obsRes <- residuals(MM.mdl)
R> MM.mdl.imputed <- impute(MM.mdl)
```

By default, `residuals()` returns the “observed” residuals, defined as the difference between the observed and the corresponding fitted response values. For missing and censored values, the observed residuals are set equal to `NA`. For a given row of the response matrix, missing values are imputed using the expected values of a multivariate normal distribution conditional on the observed values, whereas the censored values are imputed using expected values of a multivariate truncated normal distribution conditional on the observed values.

The output of the `cglasso()` function can be used also to predict the estimates of the model parameters corresponding to new λ and ρ values. For example, one can compute the predicted regression coefficient matrix (default setting) at $\lambda = 0.25$ and $\rho = 0.45$

```
R> B.pred <- predict(MM.mdl, lambda.new = 0.25, rho.new = 0.45)
```

As discussed in Section 5.4, this function can be used also to predict the estimate of the covariance matrix, the estimate of the precision matrix or the fitted values.

Finally, we conclude this section by pointing out that in some applications, the user may prefer to use the ℓ_1 -penalty function only as a tool for model selection, but then to estimate and evaluate the selected sparse model by (constrained) maximum likelihood. For this reason, we have also included a second model fitting function, called `cggm()`. This function is specifically designed to fit a conditional Gaussian graphical model via a maximum likelihood method, with nonzero parameters identified by the model structure of a given ‘`cglasso`’ object, and can be used as follows:

```
R> MM.mdl.mle <- cggm(MM.mdl)
```

We refer to Section 5.5 for the description of this function and further examples of how to link this function to the group of functions devoted to the model fitting evaluation. As show below:

```
R> class(MM.mdl.mle)

[1] "cggm"      "cglasso"
```

the function `cggm()` returns an object of class ‘`cggm`’ inheriting the class ‘`cglasso`’, hence the method and accessor functions previously described can be used also on this output.

Model selection

In this section we continue the study of the multiple myeloma dataset by fitting a sequence of conditional censored glasso models using a finer grid of tuning values, chosen after a preliminary study aimed at improving the quality of the resulting graphs:

```
R> mylambda <- seq(from = 0.30, to = 0.20, length = 10L)
R> myrho <- seq(from = 0.50, to = 0.40, length = 10L)
R> MM.mdl.path <- cglasso(data = MM.datacggm, lambda = mylambda, rho = myrho)
```

Upon completion of the fitting algorithm, optimal values of the two penalty parameters λ and ρ have to be chosen such that the fitted model is sparse enough to highlight the conditional dependence structure of the response variables, while not unnecessarily suppressing relevant data signal.

To help the user with the selection of the optimal fitted model, the `cglasso` package provides two specific goodness-of-fitting functions, called `AIC()` and `BIC()` respectively, that can be used to compute suitable extensions of the well-known Akaike information criterion (AIC, Akaike 1973) and Bayesian information criterion (BIC, Schwarz 1978). As we shall discuss in detail in Section 6, these two functions play an essential role inside the ecosystem of the functions belonging to the *model selection* group, as their output, formally an object of class ‘GoF’, can be passed to the other functions for summarizing the fitted models or selecting the optimal model. We implemented it this way, as, in some contexts, such as genomics, the computational time needed to evaluate the goodness-of-fit of a sequence of models can be very expansive. In this way, the results of the information criteria can be easily shared among functions, without having to be re-calculated.

Coming back to our study, we select the optimal fitted model using the extended BIC criterion (Chen and Chen 2008). This measure of goodness-of-fit extends the classical BIC measure by adding a hyper parameter γ and using the maximum likelihood estimates to evaluate the model fitting (see also Section 6.3 for further details). As the authors suggested, we let the hyper parameter $\gamma = 0.5$. Thus, we run the `BIC()` function by setting the arguments `g = 0.5` and `mle = TRUE`:

```
R> MM.mdl.path.eBIC <- BIC(MM.mdl.path, g = 0.5, mle = TRUE)
```

A graphical evaluation of the fitted models in terms of goodness-of-fit can be obtained using the plotting method function:

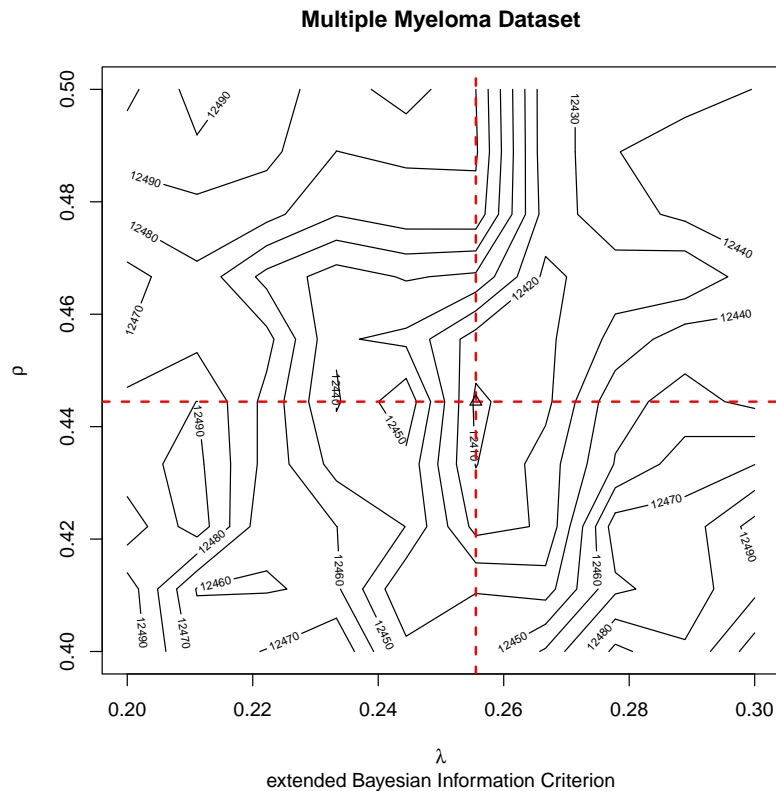


Figure 2: Contour plot used to identify the optimal values of the tuning parameters.

```
R> plot(MM.mdl.path.eBIC, main = "Multiple Myeloma Dataset")
```

Figure 2 shows the output of the call to the method function. As we will explain in Section 6, since we fitted a set of conditional Gaussian graphical models, the plotting method function returns a contour plot, where the used λ and ρ values are reported on the x and y axis, respectively, whereas the values of the measure of goodness-of-fit are reported as contours. In this figure, the optimal λ and ρ values are identified using vertical and horizontal red dashed lines, respectively.

Once the optimal values of the two tuning parameters have been identified, we can use the function `select.cglasso()` to recover the corresponding fitted model from `MM.mdl.path`:

```
R> MM.optmdl <- select.cglasso(MM.mdl.path, GoF = MM.mdl.path.eBIC)
```

As discussed above, the argument `GoF` allows to pass the evaluations of the chosen information criterion, which are enclosed in the `MM.mdl.path.eBIC` object, to this function.

Finally, as common in R, a complete summary of a fitted model is provided by the summary method function:

```
R> summary(MM.optmdl)
```

```
Call: cglasso(data = MM.datacgmm, lambda = mylambda, rho = myrho)
```

lambda	rho	df.B	df.Tht	df	(df%)	eBIC_CC
0.256	0.444	69	411	480	(21.295%)	12408

=====

Summary of the Selected Model

```

model: 'conditional censored glasso'
  nObs: 64
  nResp: 49
  nPred: 16
lambda: 0.2555556
lambda.id: 1
  rho: 0.4444444
rho.id: 1
eBIC_CC: 12407.99
  df.B: 69
  df.Tht: 411
  df: 480

```

=====

The output printed on screen is structured into two sections. The first one augments the results previously printed by the `print()` function by adding the columns reporting the number of estimated non-zero regression coefficients (`df.B`) and the number of estimated non-zero partial correlation coefficients (`df.Tht`). The last column is inherited from `MM.mdl.path.eBIC` and reports the minimum value of the chosen information criterion (the extended BIC in our example). The second section reports the primary summary statistics of the selected model.

Network analysis

The final group of functions, named *network analysis*, provides a set of routines with which the user can plot and analyze an estimated graph. These routines are related to the model fitting group by the function `to_graph()`, which is the primary function designed to return an R object that encloses all the information about the estimated graphs (see Section 7.1 for further technical details):

```
R> MM.optmdl.graph <- to_graph(MM.optmdl)
```

This function returns a named list with class `'cglasso2igraph'`, containing two entries:

```
R> names(MM.optmdl.graph)
```

```
[1] "Gyy" "Gxy"
```

which are objects of class `'igraph'`. The entry named `Gyy` encloses the information about the undirected graph that describes the conditional dependence structure among the p response variables. In contrast, `Gxy` is a directed graph used to describe the effects of the predictors on the expected values of the response variables. These two entries can be extracted from `MM.optmdl.graph` using the accessor function `getGraph()`

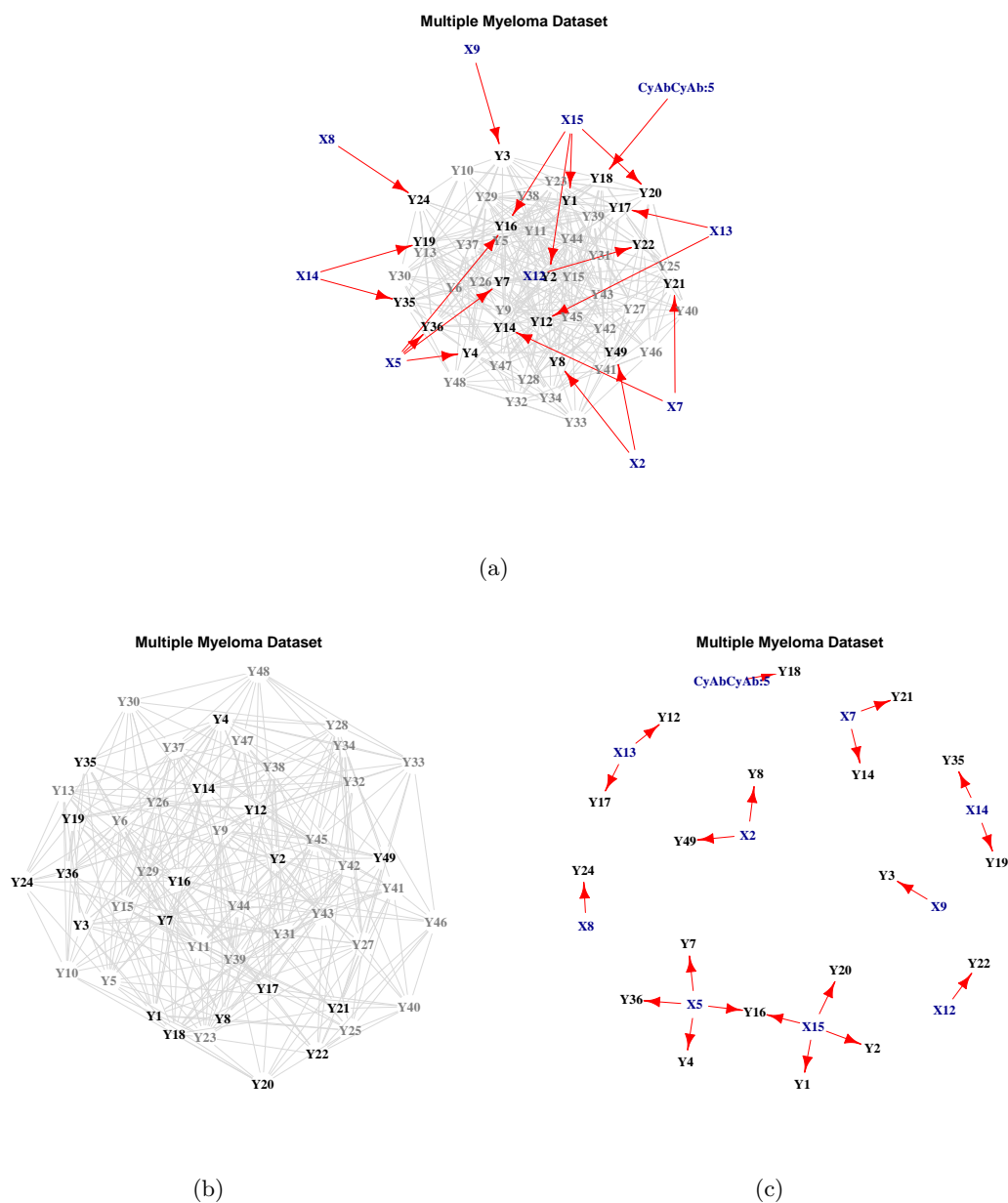


Figure 3: Output of the plotting method function associated to the object of class ‘`cglasso2igraph`’ and representing the network inferred using multiple myeloma dataset. Figures (a), (b) and (c) are obtained by changing the argument `type` in the `plot()` function.

```
R> Gyy <- getGraph(MM.optmdl.graph)
R> Gxy <- getGraph(MM.optmdl.graph, type = "Gxy")
```

By default, the `getGraph()` function returns only the undirected graph associated with the estimated precision matrix, but the optional argument `type` can be used to extract `Gxy`.

Given an object of class ‘`cglasso2igraph`’, we can plot the estimated graphs enclosed in `MM.optmdl.graph` by the plotting method function described in Section 7.2:

```
R> plot(MM.optmdl.graph, xlim = c(-0.85, 0.85), ylim = c(-0.85, 0.85),
+      main = "Multiple Myeloma Dataset", layout = layout_with_kk)
```

This results in Figure 3(a), where links corresponding to the estimated non-zero partial correlation coefficients are drawn with a gray line whereas the red arrows represent the estimated non-zero regression coefficients. If needed, the user can further split this graph into two sub-graphs by the optional arguments `type`. For example, Figure 3(b) shows the graph produced by the following code

```
R> plot(MM.optmdl.graph, xlim = c(-0.85, 0.85), ylim = c(-0.85, 0.85),
+      main = "Multiple Myeloma Dataset", type = "Gyy",
+      layout = layout_with_kk)
```

which describes only the conditional dependence structure among the response variables (`type = "Gyy"`), whereas the Figure 3(c) is produced by the code below

```
R> plot(MM.optmdl.graph, xlim = c(-0.85, 0.85), ylim = c(-0.85, 0.85),
+      main = "Multiple Myeloma Dataset", type = "Gxy",
+      layout = layout_with_kk)
```

and describes only the effects of the predictors on the response variables (`type = "Gxy"`).

3.2. Network inference from megakaryocyte-erythroid progenitors data

In this section we study a second dataset, called MKMEP, containing a subset of the data studied in Psaila *et al.* (2016). As discussed at the end of Section 3, we aimed to show how the functions previously presented can adapt their behavior according to the auxiliary information enclosed in a ‘`datacggm`’ object, which in this case is in the form of a dataset without predictors.

As before, we first load the dataset inside the R session:

```
R> data("MKMEP", package = "cglasso")
```

The study concerned the process of formation of blood cells, whereby the authors identified three distinct sub-populations of cells, which are all derived from hematopoietic stem cells through cell differentiation. One of these sub-populations, denoted by MK-MEP, is a previously unknown, rare population of bipotent cells that primarily generate megakaryocytic progeny. In this study, RT-qPCR technology was used to profile a set of genes and a set of single human MK-MEP cells.

Since the genes were profiled using RT-qPCR technology with an upper limit of detection equal to 40, we create an object of class ‘`datacggm`’ using the following code:

```
R> MKMEP.datacggm <- datacggm(Y = MKMEP, up = 40)
```

Then, `rowNames()` and `colNames()` functions are used to rename the rows and columns of `Y`:

```
R> rowNames(MKMEP.datacggm)$Y <- paste0("u", seq_len(nobs(MKMEP.datacggm)))
R> colNames(MKMEP.datacggm)$Y <- paste0("Y", seq_len(nresp(MKMEP.datacggm)))
```

When an ‘`datacggm`’ object does not enclose auxiliary information about the predictors matrix, as in this case, all printing method functions, as well as the functions aimed to set or retrieve the attributes of a ‘`datacggm`’ object, will set the corresponding `X` entry to `NULL`. As an example, a call to the printing method function will give the following output:

```
R> print(MKMEP.datacggm, n = 5L)
```

```
Printing 'datacggm' object
```

```
Y: 48 x 63 matrix
```

```

      Y1    Y2    Y3    Y4    Y5    Y6    Y7    Y8    Y9    Y10
u1 12.54 16.60 19.10 40.00+ 40.00+ 17.04 15.21 40.00+ 14.58 13.07
u2 15.35 40.00+ 19.97 40.00+ 16.86 16.45 15.03 15.15 14.35 12.95
u3 12.00 40.00+ 18.33 16.35 40.00+ 16.68 40.00+ 15.56 14.28 12.37
u4 12.42 15.80 40.00+ 40.00+ 40.00+ 15.28 13.96 40.00+ 14.48 40.00+
u5 11.74 40.00+ 40.00+ 14.69 40.00+ 14.98 40.00+ 14.01 15.37 12.93
# with 43 more rows, and 53 more variables
```

```
X: NULL
```

showing that `MKMEP.datacggm` stores the expression profiles of a set of 63 genes measured on 48 human MK-MEP cells.

Now, we perform a similar analysis to the previous one. In accordance with the categorization of the available implementations in Table 4, the function `cglasso()` will now fit a sequence of censored glasso models (Augugliaro *et al.* 2020a) across a decreasing sequence of ρ values. In the code below, the 100 values are chosen after a preliminary study so as to improve the quality of the resulting graphs:

```
R> myrho <- seq(from = 6, to = 3, length = 100L)
R> MKMEP.mdl.path <- cglasso(data = MKMEP.datacggm, rho = myrho)
```

We select the optimal fitted model using the extended BIC measure proposed in Foygel and Drton (2010) for Gaussian graphical models (see Section 6.3 for more details). Again, we let the hyper parameter γ equal to 0.5 ($g = 0.5$) and evaluate the model fitting using the maximum likelihood estimates (`mle = TRUE`):

```
R> MKMEP.mdl.path.eBIC <- BIC(MKMEP.mdl.path, g = 0.5, mle = TRUE)
R> plot(MKMEP.mdl.path.eBIC,
+       main = "Megakaryocyte-Erythroid Progenitors Dataset")
```

Figure 4(a) shows the result of the plotting method function. Differently to the previous analysis, in this case, the `plot()` function returns a plot showing the values of the selected measure of goodness-of-fit as a function of the unique tuning parameter used to fit the sequence of models. Again, the optimal value of the tuning parameter is highlighted with a vertical red dashed line.

Now, given the optimal ρ value, we can compute the maximum likelihood estimates of the parameters of the selected model by `cggm()`. Different the previous analysis, we use the

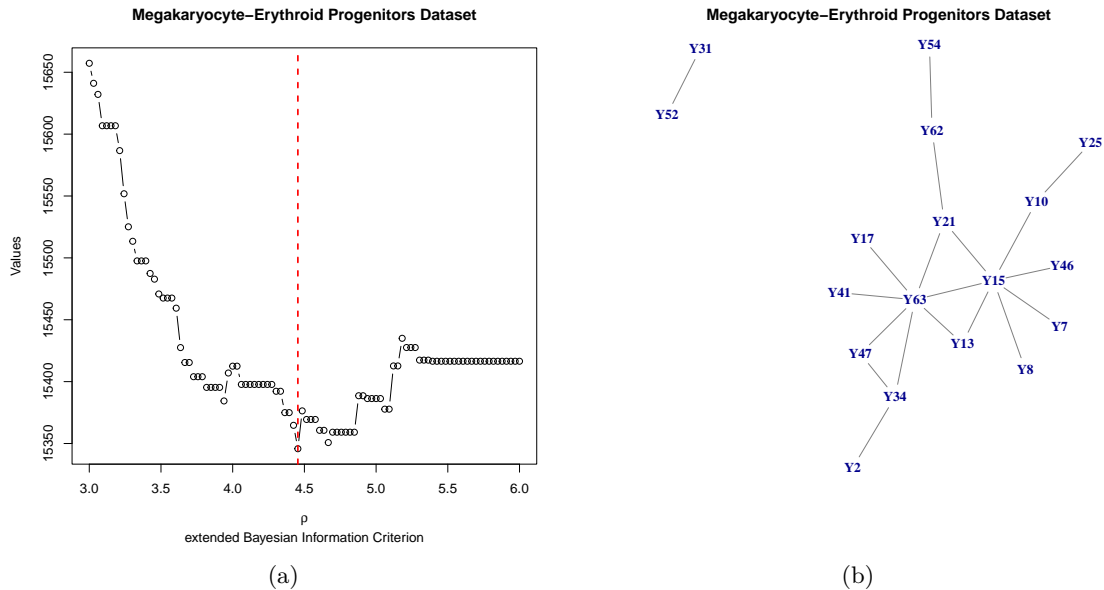


Figure 4: Left: Plot of the extended BIC criterion versus the values of the tuning parameter used to fit the model. Right: Output of the plotting method function associated to the object of class ‘*cglasso2igraph*’ and representing the network inferred from the megakaryocyte-erythroid progenitors dataset. Isolated vertices have been removed in order to improve graph visualization.

argument `GoF` to attach the results enclosed in `MKMEP.mdl.path.eBIC` directly to the function `cggm()`. In this way, we can carry out selection and model fitting in a single step (see Section 5.5 and Section 6 for more details):

```
R> MKMEP.optmdl.mle <- cggm(MKMEP.mdl.path, GoF = MKMEP.mdl.path.eBIC)
R> MKMEP.optmdl.mle
```

```
Call: cggm(object = MKMEP.mdl.path, GoF = MKMEP.mdl.path.eBIC)
```

```
df          N. Comp.
145 (6.975%)      47
```

The output reveals how the selected fitted model corresponds to an undirected conditional independence graph among the variables considered, that can be decomposed into 47 connected components. Furthermore, the estimated model contains only 145 non zero estimated parameters.

Finally, we extract and plot the undirected graph associated to `MKMEP.optmdl.mle` via the following code:

```
R> MKMEP.optmdl.graph <- to_graph(MKMEP.optmdl.mle)
R> plot(MKMEP.optmdl.graph, xlim = c(-0.85, 0.85), ylim = c(-0.85, 0.85),
+       main = "Megakaryocyte-Erythroid Progenitors Dataset",
+       layout = layout_with_kk)
```

Figure 4(b) shows the undirected graph representing the network inferred from the megakaryocyte-erythroid progenitors dataset, where the isolated vertices have been removed in order to aid with the visualization.

3.3. Simulating ‘datacggm’ objects and plotting functions

We conclude the first part of the paper, by presenting a simulating function and two plotting method functions specifically designed to allow the user to evaluate the theoretical properties and assumptions of the implemented methods.

A dataset with censored and/or missing-at-random values can be drawn from a multivariate Gaussian distribution using the function `rcggm()`. We refer to Section 4.4 for the formal definition and the complete description of the arguments, whereas, in this section we elucidate how to use this function by means of an example. Firstly, we simulate one thousand observations from a multivariate Gaussian distribution with:

$$E(Y_{ij}) = 10 \quad \text{and} \quad \text{COV}(Y_{ih}, Y_{ik}) = 0.3^{|h-k|},$$

for $j = 1, \dots, 4$:

```
R> set.seed(1234)
R> n <- 1000L
R> p <- 4L
R> b0 <- rep(10, time = p)
R> Sigma <- outer(1:p, 1:p, function(i, j) 0.3^abs(i - j))
```

For each response variable we let the probability of left-censoring, right-censoring and missing-at-random equal to 0.1:

```
R> probl <- 0.1
R> probr <- 0.1
R> probna <- 0.1
```

Finally, the desired dataset can be drawn from the multivariate Gaussian distribution passing the previous objects to the `rcggm()` function:

```
R> Z.sim <- rcggm(n = n, b0 = b0, Sigma = Sigma, probl = probl,
+   probr = probr, probna = probna)
```

Since `rcggm()` returns a ‘datacggm’ object, we can summarize it by the summary method function:

```
R> summary(Z.sim)
```

```
Y:
  Lower Min. 1st Qu. Median Mean 3rd Qu. Max. Upper NA% LC% RC%
Y1 8.72  8.73 9.47    9.97  9.98 10.5    11.3 11.3  9.7%  7.5% 10.7%
Y2 8.72  8.72 9.51   10.04 10.00 10.5    11.3 11.3 10.0%  9.9% 10.2%
Y3 8.72  8.72 9.52   10.06 10.04 10.6    11.3 11.3  8.7% 10.6%  9.9%
Y4 8.72  8.73 9.43    9.99  9.99 10.5    11.3 11.3 10.2% 10.4% 11.3%
```

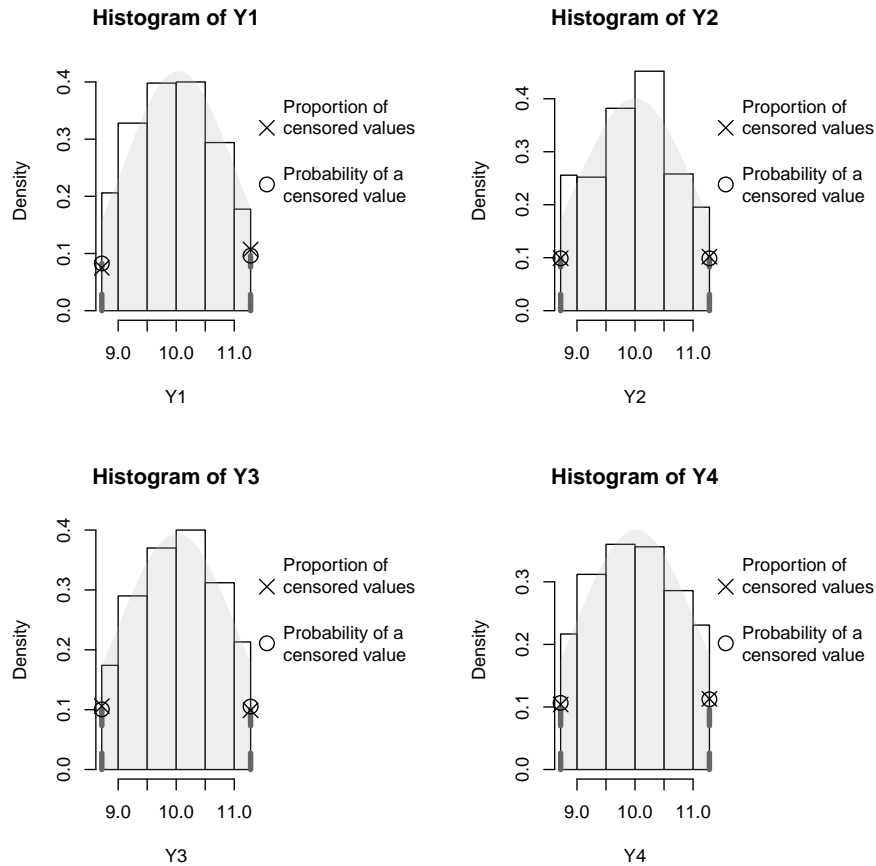


Figure 5: Output of the function `hist()` on data with censored values. The plot provides a histogram for each response variable, together with the area of the Gaussian density function as background and a comparison between the proportions of left/right censored values (crosses) and the corresponding Gaussian tail probabilities (circles).

X-numeric: NULL

X-categorical: NULL

The previous output shows that the reported proportions are close to their theoretical values. As shown later in Section 4.4, the lower and upper censoring values are implicitly computed using the arguments `probl` and `probr`, respectively.

The proposed package contains also two specific plotting functions, named `hist()` and `qqcnorm()`, which can be used to graphically evaluate the probabilistic assumptions on the distribution of the response variables. Both functions are characterized by a rich set of arguments allowing the user a high degree of personalization of the resulting graphs. We refer to the Section 4.5 for a complete description.

Function `hist()` produces a set of histograms:

```
R> hist(Z.sim, max.hist = 4L, breaks = 5L, save.hist = TRUE)
```

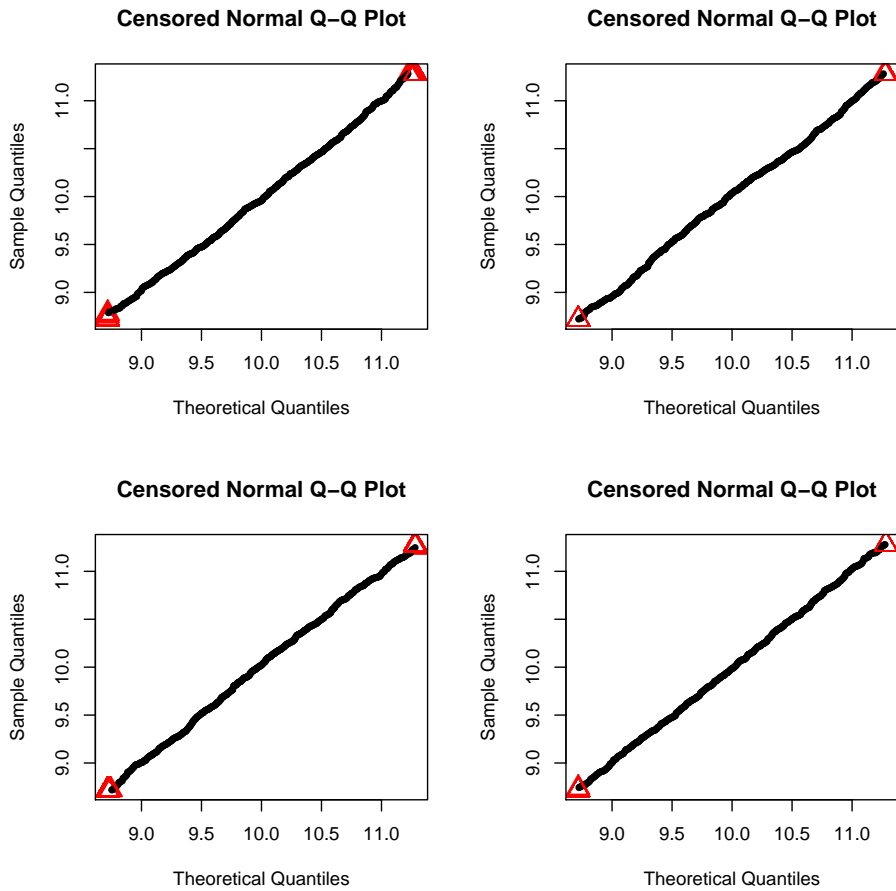



Figure 6: Output of the function `qqcnorm()` on data with censored values. Normal Q-Q plots for comparing the empirical quantiles to the truncated Gaussian quantiles.

Figure 5 assists in the evaluation of the probabilistic assumptions on the response variables, by highlighting the area of the Gaussian density function as background of each histogram. Furthermore, the proportions of left/right censored values are graphically compared with the corresponding Gaussian tail probabilities. If the assumption about the censoring mechanism is satisfied, then the proportions and tails probabilities should be approximately equal to each other, as clearly shown in Figure 5. The arguments of the function allow to setup the maximum number of histograms plotted on a single graphic device (`max.hist = 4L`), the number of breakpoints between histogram cells (`breaks = 5L`) and to save the resulting histograms on an external file.

The second plotting method function that can be used to graphically evaluate the probabilistic assumption on the response variables is `qqcnorm()`. This function produces censored normal Q-Q (quantile-quantile) plots, that is, a graphical method for comparing the empirical distribution to the censored Gaussian distribution. As usual of Q-Q plots, the graphical evaluation is done by plotting the empirical quantiles (y coordinate) against the theoretical

quantiles (x coordinate) of the censored Gaussian distribution, which are defined as follows:

$$x = \begin{cases} \text{lo} & \text{if } p \leq \Phi\{(\text{lo} - \mu)/\sigma\} \\ \mu + \sigma\Phi^{-1}(p) & \text{if } \Phi\{(\text{lo} - \mu)/\sigma\} < p < 1 - \Phi\{(\text{up} - \mu)/\sigma\} \\ \text{up} & \text{if } p \geq 1 - \Phi\{(\text{up} - \mu)/\sigma\} \end{cases}$$

where $p \in (0, 1)$. If the two distributions are similar, the points will approximately lie on the line $y = x$. Figure 6 is obtained by the following R code

```
R> qqcnorm(Z.sim, max.plot = 4L, save.plot = TRUE)
```

As with the `hist()` function, points corresponding to the censored values are plotted using a specific symbol and color (in this example a red triangle) in order to evaluate if the proportions of left/right-censored values are coherent with the Gaussian tail probabilities.

4. Data manipulation: Technical details and further examples

We start the second part of the paper, by describing in detail the group of functions named *data manipulation*. Table 1 reports the full set of functions and their description.

As we shall see in the following sections, all functions belonging to this part of the package are specifically designed with the aim of working with an R object that represents, in a comprehensive way, a dataset with censored and/or missing values drawn from a conditional Gaussian graphical model. The main function of this group is `datacggm()`, which is the function devoted to return an R object with attribute `class` set to ‘`datacggm`’. As explained in Chambers (1992), attribute `class` allows to use the S3 object-oriented system, which makes it possible to define a coherent set of method functions within a specified environment.

4.1. The objects of class ‘`datacggm`’

The function `datacggm()` collects all information about the observed values in the data, the pattern of censored/missing values, and all other elements that may be useful during the first steps of the statistical analysis. It is defined as follows:

```
datacggm(Y, lo = -Inf, up = +Inf, X = NULL,
         control = list(maxit = 1.0E+4, thr = 1.0E-4))
```

The argument `Y` is a $(n \times p)$ -dimensional response matrix whose rows are independent observations; left and right censoring values are specified by `lo` and `up`, respectively, whereas missing-at-random response values are encoded, as usual, by `NA`. By default `lo` and `up` are set equal to `-Inf` and `+Inf`, respectively, meaning that censored values are not recorded. The argument `X` is optional and, if available, can be used to specify a predictor matrix, that is, a $(n \times q)$ -dimensional matrix by which to model the expected values of the p response variables. To improve its computational efficiency, the function `datacggm()` works as a wrapper function of two specific Fortran subroutines. The first one uses `Y`, `lo` and `up` to identify the patterns of censored/missing response values and returns additional elements that will be used during the fitting process of the implemented models. The second one estimates the marginal mean and variance of the response variables using an EM algorithm, for which the argument `control`

Function	Description
<code>datacggm()</code>	Create a dataset from a conditional Gaussian graphical model with censored and/or missing values
<code>is.datacggm()</code>	Is an object of class 'datacggm'?
<code>print.datacggm()</code>	Print method for a 'datacggm' object
<code>summary.datacggm()</code>	Summarizing objects of class 'datacggm'
<code>dim.datacggm()</code>	Dimensions of a 'datacggm' object
<code>nobs()/nresp()/npred()</code>	Extract the number of observations/responses/predictors from a 'datacggm' object
<code>dimnames.datacggm()</code>	Dimnames of a 'datacggm' object
<code>rowNames()/colNames()</code>	Row and column names of a 'datacggm' object
<code>getMatrix()</code>	Retrieve matrices Y and X from a 'datacggm' object
<code>event()</code>	Status indicator matrix from a 'datacggm' object
<code>lower()/upper()</code>	Lower and upper limits from a 'datacggm' object
<code>ColMeans()/ColVars()</code>	Form column means and variances of a 'datacggm' object
<code>rcggm()</code>	Simulate from a conditional Gaussian graphical model with censored and/or missing values
<code>hist.datacggm()</code>	Histogram for a 'datacggm' object
<code>qqcnorm()</code>	Quantile-quantile plots for a 'datacggm' object

Table 1: Functions belonging to the group named *data manipulation*.

is a list used to set the maximum number of steps (`maxit`) and the convergence threshold (`thr`).

To gain an insight into the internal representation of a dataset with censored and/or missing values drawn from a conditional Gaussian graphical model, we use the two objects created in Section 3.1 and Section 3.2, respectively, i.e., `MM.datacggm` and `MKMEP.datacggm`.

As discussed above, `MM.datacggm` is an object of class 'datacggm', that is a named list with the following components:

```
R> names(MM.datacggm)
```

```
[1] "Y"    "X"    "Info"
```

The first two components enclosed in a 'datacggm' object are the $(n \times p)$ -dimensional response matrix and the $(n \times q)$ -dimensional predictor matrix, respectively, whereas `Info` is a named list collecting additional information used during the fitting processes of the implemented models:

```
R> names(MM.datacggm$Info)
```

```
[1] "lo"    "up"    "R"     "order" "Pattern" "ym"
[7] "yv"    "n"     "p"     "q"
```

For example, the component named `order` is an integer vector of length n used to permute the rows of the matrix Y in such a way that, after permutation, the data are structured as

a sequence of patterns of censored/missing values. If **X** is available, then its rows are also permuted according to that **order**. The component **Pattern** provides information about the identified patterns, that is, what response variables are observed, censored or missing.

As a second toy example we use the object named `MKMEP.datacggm`. Since in this example the predictor matrix is not available, `datacggm()` returns a list with component named **X** equal to `NULL` and number of predictors set to zero:

```
R> MKMEP.datacggm$X; MKMEP.datacggm$Info$q
```

```
NULL
[1] 0
```

Since in some applications the matrices **Y** and **X** can be very large, we have implemented a specific printing method function defined as follows:

```
print(x, digits = 3L, n = 10L, width = getOption("width"), ...)
```

where **digits** is used to specify how many significant digits are to be used, **n** is the number of rows to be shown and **width** is the width of the text printed on screen. By default, only the first ten rows are printed, while the number of printed columns is computed in such a way that the resulting matrix fits within the available screen. If needed, the entire matrices can be printed on screen by letting **n** = `Inf` and **width** = `Inf`. If **X** is not available, then the last section is returned as `NULL`, as can be seen by running the same code on the `MKMEP.datacggm` object.

The same rationale was used for the `summary()` function, which is formally defined as follows:

```
summary(object, n, quantile.type = 7L, digits = 3L, quote = FALSE, ...)
```

where **object** is an object of class `'datacggm'`, **n** is the number of rows to be printed, **quantile.type** is an integer between 1 and 9 used to select one of the nine available quantile algorithms, **digits** is an integer used for formatting the text and, finally, **quote** is a logical object, indicating whether or not strings should be printed with surrounding quotes. Again, if the matrix **X** is not included in a `'datacggm'` object, then last two sections are returned as `NULL`.

4.2. Accessor functions

This group of functions contains also a set of useful functions for extracting specific elements from a `'datacggm'` object.

Firstly, the user can recover the matrices **Y** and **X** via the accessor function `getMatrix()`:

```
getMatrix(x, name = c("Y", "X", "both"), ordered = FALSE)
```

where **x** is a `'datacggm'` object, **name** specifies the required matrix and **ordered** is a logical value used to identify if the required matrix should be retrieved with rows ordered according to the patterns of censored values. Letting **name** = `"both"`, `getMatrix()` retrieves a named list with both matrices. For example, we can extract from `MM.datacggm` the ordered response matrix running the code:

```
R> Y <- getMatrix(MM.datacggm, name = "Y", ordered = TRUE)
```

The status indicator matrix associated to Y is defined as the matrix encoding the status of the response variables. This matrix can be extracted from an object of class ‘datacggm’ by the accessor function

```
event(x, ordered = FALSE)
```

where x and ordered have the same meaning as in getMatrix(). The entries of status indicator matrix are equal to 0, -1, +1 or +9 depending on whether a response variable is observed, left-censored (-1), right-censored (+1) or missing-at-random (+9):

```
R> R <- event(MM.datacggm, ordered = TRUE)
R> cbind(Y = Y[, 1L], R = R[, 1L]) |>
+   tail()
```

```
      Y R
u39 34.59527 0
u60 40.00000 1
u48 36.35477 0
u62 37.05790 0
u63 40.00000 1
u64 40.00000 1
```

The accessor functions lower() and upper() can be used to retrieve the lower and upper censoring values of each response variable and are formally defined in the following way:

```
lower(x)
upper(x)
```

where x is an object of class ‘datacggm’.

To extract the column means and column variances of the matrices enclosed in a ‘datacggm’ object, the proposed package provides the following function:

```
ColMeans(x)
ColVars(x)
```

Both functions return a named list with three components, the first one referring to the response matrix and the last two referring to the continuous and categorical predictors. As a possible example, below we use ColMeans() function to extract from MM.datacggm the marginal means:

```
R> ColMeans(MM.datacggm) |>
+   lapply(head, n = 5L)
```

```
$Y
      Y1      Y2      Y3      Y4      Y5
38.52112 37.68152 38.00302 37.85182 36.94704
```

```

$X.numeric
      X1          X2          X3          X4          X5
-5.090329e-16 -5.054889e-16 -2.928701e-16  4.571403e-16 -5.191160e-16

$X.categorical
  CyAb
"CyAb:5"

```

Marginal means and variances of the p response variables are estimated using an EM algorithm whereas marginal means and variances of the (fully observed) continuous predictor variables are computed in the usual way. For categorical predictors, the function `ColMeans()` returns the statistical mode whereas the function `ColVars()` returns the Gini-Simpson index.

Before closing this section, we point out that the information about the sample size, the number of response variables and the number of predictors can be recovered by the accessor functions `nobs()`, `nresp()` and `npred()`, which are formally defined as follows:

```

nobs(object, ...)
nresp(object, ...)
npred(object, ...)

```

where `object` is a ‘`datacggm`’ object and `...` is used to get compatibility with the standard generic functions implemented in R.

4.3. Retrieving and setting attributes

The group of functions devoted to the data manipulation provides also a coherent set of functions to retrieve or set attributes in a specific manner. These functions are formally defined in the following way:

```

dim(x)
dimnames(x)
rownames(x)
colnames(x)

```

where the argument `x` is an R object of class ‘`datacggm`’. Since an object of class ‘`datacggm`’ has been designed to enclose information on two matrices, the functions above extends their standard R version by returning a list with components containing the required attribute of the matrix `Y` and `X`, respectively. If the matrix of predictors is not included in the ‘`datacggm`’ object, then last component is set to `NULL`. This is done mainly for internal purposes only, so that the behavior of the implemented functions can be adapted to the specific features of the dataset studied.

Since these functions return always a list with component named `Y` and `X`, the user can retrieve or set the attribute of a specific matrix by combining their usage with the standard `$` operator, as typically done in R.

4.4. Simulation function

The simulation function `rcggm()` allows the user to evaluate the theoretical properties of

		Arguments				Distribution
n	p	b0	X	B	Sigma	
✓	✓					$Y_i \sim N_p(0; I_p)$
✓					✓	$Y_i \sim N_p(0; \Sigma)$
✓		✓				$Y_i \sim N_p(b_0; I_p)$
✓		✓			✓	$Y_i \sim N_p(b_0; \Sigma)$
			✓	✓		$Y_i \sim N_p(x_i^\top B; I_p)$
			✓	✓	✓	$Y_i \sim N_p(x_i^\top B; \Sigma)$
		✓	✓	✓		$Y_i \sim N_p(b_0 + x_i^\top B; I_p)$
		✓	✓	✓	✓	$Y_i \sim N_p(b_0 + x_i^\top B; \Sigma)$

Table 2: Relationship between a multivariate Gaussian distribution and arguments of the simulation function `rcggm()`. The minimum required arguments needed to specify a certain multivariate Gaussian distribution are marked by a checkmark.

the methodologies that have been developed and implemented in the **cglasso** package. This function is defined as follows:

```
rcggm(n, p, b0, X, B, Sigma, probl, probr, probna, ...)
```

The arguments of this function can be partitioned into three groups. Firstly, we have six arguments for specifying the multivariate Gaussian distribution from which the sample should be drawn. As shown in Table 2, only a subset of these arguments may be really needed to specify certain distributions. For example, to draw a sample from a multivariate standard normal distribution, the user has to specify only the sample size, that is the argument `n`, and the number of variables Y , that is the argument `p`. If the means are different from zero and/or the covariance matrix is different to the identity matrix, the arguments `b0` and `Sigma` are used to specify these values. In this case, the argument `p` can be omitted as it can be easily recovered from the dimensions of the arguments `b0` and `Sigma`. The arguments `X` and `B` are used to specify the linear model for the expected values of the `p` response variables. Here, `X` is the $(n \times q)$ -dimensional predictor matrix whereas `B` is the $(q \times p)$ -dimensional regression coefficients matrix. Thus, the argument `b0` plays the role of the vector of p intercepts, which differently to the rest of the paper, we keep separately from the matrix B so as to distinguish the different distributions, which are reported in the last column of Table 2. The next set of arguments for this function allows to specify the probability of observing a left-censored response value (`probl`), a right-censored response value (`probr`) or a missing-at-random response value (`probna`). Each of these arguments accepts either a vector of `p` values belonging to the closed interval $[0, 1]$ or a single value, which will be repeated `p` times to get a vector with the right length. If one of these three arguments is not specified, it is set equal to zero by default. Algorithm 3 reports the pseudo-code of the implemented algorithm and shows that lower and upper censoring values are implicitly computed using the arguments `probl` and `probr` (see Algorithm 3, Step 4). If $Y[i, j]$ is not censored, i.e., $Y[i, j] \in (1o[j], up[j])$, then it is replaced with NA using a Bernoulli random variable with probability $probna / (1 - probl[j] - probr[j])$. Finally, the argument `...` is used to pass further arguments to the function `mvrnorm()` available in the R package **MASS** (Venables and Ripley 2002, see Algorithm 3, Step 2).

Algorithm 3 Pseudo-code of the algorithm implemented with function `rcggm()`.

- 1: compute $\mu = \mathbf{b0} + \mathbf{X} \% * \% \mathbf{B}$
- 2: simulate matrix \mathbf{Y} with $Y[i, j] \sim N(\mu[i, j], \Sigma)$
- 3: **for** $j = 1, \dots, p$ **do**
- 4: compute $lo[j]$ and $up[j]$ as solutions of the following equations

$$n^{-1} \sum_{i=1}^n E\{I(W_{ij} \leq lo[j])\} = \text{probl}[j],$$

$$n^{-1} \sum_{i=1}^n E\{I(W_{ij} \geq up[j])\} = \text{probr}[j],$$

- where $I(\cdot)$ is the indicator function and $W_{ij} \sim N(\mu[i, j], \Sigma[j, j])$
- 5: **if** $Y[i, j] \leq lo[j]$ **then** replace it with $lo[j]$
 - 6: **if** $Y[i, j] \geq up[j]$ **then** replace it with $up[j]$
 - 7: **if** $Y[i, j] \in (lo[j], up[j])$ **then** replace it with NA using a Bernoulli random variable with probability $\text{probr}[j]/(1 - \text{probl}[j] - \text{probr}[j])$
 - 8: **end for**
-

4.5. Plotting method functions

The proposed package contains two specific plotting functions to plot an object of class ‘`datacggm`’, namely `hist()` and `qqcnorm()`. The first one is defined as follows

```
hist(x, breaks = "Sturges", include.lowest = TRUE, right = TRUE,
     nclass = NULL, which, max.hist = 1L, save.hist = FALSE, grdev = pdf,
     grdev.arg, polygon.col = adjustcolor("grey", alpha.f = 0.25),
     polygon.border = NA, segments.lwd = 4L, segments.lty = 2L,
     segments.col = "gray40", points.pch = c(4L, 1L), points.cex = 1.8,
     points.col = rep("black", 2L), legend = TRUE, ...)
```

and produces a set of histograms. As highlighted by the number of arguments, this function is quite flexible allowing a high degree of personalization of the resulting graphs. For the sake of brevity, below we describe only the main arguments and we refer the user to the help page for more details. The first argument, named `x`, is an object of class ‘`datacggm`’ whereas the next four arguments, that is `breaks`, `include.lowest`, `right` and `nclass`, are passed to `hist.default()` and used to compute the breakpoints between histogram cells. The argument `which` is an integer vector for specifying the response variables for which the histogram is required, whereas `max.hist` is an integer specifying the maximum number of histograms plotted on a single graphic device. The argument `save.hist` can be used to save resulting histograms on external files. Letting `save.hist = TRUE`, histograms are saved on the current working directory but the user can specify a different directory by passing the absolute path through this argument. For example, letting `save.hist = paste0(getwd(), "/img")`, the figures will be saved in a subdirectory of the current working directory called `img`. Finally, the arguments `grdev` and `grdev.arg` are the graphics device to be used and the corresponding parameters.

The second plotting function is defined as follows:


```
qqcnorm(x, which, max.plot = 1L, save.plot = FALSE, grdev = pdf,
  grdev.arg, main = "Censored Normal Q-Q Plot",
  xlab = "Theoretical Quantiles", ylab = "Sample Quantiles",
  plot.it = TRUE, plot.pch = c(2L, 20L), plot.col = c(2L, 1L),
  plot.cex = c(2L, 1L), abline = FALSE, line.col = "gray50",
  line.lwd = 1L, line.lty = 2L, ...)
```

The meaning of the main arguments is the same as in `hist()`, so their description is omitted here but can be retrieved from the help page.

5. Model fitting: Technical details and further examples

The group of functions named *model fitting* represents the core group of the **cglasso** package. It provides the primary fitting functions, a set of method functions, a set of extractor functions to retrieve auxiliary results and, finally, the functions devoted to prediction and imputation. The full set of functions and their description is reported in Table 3.

5.1. Fitting conditional glasso models

`cglasso()` is the primary fitting function available in this group and it is formally defined as follows:

```
cglasso(formula, data, subset, contrasts = NULL, diagonal = FALSE,
  weights.B = NULL, weights.Tht = NULL, nlambda, lambda.min.ratio,
  lambda, nrho, rho.min.ratio, rho, maxit.em = 1.0E+4, thr.em = 1.0E-3,
  maxit.bcd = 1.0E+5, thr.bcd = 1.0E-4, trace = 0L)
```

In this function, `formula` is an optional argument by which the user can specify the model to be fitted by the usual symbolic description. If this argument is missing, `cglasso()` will automatically select the more suitable version of the `cglasso` estimator using the auxiliary

Function	Description
<code>cglasso()</code>	Conditional graphical lasso estimator
<code>print.cglasso()</code>	Print method for a 'cglasso' object
<code>plot.cglasso()</code>	Plot method for a 'cglasso' object
<code>coef.cglasso()</code>	Extract model coefficients
<code>fitted.cglasso()</code>	Extract model fitted values
<code>residuals.cglasso()</code>	Extract model residuals
<code>predict.cglasso()</code>	Predict method for 'cglasso' fits
<code>impute()</code>	Imputation of missing and censored values
<code>cggm()</code>	Post-hoc maximum likelihood refitting of a conditional graphical lasso
<code>print.cggm()</code>	Print method for a 'cggm' object
<code>plot.cggm()</code>	Plot method for a 'cggm' object
<code>predict.cggm()</code>	Predict method for 'cggm' fits

Table 3: Functions belonging to the group named *model fitting*.

Missingness mechanisms		X	Fitted model	
Censoring	MAR			
			glasso	(Friedman <i>et al.</i> 2008)
		✓	conditional glasso	(Yin and Li 2011)
✓			censored glasso	(Augugliaro <i>et al.</i> 2020a)
✓		✓	conditional censored glasso	(Augugliaro <i>et al.</i> 2020b)
	✓		missglasso	(Städler and Bühlmann 2012)
	✓	✓	conditional missglasso	(Städler and Bühlmann 2012)

Table 4: Relationship between the auxiliary information enclosed on a ‘`datacggm`’ object and the fitted model. Checkmark means that the auxiliary information is available. The first two columns refer to the mechanism of missingness for the response variables, that is, censoring or missing-at-random (MAR), whereas the third column refers to the presence or not of the predictor matrix. The last column reports the name of the fitted model.

information enclosed in the ‘`datacggm`’ object, which is passed through the argument `data`. Table 4 sums up the different implementations. For example, if the matrix `X` is not enclosed in `data`, then `cglasso()` fits a censored glasso model (Augugliaro *et al.* 2020a) or a missglasso model (Städler and Bühlmann 2012) depending on the information about the missing values contained in `data$Info` (see Section 4 for more details on this component). If the matrix `X` is included in `data`, then, starting from the `cglasso` package version 2.0.4, the function `cglasso()` can also fit conditional models with both censored (Augugliaro *et al.* 2020b) and missing-at-random (Städler and Bühlmann 2012) response values.

The function `cglasso()` has been designed in such a way that only the argument `data` is needed to fit a model, whereas the other arguments are optional, hence the user can avoid their specification if needed. Formally, `cglasso()` is a higher-level function designed only to test the correctness of the arguments; if the arguments are coherent with their definition, they are passed on to the function `cglasso.fit()` which calls the required fitting algorithm after having analyzed the entries enclosed in `data`. To ensure high computational efficiency, those fitting algorithms are written in Fortran.

We now describe the remaining optional arguments assuming that a matrix `X` is included in the ‘`datacggm`’ object. If it is not, then the arguments related to λ and to the regression coefficient matrix must be left unspecified. The arguments `weights.B` and `weights.Tht` are matrices for specifying non-negative weights for the regression coefficients and the precision values, respectively. The user can also use the entries of these matrices to specify a set of unpenalized parameters or a set of structural zeros. For example, letting `weights.Tht[h, k] = 0` then $\hat{\theta}_{hk}$ will be unpenalized whereas letting `weights.Tht[h, k] = +Inf` will set $\hat{\theta}_{hk}$ equal to zero. The next six arguments are used to specify the sequences of λ and ρ values used to fit the model. By default, `cglasso()` fits a sequence of models using a decreasing sequence of ten λ values and a decreasing sequence of ten ρ values. The length of these two sequences can be changed using `nlambda` and `nrho`, respectively. The largest λ and ρ values, denoted by λ_{\max} and ρ_{\max} , are computed using Theorem 3 in Augugliaro *et al.* (2020b). These values guarantee that only the estimated intercepts and the diagonal entries of the estimated precision matrix are nonzero. From a computational point of view, these estimates are computed by the EM algorithm implemented in the `datacggm()` function and returned by the extractor functions `ColMeans()` and `ColVars()`. The smallest values of the two sequences

of λ and ρ values, denoted by λ_{\min} and ρ_{\min} , respectively, are computed as a fraction of λ_{\max} and ρ_{\max} , that is, $\lambda_{\min} = \text{lambda.min.ratio} \times \lambda_{\max}$ and $\rho_{\min} = \text{rho.min.ratio} \times \rho_{\max}$. Hence `lambda.min.ratio` and `rho.min.ratio` are the arguments that can be used to avoid the computational problems related with the high-dimensional setting, that is, when the sample size is small with respect to the number of parameters. The default value of these two arguments is related to the sample size; if it is large enough these two arguments are set equal to $1.0\text{E-}6$ otherwise they are set equal to $1.0\text{E-}2$. The arguments `lambda` and `rho` can be used to pass a user supplied decreasing sequence of λ and ρ values, respectively, to the fitting algorithm. However, we suggest their use with care and avoid supplying a single λ or ρ value. The remaining arguments refer to the setting of the EM algorithm; for the sake of brevity their description is omitted here and we refer to the help-page of the `cglasso()` function for more details.

To elucidate the use of the `cglasso()` function, we report below an example based on the object `MKMEP.datacggm`, where the response variables are right censored (no missing-at-random values are recorded) and a predictor matrix is not available. Therefore, in accordance with Table 4, `cglasso()` will fit a sequence of censored glasso models using a decreasing sequence of ten ρ values:

```
R> MKMEP.mdl1 <- cglasso(data = MKMEP.datacggm)
```

Upon completion of the fitting algorithm, `cglasso()` returns a list with attribute `class` set to `'cglasso'`. The package `cglasso` provides a rich set of method functions specifically designed to help the user with the analysis of a `'cglasso'` object. The first one is the printing method function, which has been designed to print on screen the main summary statistics of the sequence of fitted models. Its formal definition is as follows:

```
print(x, digits = 3L, ...)
```

where `x` is an object of class `'cglasso'`, `digits` is an integer used to specify the number of printed digits and `...` is used to pass further arguments to the standard printing method function. To get some insight about its behavior, below we show the output printed on screen after a call to this function:

```
R> MKMEP.mdl1
```

```
Call:  cglasso(data = MKMEP.datacggm)
```

rho	df	(df%)	N. Comp.
8.4697	126	(6.061%)	63
7.5380	128	(6.157%)	61
6.6064	128	(6.157%)	61
5.6747	129	(6.205%)	60
4.7430	139	(6.686%)	51
3.8114	159	(7.648%)	40
2.8797	197	(9.476%)	30
1.9480	275	(13.228%)	17
1.0164	442	(21.260%)	3
0.0847	1539	(74.026%)	1

```

model: 'censored glasso'
  nObs: 48
  nResp: 63
  nPred: 0

```

For a model fitted without a predictor matrix, as in this example, `print()` organizes the output into two different parts. The first one reports information about the complexity of the fitted models and the topological structure of the corresponding estimated graphs. In detail, for each ρ value used, the output reports the degrees-of-freedom (`df`), defined as the number of nonzero estimates, and the number of connected components (`N. Comp.`), that is, the number of maximal connected subgraphs in which the estimated graph can be decomposed. For example, let us consider the results obtained using $\rho_{\max} = 8.4697$. As discussed above, this value is computed in such a way that the resulting estimated precision matrix is diagonal then the degrees-of-freedom are equal to the number of expected values plus the number of diagonal entries of the estimated precision matrix, that is, $\text{df} = 2p$. Furthermore, the estimated graph contains only isolated vertices, so the number of connected components is equal to p . As expected, reducing ρ will result in an increase in terms of degrees-of-freedom and, at the same time, a reduction in the number of connected components. From a computational point of view, decomposition of a graph into connected components gives rise to a reduction in the computational burden of the fitting algorithm as the original fitting problem can be decomposed into smaller subproblems. For more details about this aspect of the fitting problem, the interested reader is referred to [Witten *et al.* \(2011\)](#) and [Mazumder and Hastie \(2012\)](#). Finally, the last part of the printed text reports the name of the fitted model, the sample size, the number of response variables and the number of predictors, respectively. Finally, it is worth noting that the `print()` function silently returns an object of class `'data.frame'` containing the summary statistics printed on screen.

5.2. Plotting an object of class `'cglasso'`

To graphically evaluate the behavior of the estimates as the tuning parameters vary, the model fitting group provides the function `plot()`, defined as follows:

```

plot(x, what = c("Theta", "diag(Theta)", "b0", "B"),
     penalty = ifelse(x$nrho >= x$nlambda, "rho", "lambda"), given = NULL,
     GoF = AIC, add.labels, matplot.arg1, matplot.arg2, labels.arg,
     abline.arg, mtext.arg, save.plot, grdev = pdf, grdev.arg, digits = 4L,
     ...)

```

The argument `x` is an object of class `'cglasso'` that is the output of the fitting function `cglasso()`. The remaining arguments allows the user to customize several aspects of the resulting graph. For the sake of brevity, below we describe only the main arguments and we refer the user to the help page for more details.

The graphs produced by this method function closely depend on whether one has fitted a model with or without a set of predictors. For models whose vector of expected values is modeled by a set of q predictors, such as `MM.mdl`, the function `plot()` returns a conditional

coefficient path, that is a graph showing a set of estimated parameters as a function of a chosen tuning parameter while the remaining tuning parameter, called “conditional tuning parameter”, is held fixed to a specific value.

The user can specify a conditional coefficient path in two different ways, either via the three main arguments `what`, `penalty` and `given` or via a specific model formula. We start by describing the first method. In this case, the argument `what` is used to specify the set of estimated parameters reported on the y axis. By default, the estimated precision values (`what = "Theta"`) are reported on the y axis, but the user can also plot the diagonal entries of the estimated precision matrix (`what = "diag(Theta)"`), the p estimated intercepts (`what = "b0"`) or, finally, the estimated regression coefficients (`what = "B"`). The argument `penalty` is used to specify the tuning parameter reported on the x axis, that is ρ (`penalty = "rho"`) or λ (`penalty = "lambda"`) while `given` is an optional vector of integers used to identify one or more values of the conditional tuning parameter used during the fitting process. If this argument is not specified then `plot()` returns a sequence of graphs, one for each value of the conditional tuning parameter used to fit the model.

To gain more insight into how these three arguments are used, we report below the R code needed to get the conditional path of the estimated precision values. For illustration purposes, we fit the model using a subset of five response variables randomly drawn from the dataset `MM.datacggm`. Furthermore, to improve the quality of the resulting graph we use a decreasing sequence of one hundred ρ values and only two possible λ values:

```
R> set.seed(1234)
R> sample(nresp(MM.datacggm), 5L) |>
+   sort() -> Y.id
R> colNames(MM.datacggm)$Y[Y.id]

[1] "Y16" "Y22" "Y28" "Y37" "Y44"

R> mylambda <- c(0.2, 0.1)
R> MM.submdl <- cglasso(cbind(Y16, Y22, Y28, Y37, Y44) ~ .,
+   data = MM.datacggm, nrho = 100L, lambda = mylambda)
```

Given the object `MM.submdl`, the conditional path of the estimated precision values with respect to ρ and given the first λ value is obtained by the following R code:

```
R> plot(MM.submdl, what = "Theta", penalty = "rho", given = 1L,
+   matplot.arg1 = list(ylab = "Precision Values",
+   main = "Multiple Myeloma Dataset"), GoF = AIC)
```

Figure 7(a) shows the resulting graph, where we used the argument `matplot.arg1` to specify the title of the graph and the y axis label, and the argument `GoF` for the measure of goodness-of-fit corresponding to the optimal value of the tuning parameter reported on the x axis. In our example, we use the AIC (`GoF = AIC`) to identify the optimal ρ value given the first λ value used to fit the model. Although model selection is part of the graph returned by `plot()`, we postpone the description of how the model fitting evaluation is managed in the `cglasso` package to Section 6. Here, we just note that `GoF` can accept the name of a goodness-of-fit function, such as AIC or BIC, by which the fitted models are evaluated, or an object

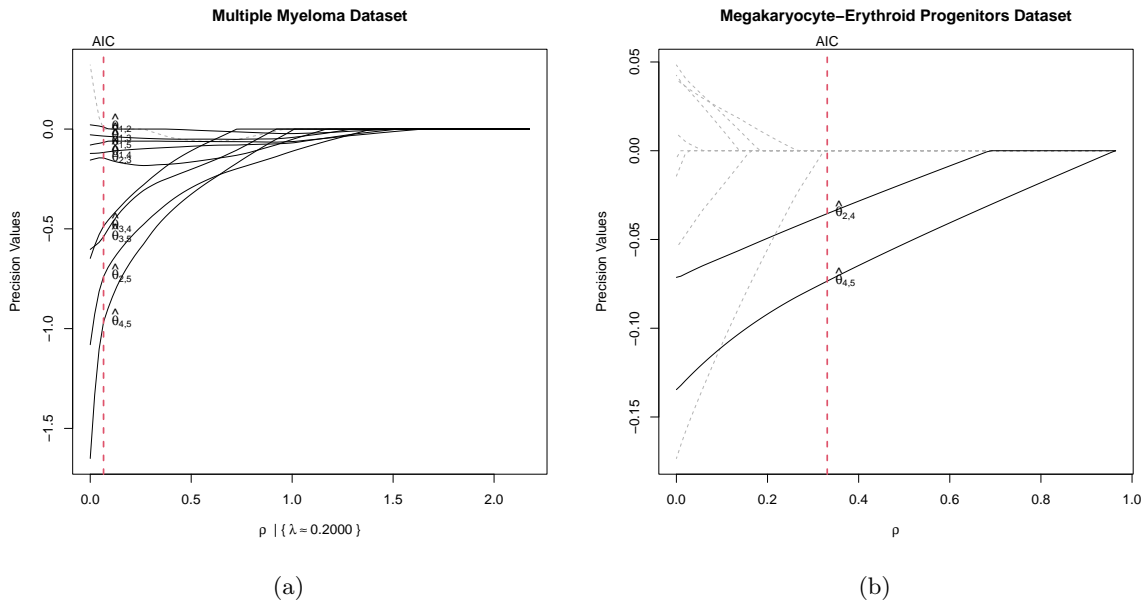


Figure 7: Output of the method function `plot()`. Vertical red dashed line identifies the optimal value of the tuning parameter reported on the x axis. By default, paths of the identified non-zero estimates are drawn using black solid lines whereas dashed gray lines are used to draw the path of the zero estimates.

of class `GoF`. If one lets `GoF = NULL`, then information about the model fitting evaluation is suppressed from the resulting graph.

As mentioned above, the user can also specify a conditional coefficient path using a model formula, whose general expression is of the form:

```
what ~ penalty | given
```

The following code shows how the argument `what` can be used to replicate Figure 7(a) using this second method:

```
R> plot(MM.submdl, what = Theta ~ rho | 1,
+       matplot.arg1 = list(ylab = "Precision Values",
+       main = "Multiple Myeloma Dataset"), GoF = AIC)
```

Finally, we discuss how to use `plot()` in the case of a model fitted without predictors. Since now the only tuning parameter is ρ , the argument `penalty` can be omitted while `given` must be left unspecified. As a consequence, the proposed plotting method function will return a standard coefficient path as default. Again, the argument `what` can be used to specify a set of estimated parameters or a model formula whose form is as follows:

```
what ~ rho
```

Since by default `what = "Theta"`, the function `plot()` will return as default the path of the precision values. We show this case without predictors by using the proposed plotting

method function to graphically evaluate a model fitted using a set of five random variables randomly drawn from the dataset `MKMEP.datacggm`. As previously done, we fit the model using a decreasing sequence of one-hundreded ρ values:

```
R> set.seed(1234)
R> sample(nresp(MKMEP.datacggm), 5L) |>
+   sort() -> Y.id
R> colNames(MKMEP.datacggm)$Y[Y.id]

[1] "Y16" "Y22" "Y28" "Y37" "Y58"

R> MKMEP.submdl <- cglasso(cbind(Y16, Y22, Y28, Y37, Y58) ~ .,
+   data = MKMEP.datacggm, nrho = 100L)
```

Figure 7(b) shows the path of the estimated precision values and it is obtained by:

```
R> plot(MKMEP.submdl, matplot.arg1 = list(ylab = "Precision Values",
+   main = "Megakaryocyte-Erythroid Progenitors Dataset"),
+   GoF = AIC)
```

As discussed above, in this case the argument `what` can be omitted.

5.3. Further method functions for ‘cglasso’ objects

The group of functions devoted to model fitting provides also a set of method functions for extracting the results of the fitting process from a ‘cglasso’ object. For illustration purposes, in what follows we will explain the use of these functions with a model fitted using a set of q predictors. In the other cases, the functions can be used by setting the arguments related to λ unspecified.

The estimated model coefficients can be extracted using the function `coef()`, which is defined as:

```
coef(object, type = c("all", "B", "Sigma", "Theta"), lambda.id, rho.id,
  drop = TRUE, ...)
```

where `object` is an object of class ‘cglasso’ and `type` is used to specify the required estimates. In particular, the user can extract the regression coefficient matrix (`type = "B"`), the covariance matrix (`type = "Sigma"`) or the precision matrix (`type = "Theta"`). Letting `type = "all"`, a named list enclosing all these estimates is returned. The next two arguments refer to the number of λ and ρ values used to fit the model, respectively. The user can extract the estimates corresponding to a specific pair of tuning parameters via the optional arguments `lambda.id` and `rho.id`, which are two vectors of positive integers specifying the position in the sequence of λ and ρ values used to fit the model.

For example, to extract from `MM.submdl` the estimated precision matrix corresponding to the pair:

```
R> c(lambda = MM.submdl$lambda[1L], rho = MM.submdl$rho[1L])
```

```

lambda      rho
0.200000 2.175208

```

we can use the following code:

```

R> Theta_hat <- coef(MM.submdl, type = "Theta", lambda.id = 1L,
+   rho.id = 1L)
R> dim(Theta_hat)

```

```
[1] 5 5
```

The output above shows that, by default, the unused dimensions are dropped. This can be avoided by setting `drop = FALSE`.

Similarly, the fitted values can be extracted by the following function:

```
fitted(object, lambda.id, rho.id, drop = TRUE, ...)
```

whose arguments have the same meaning as in `coef()`. For example, the code:

```

R> FittedValues <- fitted(MM.submdl, type = "Theta", lambda.id = 1L,
+   rho.id = 1L)

```

will extract the fitted values corresponding to the first λ and ρ values used to fit the `MM.submdl` model.

Finally, the model residuals can be extracted by the function:

```
residuals(object, type = c("observed", "working"), lambda.id, rho.id,
  drop = TRUE, ...)
```

Using this function, the user can extract two different types of model residuals. The former, called “observed” residuals (`type = "observed"`), is defined as the difference between the observed and the corresponding fitted response values. For missing and censored values, these residuals are set to NA. The later is called “working” residuals (`type = "working"`) and it is obtained as a byproduct of the EM algorithm used to fit the model. Formally, “working” residuals are defined as the difference between the working response matrix and the matrix of fitted values. The call of this function is similar to that of the fitted values. For example, the code:

```

R> WorkingResiduals <- residuals(MM.submdl, type = "working",
+   lambda.id = 1L, rho.id = 1L)

```

will extract the “working” residuals corresponding to the first λ and ρ values used to fit the `MM.submdl` model.

5.4. Prediction and missing data imputation using a ‘*cglasso*’ object

Using the results enclosed in a ‘*cglasso*’ object, the user can make predictions via the function


```
predict(object, type = c("B", "mu", "Sigma", "Theta"), X.new, lambda.new,
        rho.new, ...)
```

where `object` is the output of the `cglasso()` function. As done for the method functions described in the previous section, in what follows we shall assume that `object` is the output of a model fitted using a predictor matrix. If not, the arguments related to λ must be left unspecified.

We now describe the remaining arguments. Given a new pair of λ and ρ values, passed by the arguments `lambda.new` and `rho.new`, the user can choose among a number of quantities for which the prediction can be made. For example, letting `type = "B"`, the implemented function will use a bilinear interpolation to predict the estimated regression coefficient matrix, whereas letting `type = "mu"` the predicted fitted values will be returned. These values are computed as the product between a new predictor matrix, passed by `X.new`, and the predicted regression coefficient matrix. If a new predictor matrix is not available, that is argument `X.new` is left unspecified, then prediction is made using the matrix enclosed in the ‘`datacggm`’ object used to fit the model. Finally, the user can also ask for the prediction of the estimated covariance matrix (`type = "Sigma"`) and of the estimated precision matrix (`type = "Theta"`). As done for the regression coefficient matrix, these matrices are computed by bilinear interpolation.

As a possible example, we use the `MM.submdl` object to predict fitted values. As new λ and ρ values we use the average values of the λ and ρ values used to fit the model:

```
R> lambda.new <- mean(MM.submdl$lambda)
R> rho.new <- mean(MM.submdl$rho)
R> mu_pred <- predict(MM.submdl, type = "mu", lambda.new = lambda.new,
+   rho.new = rho.new)
```

The output of the `cglasso()` function can be used also to impute missing and censored data. This is achieved via the function `impute()`, whose formal definition is:

```
impute(object, type = c("mar", "censored", "both"), lambda.new, rho.new)
```

The arguments `object`, `lambda.new` and `rho.new` have the same meaning as in `predict()` whereas `type` is a description of the required imputation, that is of missing-at-random response values (`type = "mar"`), censored response values (`type = "censored"`) or both (`type = "both"`). For a given row of the response matrix, missing values are imputed using the expected values of a multivariate normal distribution conditional on the observed values whereas the censored values are imputed using expected values of a multivariate truncated normal distribution conditional on the observed values.

5.5. Post-hoc maximum likelihood refitting of a model

In some applications, the user may prefer to use the ℓ_1 -penalty function only as a tool for model selection, but then to estimate and evaluate the selected model by maximum likelihood. For this reason, we include in the package a second model fitting function, called `cggm()`. This function is specifically designed to fit a conditional Gaussian graphical model via a maximum likelihood method, with nonzero parameters identified by the model structure of a given ‘`cglasso`’ object.

Algorithm 4 Pseudo-code of the algorithm implemented with function `cggm()`.

- 1: let λ^{ini} and ρ^{ini} be two tuning values used to fit a model by `cglasso()`
- 2: extract the corresponding estimates: $\widehat{B}^{\text{ini}} = \{\widehat{\beta}_{ij}^{\text{ini}}\}$ and $\widehat{\Theta}^{\text{ini}} = \{\widehat{\theta}_{ij}^{\text{ini}}\}$
- 3: compute the matrices `weights.B` and `weights.Tht` with entries:

$$\text{weights.B}[i, j] = \begin{cases} +1 & \text{if } \widehat{\beta}_{ij}^{\text{ini}} \neq 0 \\ +\infty & \text{otherwise} \end{cases} \quad \text{weights.Tht}[i, j] = \begin{cases} +1 & \text{if } \widehat{\theta}_{ij}^{\text{ini}} \neq 0 \\ +\infty & \text{otherwise} \end{cases}$$

- 4: **for** $k = 1, \dots, \text{ntp}$ **do**
 - 5: fit a model using the Algorithm 1 with setting:
 - $\rho = \rho^{\text{ini}} - \frac{(k-1)}{(\text{ntp}-1)}(\rho^{\text{ini}} - \text{tp.min})$ and $\lambda = \lambda^{\text{ini}} - \frac{(k-1)}{(\text{ntp}-1)}(\lambda^{\text{ini}} - \text{tp.min})$
 - matrices of weights computed in Step 3
 - previous estimates as warm starting values
 - 6: **end for**
 - 7: return last fitted model
-

The function `cggm()` is therefore defined as follows:

```
cggm(object, GoF = AIC, lambda.id, rho.id, tp.min = 1e-06, ntp = 100L,
      maxit.em = 10000, thr.em = 1e-04, maxit.bcd = 1e+05, thr.bcd = 1e-04,
      trace = 0L, ...)
```

where `object` is an object of class ‘`cglasso`’.

To gain more insight into the meaning of the remaining arguments, Algorithm 4 reports the pseudo-code of the algorithm implemented to compute the maximum likelihood estimates for the case where `object` is the output of a model fitted using a predictor matrix. In particular, let λ^{ini} and ρ^{ini} be two tuning values used to fit a model by `cglasso()`. Then `cggm()` extracts the corresponding estimates and computes two matrices of weights with entries equal to 1 or $+\infty$, depending on whether the corresponding estimates are different from zero or not (Algorithm 4, Step 3). Further to this, the algorithm fits a sequence of penalized models by reducing simultaneously the tuning parameters from λ^{ini} and ρ^{ini} to a value close enough to zero (Algorithm 4, Step 4). The arguments `ntp` and `tp.min` are the length and the last value of the sequences of λ and ρ values used in this step of the algorithm, respectively. The matrices of weights ensure that the sparse structure of the initial estimates is held fixed during the computation of the entire sequence of fitted models. By construction, the last fitted model is unpenalized, i.e. it is fitted by maximum likelihood. To reduce the computational burden, the estimates obtained at a given point of the sequence are used as warm starting values for the next fitting problem. The arguments `maxit.em`, `thr.em`, `maxit.bcd` and `thr.bcd` have the same meaning as in `cglasso()` and are used for the implemented EM algorithm. From a computational point of view, `cggm()` works as a high-level function used to test the arguments passed to it, and to then call the function `cggm.fit()` which is a wrapper of a Fortran sub-routine implementing the algorithm reported in Algorithm 4.

The user can select λ^{ini} and ρ^{ini} in two different ways, either via the arguments `lambda.id` and `rho.id` which are two integers used to identify the λ and ρ values enclosed in `object`, respectively, (so for models fitted without a predictor matrix, the argument `lambda.id` must

be left unspecified), or via the argument `GoF`. In this case λ^{ini} and ρ^{ini} are selected using the output of a goodness-of-fit function, such as `AIC()` or `BIC()`. Again, we refer to Section 6 for a complete description of how the model selection step is managed in the `cglasso` package.

We complete this section by reporting an example on how to use the `cggm()` function. First, we use the `cglasso()` function to fit a sequence of models to `MM.datacggm` using twenty λ and ρ values:

```
R> MM.mdl2 <- cglasso(data = MM.datacggm, nlambda = 20L, nrho = 20L,
+   lambda.min.ratio = 0.2, rho.min.ratio = 0.2)
```

The arguments `lambda.min.ratio` and `rho.min.ratio` are used to reduce the computational burden of the overall algorithm. Then, we compute the maximum likelihood estimates of the model selected by BIC:

```
R> MM.mdl2.mle <- cggm(MM.mdl2, GoF = BIC)
R> MM.mdl2.mle
```

```
Call: cggm(object = MM.mdl2, GoF = BIC)
```

df		N. Comp.
448	(19.876%)	1

As shown below

```
R> class(MM.mdl2.mle)
```

```
[1] "cggm"      "cglasso"
```

the function `cggm()` returns an object of class `'cggm'` inheriting class `'cglasso'`, hence the method functions previously described can be used also on this output (see also Table 3). For example, by the following code:

```
R> Theta.MM.mdl2.mle <- coef(MM.mdl2.mle, type = "Theta")
```

we use the function `coef()` to extract the maximum likelihood estimate of the precision matrix of the model selected by BIC. In this case, the other arguments of the function `coef()` can be omitted since `cggm()` returns a single fitted model.

6. Model selection

The group of functions devoted to model fitting and model selection are intimately related to each other. The main link between the two is the object of class `'GoF'`, that is, a named list encapsulating all the relevant elements to evaluate the goodness-of-fit of the models fitted by `cglasso()` or `cggm()`. A `'GoF'` object is created through the use of the goodness-of-fit functions `AIC()` and `BIC()`. Inside the overall structure of the proposed `cglasso` package, the object of class `'GoF'` is designated to play the essential role of attaching the results from a chosen information criterion to the other functions used to evaluate, from different points of

Function	Description
<code>QFun()</code>	Extract Q -function
<code>print.QFun()</code>	Print method for a ‘QFun’ object
<code>AIC.cglasso()/BIC.cglasso()</code>	Goodness-of-fit functions
<code>print.GoF()</code>	Print method for a ‘GoF’ object
<code>summary.cglasso()</code>	Summarizing ‘cglasso’ and ‘cggm’ fits
<code>plot.GoF()</code>	Plot method for a ‘GoF’ object
<code>select.cglasso()</code>	Model selection for conditional graphical lasso estimator

Table 5: Functions belonging to the group named *model selection*.

view, the fitted models. For example, as seen in Section 5, an object of class ‘GoF’ can be passed to the function `plot()` through the argument `GoF`, to be shown as additional information on a conditional coefficient path plot, or it can be attached to the fitting function `cggm()` to select a specific model that will be refitted by a maximum likelihood method. Table 5 lists all the functions belonging to the model selection group along with their description.

6.1. Evaluating the goodness-of-fit

As discussed in the previous section, the objects of class ‘GoF’ are key for the evaluation of the fitted models and are returned by the goodness-of-fit functions. These functions compute a number of information criteria, of the type:

$$\text{measure of model fit} + a_n \times \text{measure of model complexity}, \quad (8)$$

where a_n is some positive sequence that depends only on the sample size and that controls the penalty on model complexity (Fan and Tang 2013). In particular, in our package, the model complexity is measured by the degrees-of-freedom, that is, the number of unique non-zero estimated parameters. The rationale of the information criteria for model selection is that the true model is the one that optimizes the information criterion (8). Clearly, the well known AIC (Akaike 1973) and BIC (Schwarz 1978) can be seen as special cases of the definition (8) with appropriate choices of the quantity a_n and of the measure used to quantify the complexity of the fitted model.

6.2. Measuring model fit

Traditionally, the log-likelihood is used as the measure of model fit in model selection criteria. However, the computational burden related to the evaluation of the log-likelihood for data with censored and missing values can be excessive also for models of moderate size. For this reason, following the approach originally proposed in Ibrahim, Zhu, and Tang (2008), in the *cglasso* package the model fit is measured using the Q -function:

$$Q(B, \Theta) = n/2[\log \det \Theta - \text{tr}\{\Theta \widehat{S}_{y|x}(B)\} - p \log(2\pi)],$$

where n is the sample size and $\widehat{S}_{y|x}(B)$ is the working empirical covariance matrix (4). The values of the Q -function are easily obtained as a byproduct of the M-Step of the Algorithm 1. The user can extract the Q values by the function `QFun()`, whose formal definition is reported below:

```
QFun(object, mle, verbose = FALSE, ...)
```

where `object` is an R object inheriting class ‘`cglasso`’. Argument `mle` is a logical value used to specify if the Q values should be computed using the penalized (`mle = FALSE`) or the maximum likelihood estimates (`mle = TRUE`). In the last case, the function `QFun()` acts as a high-level function calling `cggm()` to compute the maximum likelihood estimates and then to evaluate the Q -function. In this case, the argument `...` can be used to specify the arguments of the function `cggm()`. Finally, the argument `verbose` is a logical value for deciding whether to print out a progress bar on the R console. As discussed above, the computational burden related to the evaluation of the log-likelihood for data with censored and missing values can be excessive. Thus the proposed package does not implement a method function called by `logLik()`, but rather replaces this with the more efficient `QFun()` function.

Below is an example of how to extract the Q values:

```
R> mylambda <- c(1, 0.5)
R> MM.mdl1 <- cglasso(data = MM.datacggm, lambda = mylambda, nrho = 5L)
R> MM.mdl1.Qvalue <- QFun(MM.mdl1, mle = FALSE)
R> MM.mdl1.Qvalue
```

Q-values of the fitted 'conditional censored glasso' models

Details:

1 :

lambda	rho	df	Q-Values
1	4.54e+00	117	-6525
1	3.40e+00	133	-6498
1	2.27e+00	201	-5970
1	1.13e+00	438	-5453
1	4.54e-06	1274	-3748

2 :

lambda	rho	df	Q-Values
0.5	4.54e+00	185	-6266
0.5	3.40e+00	193	-6255
0.5	2.27e+00	193	-5968
0.5	1.13e+00	387	-5485
0.5	4.54e-06	1273	-3739

As the function `QFun()` returns an R object of class ‘`QFun`’, we have improved the readability of the output printed on screen by formatting the text using the same rationale of the `print()` function for an object of class ‘`cglasso`’, i.e., for each pair of tuning values we report the degrees-of-freedom (`df`) and the corresponding Q values (`Q-Values`). For models fitted without predictors, the printed section named `Details` contains a single section reporting the ρ values used.

6.3. The goodness-of-fit functions

Now we describe the two goodness-of-fit functions belonging to this group of functions. Given

the pair of estimates $(\hat{B}, \hat{\Theta})$, the first function, named `AIC()`, returns the values of the following AIC:

$$\text{AIC} = -2Q(\hat{B}, \hat{\Theta}) + k \text{ df},$$

where k is the penalty per parameter and `df` denotes the number of non-zero estimates. The function `AIC()` is defined as follows:

```
AIC(object, k = 2, mle, ...)
```

where `object` is an R object inheriting class `'cglasso'` and `k` represents the penalty for the degrees-of-freedom. The argument `mle` is passed to the `QFun()` to specify if the Q -function must be evaluated using the penalized estimates (`mle = FALSE`) or the maximum likelihood estimates (`mle = TRUE`). The default depends on the class of the fitted object. In particular, `mle` is `FALSE` for objects of class `'cglasso'` whereas it is `TRUE` for `'cggm'` objects. Below is an example where we use the penalized estimates from `MM.mdl1` to compute the AIC values:

```
R> MM.mdl1.AIC <- AIC(MM.mdl1, mle = FALSE)
R> class(MM.mdl1.AIC)
```

```
[1] "GoF"
```

As discussed above, the function `AIC()` returns an R object of class `'GoF'`, that is a named list enclosing a number of entries that will be used to evaluate the goodness-of-fit of a fitted model. In particular, it includes:

```
R> names(MM.mdl1.AIC)
```

```
[1] "value_gof" "df"          "dfB"          "dfTht"        "value"
[6] "n"          "p"           "q"            "lambda"       "nlambda"
[11] "rho"        "nrho"        "type"         "model"        "call"
```

Among these elements, one can find the values of the chosen information criterion (`value_gof`), the degrees-of freedom (`df`), the number of estimated non-zero regression coefficients (`dfB`) and the number of estimated non-zero partial correlation coefficients (`dfTht`). To improve readability, there is a specific method function for printing the output on screen:

```
R> MM.mdl1.AIC
```

Sequence of 'AIC' values of the fitted 'conditional censored glasso' models

Details:

```
1 :
  lambda      rho      df      AIC
1  4.54e+00    117    13283
1  3.40e+00    133    13263
1  2.27e+00    201    12342
1  1.13e+00    438    11781
1  4.54e-06   1274    10044
```

```

2 :
  lambda      rho      df      AIC
  0.5  4.54e+00   185  12901
  0.5  3.40e+00   193  12897
  0.5  2.27e+00   193  12323
  0.5  1.13e+00   387  11744
  0.5  4.54e-06  1273  10025

```

Since the model is fitted with a set of predictors, the output is structured by taking the two tuning parameters into account.

The second goodness-of-fit function is named `BIC()` and is defined as follows:

```
BIC(object, g = 0, type, mle, ...)
```

where the arguments `object` and `mle` have the same meaning as in `AIC()`, whereas the argument `type` allows the user to choose between two possible extensions of the classical BIC. In particular, letting `type = "FD"`, the default setting for models fitted without predictors, the `BIC()` function returns the values of the following extended BIC ([Foygel and Drton 2010](#)):

$$\text{eBIC}_{FD} = -2Q(\hat{B}, \hat{\Theta}) + (\log n + 4\gamma \log p) \text{ df},$$

where γ (argument `g`) is a tuning parameter belonging to the closed interval $[0, 1]$ (see [Foygel and Drton \(2010\)](#) for more details). It is easy to see that by letting $\gamma = 0$ (default setting for `g`), the previous criterion is formally equivalent to the classical BIC. Below is an example where we compute the extended BIC for the fitted models enclosed in `MKMEP.md11`. As suggested in [Foygel and Drton \(2010\)](#), we let $\gamma = 0.5$ and we use the maximum likelihood estimates to compute the Q values:

```

R> MKMEP.md11.eBIC <- BIC(MKMEP.md11, g = 0.5, type = "FD", mle = TRUE)
R> MKMEP.md11.eBIC

```

Sequence of 'eBIC_FD' values of the fitted 'censored glasso' models

Details:

rho	df	eBIC_FD
8.4697	126	15419
7.5380	128	15415
6.6064	128	15415
5.6747	129	15416
4.7430	139	15359
3.8114	159	15395
2.8797	197	15708
1.9480	275	16363
1.0164	442	17920
0.0847	1539	29332

The second type of the BIC is computed by letting `type = "CC"`, which is the default setting for models fitted using a set of q predictors. In this case, the `BIC()` function returns the values of the following extended BIC (Chen and Chen 2008):

$$eBIC_{CC} = -2Q(\hat{B}, \hat{\Theta}) + (\log n + 2\gamma \log q) \text{ df},$$

where $\gamma \in [0, 1]$. Below we use the `BIC()` function to compute the extended BIC proposed in Chen and Chen (2008) for the models enclosed in `MM.md11`. Again, we let $\gamma = 0.5$ and we compute the Q values using the maximum likelihood estimates:

```
R> MM.md11.eBIC <- BIC(MM.md11, g = 0.5, type = "CC", mle = TRUE)
R> MM.md11.eBIC
```

```
Sequence of 'eBIC_CC' values of the fitted 'conditional censored glasso'
models
```

```
Details:
```

```
1 :
```

lambda	rho	df	eBIC_CC
1	4.54e+00	117	13550
1	3.40e+00	133	13268
1	2.27e+00	201	12327
1	1.13e+00	438	12631
1	4.54e-06	1274	16579

```
2 :
```

lambda	rho	df	eBIC_CC
0.5	4.54e+00	185	13546
0.5	3.40e+00	193	13401
0.5	2.27e+00	193	12384
0.5	1.13e+00	387	12349
0.5	4.54e-06	1273	16574

We conclude this section by pointing out that, by default, the `BIC()` function computes the BIC values using the penalized estimates. In this case, `BIC()` works as a high-level function calling `AIC()` with argument `k` equal to $\log n$ and `mle` set to `FALSE`.

6.4. Summary of the fitted models

As previously advocated, the objects of class ‘GoF’ represent the link joining a number of functions devoted to the evaluation of model fit. In this section we describe how the information contained in a ‘GoF’ object can be attached to the output of the summary function:

```
summary(object, GoF = AIC, print.all = TRUE, digits = 3L, ...)
```

where `object` is the output of a model fitting function. The argument `GoF` allows the user to manage the part of the function `summary()` devoted to the model fit evaluation and can be used in two different ways.

The simplest way is by using an object of class ‘GoF’ created in a previous step. For example, below we attach the elements enclosed in `MM.mdl1.eBIC` to the summary of the object `MM.mdl1`:

```
R> summary(MM.mdl1, GoF = MM.mdl1.eBIC)
```

```
Call: cglasso(data = MM.datacggm, lambda = mylambda, nrho = 5L)
```

```
1 :
```

lambda	rho	df.B	df.Tht	df	(df%)	eBIC_CC	Rank
1	4.54e+00	68	49	117	(5.191%)	13550	8
1	3.40e+00	66	67	133	(5.901%)	13268	5
1	2.27e+00	53	148	201	(8.917%)	12327	1 <-
1	1.13e+00	49	389	438	(19.432%)	12631	4
1	4.54e-06	49	1225	1274	(56.522%)	16579	10

```
2 :
```

lambda	rho	df.B	df.Tht	df	(df%)	eBIC_CC	Rank
0.5	4.54e+00	136	49	185	(8.208%)	13546	7
0.5	3.40e+00	136	57	193	(8.563%)	13401	6
0.5	2.27e+00	56	137	193	(8.563%)	12384	3
0.5	1.13e+00	57	330	387	(17.169%)	12349	2
0.5	4.54e-06	49	1224	1273	(56.477%)	16574	9

```
=====
```

Summary of the Selected Model

```

model: 'conditional censored glasso'
nObs: 64
nResp: 49
nPred: 16
lambda: 1
lambda.id: 1
rho: 2.269075
rho.id: 3
eBIC_CC: 12327.04
df.B: 53
df.Tht: 148
df: 201
=====
```

As shown above, the output printed on screen is structured into two sections. The first one augments the results previously printed by `print()` function by adding the columns reporting the number of estimated non-zero regression coefficients (`df.B`) and the number

of estimated non-zero partial correlation coefficients (`df.Tht`). The last two columns, that is `eBIC_CC` and `Rank`, are inherited from `MM.md11.eBIC` and report the values of the chosen information criterion (the extended BIC in our example) and the corresponding ranking of the fitted models, respectively. Moreover, the fitted model minimizing the extended BIC is indicated by an arrow. Finally, the second section reports the primary summary statistics of the selected model.

Instead of using an object of class ‘GoF’, the second way in which the user can use the argument `GoF` is by specifying the name of a goodness-of-fit function and by passing further arguments of this function via the argument `...`. For example, the previous summary can be obtained also by the following code:

```
R> summary(MM.md11, GoF = BIC, g = 0.5, mle = TRUE)
```

From a computational point of view, with this code, the function `summary()` creates the needed ‘GoF’ object by calling `BIC()` with the arguments passed by `...`, thus essentially creating internally the object `MM.md11.eBIC`. The two approaches are clearly equivalent. However, from a practical point of view, we suggest to summarize the fitted models using a number of objects of class ‘GoF’ in a previous step of the analysis, that is following the first example for different information criteria. In this way, the results about the information criteria can be easily shared by the other functions without the need for re-calculating them.

Finally, the remaining arguments of the function `summary()` can be used to format the output printed on screen. For the sake of brevity, we refer to the help page for mode details.

6.5. Plotting method function

In order to graphically evaluate the results enclosed in a ‘GoF’ object, the model selection group provides the following plotting function:

```
plot(x, add.line = TRUE, arg.line = list(lty = 2L, lwd = 2L,
  col = "red"), add.text = FALSE, arg.text = list(side = 3L),
  arg.points = list(pch = 2L), ...)
```

where `x` is the output of a goodness-of-fit function whereas the remaining parameters can be used to customize the resulting graph. We refer the user to the help-page for more details. As done with the `plot()` function implemented for an object of class ‘*cglasso*’, the shape of the returned graph closely depends on whether the model has been fitted using a set of predictors.

For models fitted without predictors, `plot()` returns a graph showing the values of the chosen information criterion as a function of the tuning parameter whereas, for models fitted using a set of predictors, this function returns a contour plot with λ and ρ values reported on the x and y axis, respectively, whereas the values of the chosen measure of goodness-of-fit are used to compute the contour lines. Optimal λ and ρ values are identified using vertical and horizontal red dashed lines, respectively. For an application of this method function, we refer the reader to the Sections 3.1 and 3.2, where we use this function to evaluate the goodness-of-fit of the models fitted to the datasets `MM` and `MKMEP`, respectively.

6.6. Selection of the optimal fitted model

Once the optimal values of the tuning parameters have been identified by the functions

`summary()` and `plot()`, the user can recover the corresponding fitted model from a ‘`cglasso`’ object by the following extractor function:

```
select.cglasso(object, GoF = AIC, ...)
```

whose arguments have the same meaning as in `summary()`. This function returns an object of class ‘`cglasso`’, thus the user can then analyze the results enclosed in the returned model using the method functions described in the previous sections. We refer the reader to the Section 3.1 for an example of how to use this function.

7. Network analysis

The final group of functions of the proposed package, named *network analysis*, provides a set of functions to plot the graphs associated to the fitted model parameters. The main function of this group is `to_graph()` which returns an object of class ‘`cglasso2igraph`’, which provides a link with the R package **igraph** (Csardi and Nepusz 2006) and its functionalities. Table 6 lists all the implemented functions and their descriptions.

7.1. The objects of class ‘`cglasso2igraph`’

As discussed in the previous section, the key element enclosing the information about the estimated graph is the object of class ‘`cglasso2igraph`’, which is returned by the function `to_graph()`. Below we report the formal definition of this function:

```
to_graph(object, GoF = AIC, lambda.id, rho.id, weighted = FALSE,
  simplify = TRUE, ...)
```

where `object` is an R object inheriting class ‘`cglasso`’, whereas the arguments `GoF`, `lambda.id` and `rho.id` have the same meaning as in `cggm()`, i.e., they allow the user to select a fitted model enclosed in `object`. These arguments can be omitted if `object` encloses only one fitted model, such as, when it is the output of the functions `cggm()` or `select.cglasso()`.

As shown in Section 3.1, `to_graph()` returns a named list with class ‘`cglasso2igraph`’, containing the entries `Gyy` and `Gxy`. The first one is an object of class ‘`igraph`’ enclosing the information about the undirected graph that describes the conditional dependence structure among the p response variables. Formally, given an estimate of the precision matrix, say $\hat{\Theta} = (\hat{\theta}_{ij})$, the edge set enclosed in `Gyy` is defined as $\hat{\mathcal{E}}_{yy} = \{(i, j), \text{ such that } \hat{\theta}_{ij} \neq 0\}$. If `object` is the output of a model fitted using a set of predictors, then the entry `Gxy` is a directed graph used to describe the effects of the predictors on the expected values of the response

Function	Description
<code>to_graph()</code>	Create graphs from ‘ <code>cglasso</code> ’ or ‘ <code>cggm</code> ’ objects
<code>is.cglasso2igraph()</code>	Is an object of class ‘ <code>cglasso2igraph</code> ’?
<code>print.cglasso2igraph()</code>	Print method for a ‘ <code>cglasso2igraph</code> ’ object
<code>getGraph()</code>	Retrieve graphs from a ‘ <code>cglasso2igraph</code> ’ object
<code>plot.cglasso2igraph()</code>	Plot method for a ‘ <code>cglasso2igraph</code> ’ object

Table 6: Functions belonging to the group named *network analysis*.

variables. That is, given an estimate of the regression coefficient matrix, say $\hat{B} = (\hat{\beta}_{hk})$, the edge set enclosed in \mathbf{Gxy} is defined as $\hat{\mathcal{E}}_{xy} = \{(h, k), \text{ such that } \hat{\beta}_{hk} \neq 0\}$. For models fitted without predictors this entry is equal to `NULL`.

The argument `weighted` is logical and it is set equal to `TRUE` if \mathbf{Gyy} and \mathbf{Gxy} should be considered as two weighted graphs. In this case $\hat{\theta}_{ij}$ is used as a weight for the pair $(i, j) \in \hat{\mathcal{E}}_{yy}$ and $\hat{\beta}_{hk}$ as a weight for $(h, k) \in (i, j) \in \hat{\mathcal{E}}_{xy}$. Finally, the argument `simplify` can be used to remove isolated vertices from the returned graphs.

The user can test whether a given object is of class `'cglasso2igraph'` by the function

```
is.cglasso2igraph(x)
```

Finally, the user can extract the objects of class `'igraph'` from the output of the function `to_graph()` by the following extract function

```
getGraph(x, type = c("Gyy", "Gxy", "both"))
```

where `x` is an object of class `'cglasso2igraph'` and `type` is a string of characters used to describe the required graph. By letting `type = "both"`, the implemented extractor function will return a graph defined as a union of the graphs \mathbf{Gyy} and \mathbf{Gxy} . We refer to Section 3.1 and Section 3.2 for examples showing how to use the functions discussed in this section.

7.2. Plotting the estimated graph

The user can plot the estimated graph enclosed in an object of class `'cglasso2igraph'` by the following plotting function:

```
plot(x, type, ...)
```

where `x` is the output of the function `to_graph()` and `type` has the same meaning as in `getGraph()`. The argument `...` can be used to specify additional graphical parameters. From a computational point of view, `plot()` is a high level function calling `getGraph()` to recover the appropriate object of class `'igraph'`. Then, the estimated graph is plotted by calling the plotting method function implemented in the R package `igraph`. Like with the other plotting method functions, the graph returned by the proposed method function depends on whether the model is fitted using a set of predictors or not.

For models fitted using a set of predictors, by default, the function `plot()` returns a graph showing both the undirected graph of the p response variables and the directed graph describing the effects of the q predictors on the p response variables. For models fitted without predictors, this function returns the estimated undirected network of the p response variables.

8. Summary

In this paper we have presented the `cglasso` package, an R package providing comprehensive and user-friendly core routines for manipulation, simulation, visualization and analysis of multivariate data according to a conditional Gaussian graphical model, possibly featuring censored and/or missing values. The proposed R package is available from CRAN at <https://CRAN.R-project.org/package=cglasso>.

The **cglasso** package has been designed with the aim of helping the user with the analysis of multivariate data drawn from a conditional Gaussian graphical model, possibly featuring censored and/or missing values. This goal was achieved by structuring the proposed package as a sequence of four specific groups of functions.

The group named *data manipulation* contains both the main functions used to retrieve the internal representation of a dataset drawn from a conditional Gaussian graphical model and a set of specific method functions. The key element of this group is the object of class ‘**datacggm**’, which collects all information about the observed values in the data, including the pattern of censored/missing values and all other elements that may be useful for the other groups of functions of the proposed package. The group named *model fitting* is devoted to the fitting step of the implemented models and to the analysis of the corresponding results. The primary model fitting function, i.e., **cglasso()**, has been designed to fit a broad range of **cglasso** models and the appropriate version is automatically selected using the auxiliary information enclosed in a ‘**datacggm**’ object. The model fitting function returns an R object of class ‘**cglasso**’, whose results can be plotted and analysed by specific method functions. The third group, called *model selection*, concerns model evaluation. The goodness-of-fit functions **AIC()** and **BIC()** compute a number of extensions of the classical AIC and BIC criteria for the corresponding ‘**cglasso**’ object. These functions return an object of class ‘**GoF**’ which contain the elements needed for evaluating and comparing the fitted models through graphs and summary statistics. Finally, the group named *network analysis* is devoted to the statistical analysis of the fitted conditional independence graph. To this end, the main function of this group builds upon existing functions from the R package **igraph**, augmenting it with a set of specific plotting method functions. To improve the computational efficiency of the proposed package, all main algorithms are written in **Fortran** and called by specific wrapper functions. The **cglasso** package is being actively maintained and will be developed further to include additional features. Possible future developments may consider the implementation of penalized inference based on a group lasso penalty function (Yuan and Lin 2006) and the incorporation of more complicated censoring patterns both for response and predictor variables. We hope that the availability of the proposed package will facilitate the exploration of the implemented methodologies and their applications by statisticians and the R community.

Computational details

The results in this paper were obtained using R 4.2.1 with the **cglasso** package version 2.0.6. Results are based on a iMAC with a 3.3 GHz Intel Core i5 (6 core) processor running with macOS Monterey version 12.6. Results may be slightly different on other operating systems, but these differences are qualitatively negligible. R itself and all packages used are available from CRAN at <https://CRAN.R-project.org/>.

Acknowledgments

We thank the Editor, and the anonymous reviewer for thoughtful comments on our software implementation and article. Luigi Augugliaro and Gianluca Sottile gratefully acknowledge financial support from the University of Palermo (FFR2021). Gianluca Sottile acknowledges support by the Italian Ministry of University and Research (MUR) through the project PON-

AIM Attraction and International Mobility: AIM1873193-2 activity 1. Ernst C. Wit acknowledges support from the Fondazione Leonardo (514.7.010.098-4) and funding from the Swiss National Science Foundation (SNSF 188534).

References

- Akaike H (1973). “Information Theory As an Extension of the Maximum Likelihood Principle.” In BN Petrov, F Czaki (eds.), *Second International Symposium on Information Theory*, pp. 267–281. Akademiai Kiado, Budapest.
- Aragam B, Gu J, Zhou Q (2019). “Learning Large-Scale Bayesian Networks with the **sparsebn** Package.” *Journal of Statistical Software*, **91**(11), 1–38. doi:10.18637/jss.v091.i11.
- Augugliaro L, Abbruzzo A, Vinciotti V (2020a). “ ℓ_1 -Penalized Censored Gaussian Graphical Model.” *Biostatistics*, **21**(2), e1–e16. doi:10.1093/biostatistics/kxy043.
- Augugliaro L, Sottile G, Vinciotti V (2020b). “The Conditional Censored Graphical Lasso Estimator.” *Statistics and Computing*, **30**, 1273–1289. doi:10.1007/s11222-020-09945-7.
- Augugliaro L, Sottile G, Wit EC, Vinciotti V (2023). *cglasso: Conditional Graphical LASSO for Gaussian Graphical Models with Censored and Missing Values*. R package version 2.0.6, URL <https://CRAN.R-project.org/package=cglasso>.
- Chambers JM (1992). *Statistical Models in S*, chapter Classes and Methods: Object-Oriented Programming in S, pp. 445–480. Chapman & Hall, London. doi:10.1201/9780203738535.
- Chen J, Chen Z (2008). “Extended Bayesian Information Criteria for Model Selection with Large Model Spaces.” *Biometrika*, **95**(3), 759–771. doi:10.1093/biomet/asn034.
- Chen M, Ren Z, Zhao H, Zhou H (2016). “Asymptotically Normal and Efficient Estimation of Covariate-Adjusted Gaussian Graphical Model.” *Journal of the American Statistical Association*, **111**(513), 394–406. doi:10.1080/01621459.2015.1010039.
- Csardi G, Nepusz T (2006). “The **igraph** Software Package for Complex Network Research.” *InterJournal, Complex Systems*, 1695. URL <https://igraph.org/>.
- Derveaux S, Vandesompele J, Hellemans J (2010). “How to Do Successful Gene Expression Analysis Using Real-Time PCR.” *Methods*, **50**(4), 227–230. doi:10.1016/j.ymeth.2009.11.001.
- Fan J, Lv J (2010). “A Selective Overview of Variable Selection in High Dimensional Feature Space.” *Statistica Sinica*, **20**(1), 101–148. URL <https://www3.stat.sinica.edu.tw/statistica/j20n1/J20N12/J20N12.html>.
- Fan Y, Tang CY (2013). “Tuning Parameter Selection in High Dimensional Penalized Likelihood.” *Journal of the Royal Statistical Society B*, **75**(3), 531–552. doi:10.1111/rssb.12001.

- Foygel R, Drton M (2010). “Extended Bayesian Information Criteria for Gaussian Graphical Models.” In J Lafferty, C Williams, J Shawe-Taylor, R Zemel, A Culott (eds.), *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Vancouver. URL <https://proceedings.neurips.cc/paper/2010/file/072b030ba126b2f4b2374f342be9ed44-Paper.pdf>.
- Franzin A, Sambo F, di Camillo B (2017). “**bnstruct**: An R Package for Bayesian Network Structure Learning in the Presence of Missing Data.” *Bioinformatics*, **33**(8), 1250–1252. doi:10.1093/bioinformatics/btw807.
- Friedman JH, Hastie T, Tibshirani R (2008). “Sparse Inverse Covariance Estimation with the Graphical Lasso.” *Biostatistics*, **9**(3), 432–441. doi:10.1093/biostatistics/kxm045.
- Friedman JH, Hastie T, Tibshirani R (2019). **glasso**: *Estimation of Gaussian Graphical Models*. R package version 1.11, URL <https://CRAN.R-project.org/package=glasso>.
- Gramacy RB, Moler C, Turlach BA (2022). **monomvn**: *Estimation for MVN and Student t Data with Monotone Missingness*. R package version 1.9-16, URL <https://CRAN.R-project.org/package=monomvn>.
- Guo J, Levina E, Michailidis G, Zhu J (2015). “Graphical Models for Ordinal Data.” *Journal of Computational and Graphical Statistics*, **24**(1), 183–204. doi:10.1080/10618600.2014.889023.
- Gutiérrez NC, Sarasquete ME, Misiewicz-Krzeminska I, Delgado M, De Las Rivas J, Ticona FV, Ferriñán E, Martín-Jiménez P, Chillón C, Risueño A, Hernández JM, García-Sanz R, González M, San Miguel JF (2010). “Deregulation of microRNA Expression in the Different Genetic Subtypes of Multiple Myeloma and Correlation with Gene Expression Profiling.” *Leukemia*, **24**(3), 629–637. doi:10.1038/leu.2009.274.
- Hsieh CJ, Sustik MA, Dhillon IS, Ravikumar P (2011). “Sparse Inverse Covariance Matrix Estimation Using Quadratic Approximation.” In J Shawe-Taylor, R Zemel, P Bartlett, F Pereira, KQ Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates. URL <https://proceedings.neurips.cc/paper/2011/file/2ba8698b79439589fdd2b0f7218d8b07-Paper.pdf>.
- Ibrahim JG, Zhu H, Tang N (2008). “Model Selection Criteria for Missing-Data Problems Using the EM Algorithm.” *Journal of the American Statistical Association*, **103**(484), 1648–1658. doi:10.1198/016214508000001057.
- Jiang H, Fei X, Liu H, Roeder K, Lafferty J, Wasserman L, Li X, Zhao T (2021). **huge**: *High-Dimensional Undirected Graph Estimation*. R package version 1.3.5, URL <https://CRAN.R-project.org/package=huge>.
- Kunji KB (2022). **BigQuic**: *Big Quadratic Inverse Covariance Estimation*. R package version 1.1-11, URL <https://CRAN.R-project.org/package=BigQuic>.
- Lafferty J, McCallum A, Pereira FCN (2001). “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data.” In *Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001)*, pp. 282–289. URL <https://dl.acm.org/doi/10.5555/645530.655813>.

- Lauritzen SL (1996). *Graphical Models*. Oxford University Press, Oxford.
- Li B, Chun H, Zhao H (2012). “Sparse Estimation of Conditional Graphical Models with Application to Gene Networks.” *Journal of the American Statistical Association*, **107**(497), 152–167. doi:10.1080/01621459.2011.644498.
- Liu H, Lafferty J, Wasserman L (2009). “The Nonparanormal: Semiparametric Estimation of High Dimensional Undirected Graphs.” *Journal of Machine Learning Research*, **10**(80), 2295–2328. URL <https://jmlr.csail.mit.edu/papers/v10/liu09a.html>.
- Mazumder R, Hastie T (2012). “Exact Covariance Thresholding into Connected Components for Large-Scale Graphical Lasso.” *Journal of Machine Learning Research*, **13**, 781–794. URL <https://jmlr.org/papers/v13/mazumder12a.html>.
- Meinshausen N, Bühlmann P (2006). “High-Dimensional Graphs and Variable Selection with the Lasso.” *The Annals of Statistics*, **34**(3), 1436–1462. doi:10.1214/009053606000000281.
- Mohammadi R, Wit EC (2019). “**BDgraph**: An R Package for Bayesian Structure Learning in Graphical Models.” *Journal of Statistical Software*, **89**(3), 1–30. doi:10.18637/jss.v089.i03.
- Psaila B, Barkas N, Iskander D, Roy A, Anderson S, Ashley N, Caputo VS, Lichtenberg J, Loaiza S, Bodine DM, Karadimitris A, Mead AJ, Roberts I (2016). “Single-Cell Profiling of Human Megakaryocyte-Erythroid Progenitors Identifies Distinct Megakaryocyte and Erythroid Differentiation Pathways.” *Genome Biology*, **17**, 83–102. doi:10.1186/s13059-016-0939-7.
- R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Rothman AJ, Levina E, Zhu J (2010). “Sparse Multivariate Regression with Covariance Estimation.” *Journal of Computational and Graphical Statistics*, **19**(4), 947–962. doi:10.1198/jcgs.2010.09188.
- Schwarz G (1978). “Estimating the Dimension of a Model.” *The Annals of Statistics*, **6**(2), 461–464. doi:10.1214/aos/1176344136.
- Scutari M (2017). “Bayesian Network Constraint-Based Structure Learning Algorithms: Parallel and Optimized Implementations in the **bnlearn** R Package.” *Journal of Statistical Software*, **77**(2), 1–20. doi:10.18637/jss.v077.i02.
- Städler N, Bühlmann P (2012). “Missing Values: Sparse Inverse Covariance Estimation and an Extension to Sparse Regression.” *Statistics and Computing*, **22**(1), 219–235. doi:10.1007/s11222-010-9219-7.
- Sustik MA, Calderhead B, Clavel J (2018). **glassoFast**: *Fast Graphical LASSO*. R package version 1.0, URL <https://CRAN.R-project.org/package=glassoFast>.
- Trainor P, Wang H (2017). **BayesianGLasso**: *Bayesian Graphical Lasso*. R package version 0.2.0, URL <https://CRAN.R-project.org/package=BayesianGLasso>.

- Tseng P (2001). “Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization.” *Journal of Optimization Theory and Applications*, **109**(3), 475–494. doi:[10.1023/a:1017501703105](https://doi.org/10.1023/a:1017501703105).
- Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York. URL <http://www.stats.ox.ac.uk/pub/MASS4/>.
- Wang H (2012). “Bayesian Graphical Lasso Models and Efficient Posterior Computation.” *Bayesian Analysis*, **7**(4), 867–886. doi:[10.1214/12-ba729](https://doi.org/10.1214/12-ba729).
- Wang J (2015). “Joint Estimation of Sparse Multivariate Regression and Conditional Graphical Models.” *Statistica Sinica*, **25**(3), 831–851. doi:[10.5705/ss.2013.192](https://doi.org/10.5705/ss.2013.192).
- Williams DR, Mulder J (2020). “**BGGM**: Bayesian Gaussian Graphical Models in R.” *Journal of Open Source Software*, **5**(51), 2111. doi:[10.21105/joss.02111](https://doi.org/10.21105/joss.02111).
- Witten DM, Friedman JH, Simon N (2011). “New Insights and Faster Computations for the Graphical Lasso.” *Journal of Computational and Graphical Statistics*, **20**(4), 892–900. doi:[10.1198/jcgs.2011.11051a](https://doi.org/10.1198/jcgs.2011.11051a).
- Yin J, Li H (2011). “A Sparse Conditional Gaussian Graphical Model for Analysis of Genetical Genomics Data.” *The Annals of Applied Statistics*, **5**(4), 2630–2650. doi:[10.1214/11-aos494](https://doi.org/10.1214/11-aos494).
- Yin J, Li H (2013). “Adjusting for High-Dimensional Covariates in Sparse Precision Matrix Estimation by ℓ_1 -Penalization.” *Journal of Multivariate Analysis*, **116**, 365–381. doi:[10.1016/j.jmva.2013.01.005](https://doi.org/10.1016/j.jmva.2013.01.005).
- Yuan M, Lin Y (2006). “Model Selection and Estimation in Regression with Grouped Variables.” *Journal of the Royal Statistical Society B*, **68**(1), 49–67. doi:[10.1111/j.1467-9868.2005.00532.x](https://doi.org/10.1111/j.1467-9868.2005.00532.x).

Affiliation:

Luigi Augugliaro
Gianluca Sottile
Department of Economics, Business and Statistics
University of Palermo, Building 13
Viale delle Scienze, 90128 Palermo, Italy
E-mail: luigi.augugliaro@unipa.it, gianluca.sottile@unipa.it
URL: <https://www.unipa.it/persona/docenti/a/luigi.augugliaro>,
<https://www.unipa.it/persona/docenti/s/gianluca.sottile>

Ernst C. Wit
Università della Svizzera Italiana
Via Buffi 13, 6900 Lugano, Switzerland
E-mail: wite@usi.ch
URL: <http://www.math.rug.nl/~ernst>

Veronica Vinciotti
Department of Mathematics
University of Trento
via Sommarive 14, 38123 Povo (Trento), Italy
E-mail: veronica.vinciotti@unitn.it
URL: <https://webapps.unitn.it/du/en/Persona/PER0222143/Curriculum>