POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

Department of Mathematics

Master of Science in Mathematical Engineering

A method for the joint inference of multiple Gaussian graphical models in high-dimensional setting

Advisor: Prof.ssa Francesca Ieva, Politecnico di Milano

Correlators: Dr. Leonardo Bottolo, University of Cambridge

Dr. Gwenaël Leday, University of Cambridge

Candidate:

Ilaria Speranza

Matr. 854196

Academic Year 2016 - 2017

Contents

List of Figures

List of Tables

Sommario

Negli ultimi decenni, la crescente disponibilità di dati ha reso possibile lo sviluppo di modelli su larga scala per lo studio di molti fenomeni. I modelli grafici probabilistici descrivono in maniera compatta l'interazione tra le variabili, tramite una rappresentazione a grafo, nella quale i nodi corrispondono alle variabili aleatorie di interesse e gli archi alle interazioni probabilistiche tra di esse. In molte applicazioni, il problema sotto studio è modellato più accuratamente da una collezione di modelli grafici, piuttosto che da uno singolo. Ciò avviene quando i dati sono caratterizzati da una eterogeneità di cui si vuole tenere conto, ossia sono divisi in categorie: le variabili sono le stesse, ma interagiscono tra loro in maniera differente. Di conseguenza, alcuni archi saranno presenti in tutte le categorie, altri saranno specifici di un determinato gruppo. In questo contesto, la stima congiunta dei diversi modelli grafici permette lo scambio di informazione tra categorie, favorendo l'individuazione della struttura comune e riducendo la variabilità della stima, specialmente nelle categorie con poche osservazioni.
Scopo del presente progetto di tesi, sviluppato nel corso di un tirocinio di sei mesi presso la MRC Biostatistics Unit (Università di Cambridge), è lo sviluppo teorico di un metodo Bayesiano per la stima congiunta di una collezione di modelli grafici Gaussiani, che sia efficiente dal punto di vista computazionale.

Parole chiave: modelli grafici Gaussiani, categorie multiple, test d'ipotesi, prior coniugato, dati alto-dimensionali, efficienza computazionale.

Abstract

In the last decades data availability increased substantially. Thanks to that, several phenomena are now studied by means of large-scale models. The framework of probabilistic graphical models provides a mechanism to describe the interplay between variables in a compact way: this is achieved by using a graph-based representation, where the nodes correspond to the random variables of interest and the edges to direct probabilistic interactions between them. In many applications the estimation of a collection of graphical models is better suited than a single one to describe the problem under study. This happens when data are characterized by informative heterogeneity, i.e. they belong to different categories: the variables are the same, but they interact in different ways, so that some edges will be present in all categories, while others will be category-specific. In such cases, jointly estimating the multiple graphical models enables the borrowing of information across conditions: this favours the detection of a common structure and reduces the variance of the estimates, especially in categories with few observations.
The aim of my thesis project, carried out during a six months internship at the MRC Biostatistics Unit (University of Cambridge), is the theoretical development of a computationally efficient Bayesian method for the joint inference of multiple Gaussian graphical models.

Key words: Gaussian graphical models, multiple conditions, hypothesis testing, conjugate prior, high-dimensional setting, code optimization.

Introduction

In the last decades we witnessed a substantial increase in data availability, enabled by new technologies for their collection and storage. Thanks to that, several phenomena in various fields are now studied by means of large-scale models, involving complex relationships between tens of thousands or millions variables (Jordan et al. [2004]).
The framework of probabilistic graphical models provides a mechanism to describe the interplay between variables in a compact way: this is achieved by using a graph-based representation, where the nodes correspond to the random variables of interest and the edges to direct probabilistic interactions between them.
In the introduction to their book on probabilistic graphical models, Koller and Friedman [2009] present three advantages of this framework: first, the type of representation is transparent, so that an expert can easily evaluate its properties; second, the graph structure enables much faster inference of the posterior probability of some variables given evidence on others with respect to algorithms manipulating the joint distribution explicitly; third, this framework supports a very effective data-driven approach to model construction.
Graphical models are currently used in many fields, such as information retrieval, image and speech processing, communications and genomics; several methods have been developed for their inference, both in Bayesian and frequentist settings.
As clearly illustrated in the introduction of the paper by Guo et al. [2011], in many applications the estimation of a collection of graphical models is better suited than a single one to describe the problem under study. This happens when data are characterized by informative heterogeneity, i.e. they belong to different categories: the variables are the same, but they interact in different ways, so that some edges will be present in all categories, while others will be category-specific. As an example, we can consider gene networks describing different subtypes of the same cancer: there will be some shared sub-networks across multiple subtypes, and some links present only in a par-

ticular subtype. Detecting common sub-networks might be useful, especially when one of the subtypes is, say, uncurable: the discovery of similarities with another subtype that can be cured, may lead to the design of a new drug targeting the shared mechanism, thus curing also the first subtype. On the other hand, the detection of category-specific edges can help to pinpoint more precisely which genes are responsible for a specific mutation. Another example is the comparison between healthy subjects and patients affected by a particular disease: the inference of the differential network, composed by those edges that are present only in one of the two conditions, can reveal useful information to understand the causes of that specific disease.

In such cases, jointly estimating the multiple graphical models enables the borrowing of information across conditions: this favours the detection of a common structure and reduces the variance of the estimates, especially in categories with few observations.

Reviewing the existing literature, it can be evinced that many of them are designed to address only specific tasks, like detecting the differential (Tesson et al. [2010]) or common (Hara and Washio [2013]) network structure; some are restricted to the framework with only two conditions (Gill et al. [2010], Valcárcel et al. [2011]). The majority of the methods use regularization to assess the sparsity of the network (Danaher et al. [2014], Chiquet et al. [2011]), others, like Peterson et al. [2015], go Bayesian. Both approaches have specific points of strength and drawbacks: the former scales quite well in a high-dimensional setting, but the estimation heavily depends on the choice of the penalization hyperparameter, which is not straightforward; the latter is less parameter dependent, however it is computationally very demanding, so that it can be employed only when the number of features under analysis is small.

The aim of my thesis project, carried out during a six months internship at the MRC Biostatistics Unit (University of Cambridge), is the theoretical and computational development of a method for the joint inference of multiple Gaussian graphical models. The starting point has been the paper by Leday and Richardson [2018]. The method they introduce is Bayesian, with conjugate prior, so that the posterior can be computed analytically. Thus, no sampling phase is required and the method scales very well. The Bayesian model outputs a dense estimation, i.e. a network with all edges present: the subsequent phase consists in a multiple testing procedure to remove from the network all the edges for which the null hypothesis of being absent is not rejected.

The goal is to extend their Bayesian model to the multi-condition case, preserving the prior conjugacy and the error control, assessed through the hypothesis testing.

The difficult part to extend has been the graph structure recovery by means of multiple testing. Indeed, the new framework adds a further layer of multiplicity: besides having multiple edges to test simultaneously within each condition, we also have multiple conditions. To tackle this problem, we tried to adapt different approaches present in the literature to our case: Union-Intersection tests (Van Deun et al. [2009]), hierarchical testing (Goeman et al. [2011]) and hierarchical BH (Peterson et al. [2016]).
We will also introduce some concepts borrowed from information theory to provide an interpretable and well-defined notion of distance and, diametrically, of similarity to measure the relatedness between conditions.
Big effort has been put also in the phase of data simulation, necessary to evaluate the performances of our method. We had to simulate sparse positive definite matrices, representing the graph structure: this task becomes harder and harder as the number of features to include in the simulation increases.
The last part of the thesis project was devoted to the internal comparison between the uni-dimensional method applied independently to each condition, and the joint one: we found out that the benefit of joint estimation is not as significant as we hoped. Therefore, we propose two new methods for the estimation of the hyperparameters in the joint prior.
Together with the paper, Leday developed the R package `beam` which implements the proposed method. Also the computational part of our method, besides the theoretical one, tried to follow the single-condition framework. Consequently, the R package `beamDiff` we started developing for the joint estimation in the multi-condition case heavily relies on `beam`.
For this reason, the very first step has been the optimization of `beam`, mainly aimed at a better memory management. This has been beneficial for `beam` itself and for `beamDiff` in cascade.
The outline of this thesis is as follows.

- **Chapter 1: Background and preliminaries**. Multivariate Gaussian model and association graphs are revised; after that, the paper by Leday and Richardson [2018] is presented.
- **Chapter 2: R package 'beam'**. This chapter is devoted to the software development of R package `beam` and to the optimizations performed on it.
- **Chapter 3: Joint inference of multiple Gaussian graphical models**. An introduction to multi-condition framework is given; then the model we propose is presented.

- **Chapter 4: Computational development**. This chapter is dedicated to the R package `beamDiff` implementing our method and to data simulation.
- **Chapter 5: Results on simulated and real data.** Our method is tested on simulated data and on a real dataset containing the protein expression for 19 cancer types.

All the code written during my thesis project has been implemented using the statistical software R (R Core Team [2013]) and Rstudio (RStudio Team [2016]).

Chapter 1

Background and preliminaries

This thesis deals with Bayesian inference of Gaussian graphical models from high-dimensional data: this chapter is devoted to the presentation of some preliminaries about the Gaussian model, its peculiar relationship with graphical models and the Bayesian approach to the inference. In the last section the work by Leday and Richardson [2018] is presented.

1.1 Multivariate Gaussian models

Since we will assume a normal distribution for our data, we here introduce the nomenclature and some useful properties, for which we refer to Pearson [2001]. Denote a p-dimensional random variable that is normally distributed with mean $\mu = (\mu_1, ..., \mu_p)^T$ and covariance matrix $\Sigma = [\sigma_{ij}]_{i,j=1..p}$ by:

$$Y \sim \mathcal{N}(\mu, \Sigma).$$

We also denote the inverse of the covariance matrix by $\Sigma^{-1} = \Omega = [\omega_{ij}]_{i,j=1..p}$ and we will refer to it as the *precision matrix*. The partition of Y into two disjoint subsets indexed by $a \subset V = \{1, ..., p\}$, $\text{card}(a) = q$, and $b = V \setminus a$, $Y = [Y_a, Y_b]$, induces the partition of μ into $[\mu_a, \mu_b]$ and the following block-wise decomposition of Σ:

$$\Sigma = \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix},$$

with $\Sigma_{ba} = \Sigma_{ab}^T$.

Proposition 1.1. *The family of normal distributions is closed under marginalisation and conditioning, that is the marginal distribution of Y_a and the conditional distribution of Y_a given Y_b are again normal, and given by*

$$Y_a \sim \mathcal{N}(\mu_a, \Sigma_{aa})$$

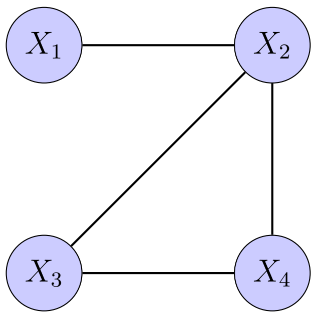


Figure 1.1: Example of undirected graph.

and

$$Y_a|Y_b \sim \mathcal{N}(\mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(Y_b - \mu_b), \Sigma_{aa.b}),$$

with $\Sigma_{aa.b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$*, respectively.*

It is interesting to use the blockwise inversion of Σ to link the elements of Σ to those of Ω:

$$\Omega = \begin{bmatrix} \Sigma_{aa.b}^{-1} & -\Sigma_{aa.b}^{-1}\Sigma_{ab}\Sigma_{bb}^{-1} \\ -\Sigma_{bb}^{-1}\Sigma_{ba}\Sigma_{aa.b}^{-1} & \Sigma_{bb}^{-1} + \Sigma_{bb}^{-1}\Sigma_{ba}\Sigma_{aa.b}^{-1}\Sigma_{ab}\Sigma_{bb}^{-1} \end{bmatrix}.$$

In particular we evince that $\Omega_{aa} = \Sigma_{aa.b}^{-1}$, which directly entails

$$\Sigma_{aa.b} = \Omega_{aa}^{-1}.$$

Exploiting this equality, the conditional distribution presented in Proposition 1.1 becomes:

$$Y_a|Y_b \sim \mathcal{N}_q(\mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(Y_b - \mu_b), \Omega_{aa}^{-1}). \tag{1.1}$$

These results show that the sub-matrix Σ_{aa} captures marginal dependencies between the variables indexed by a (Proposition 1.1), whereas the inverse of sub-matrix Ω_{aa} captures dependencies between the variables indexed by a conditional to (or adjusted by) the variables indexed by b (Equation 1.1).

1.2 Association graphs

In this section we introduce some basic notions and notations about graphs, from the book by Edwards [2012]
We denote with *graph*, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a structure consisting of a finite set $\mathcal{V}$ of *vertices* (also called nodes) and a finite set $\mathcal{E}$ of *edges* (also called arcs) connecting them. A graph is said to be *undirected* if all edges do not have a direction, i.e. there is a two-way relationship between the nodes connected

by an edge. We also assume that between two vertices one edge can be present at most once: multiple edges are not allowed. Lastly, it is possible to associate to each edge a weight, indicating the strength of the connection between nodes: in this case, we will talk about *weighted* graphs.
From this brief description, it is clear that association graphs are quite generic and flexible tools. We will work with undirected graphs, where each node corresponds to a feature and the edges represent dependence (marginal or conditional, as we will see later) between the pair of connected variables. A simple example with 4 nodes and 4 edges is reported in Figure 1.1.

1.3 Gaussian graphical models

We now investigate the particular relationship between multivariate Gaussian distribution and graphical models. Before starting, it is useful to recall the notions of marginal and conditional independence:

Definition 1.1. *Two random variables X_1 and X_2 are independent if their joint distribution factorizes in the product of the two marginal distributions:*

$$f_{X_1,X_2}(x_1,x_2) = f_{X_1}(x_1)f_{X_2}(x_2).$$

Definition 1.2. *Let X_1, X_2, and X_3 be three random variables. X_1 and X_2 are said to be conditionally independent given X_3 ($X_1 \perp\!\!\!\perp X_2|X_3$) if X_1 and X_2 are independent in the conditional distribution given $X_3 = x_3$, for each value x_3.*

The structure of marginal and conditional dependence is exactly what we would like to encode in our networks, so that an edge between two variables will indicate that they are related: in particular, marginal dependence means that the two variables affect reciprocally, but this might be due to one or more other variables they are linked to; conditional dependence means, instead, direct effect one on the other, after all the external effects have been removed by conditioning on them. The conditional network will usually be much sparser than the marginal one, since the edges represent a stronger, unmediated, dependence: for this reason, it is considered a more interpretable tool by the experts analysing the results.
As an example, let us consider the graph in Figure 1.1: assuming that it represents the conditional association structure, we can evince from it that $X_1 \perp\!\!\!\perp X_3|(X_2, X_4)$ and $X_1 \perp\!\!\!\perp X_4|(X_2, X_3)$.
Therefore, with networks we want to display the pairwise relationship between variables, say, i and j: let us then revise the properties of multivariate

Gaussian models, introduced in Section 1.1 when the partition is $a = \{i, j\}$ and $b = \{1, ..., p\} \setminus a$. Recalling Equation 1.1, the conditional distribution of $Y_{\{i,j\}}$ given "the rest" is a bivariate normal, with covariance matrix Ω_{aa}^{-1}, i.e. the block of Ω relative to variables i and j:

$$\Omega_{aa}^{-1} = \begin{bmatrix} \omega_{ii} & \omega_{ij} \\ \omega_{ij} & \omega_{jj} \end{bmatrix}^{-1} = \frac{1}{\omega_{ii}\omega_{jj} - \omega_{ij}^2} \begin{bmatrix} \omega_{jj} & -\omega_{ij} \\ -\omega_{ij} & \omega_{ii} \end{bmatrix}.$$

We will indicate the correlation coefficient of this distribution with ρ_{ij} and call it *partial correlation coefficient*:

$$\rho_{ij} = \frac{-\omega_{ij}}{\sqrt{\omega_{ii}\omega_{jj}}}.$$

It is immediate to see that

$$\rho_{ij} = 0 \Leftrightarrow \omega_{ij} = 0,$$

i.e. two variables are independent given the remaining ones if and only if the corresponding element in the inverse covariance matrix Ω is null.
Since we proved that all the information about partial independence is encoded inside the inverse covariance matrix, the first goal will be its estimation, followed by the application of a selection criterion to identify the elements significantly different from zero.

1.4 The Bayesian approach

As we showed in the last section, all the inference for Gaussian graphical models boils down to the estimation of the precision matrix Ω, which must be positive definite by definition and sparse, in order for the corresponding network to be meaningful and interpretable. To the best of our knowledge, there are two main approaches to deal with it: the first one is frequentist and based on the optimization of a penalized likelihood which encourages sparsity, the second is Bayesian and it can be thought as a problem of model selection. We will present a method belonging to the first category in Section 3.1, when talking about joint estimation across multiple conditions. Now we focus on the Bayesian approach, since this is what we are going to use for our model.
Bayes rule is the core of all Bayesian statistics: the goal is to update the a priori distribution of one or more unknown parameters given the data; the posterior is proportional to the product of prior and likelihood. In our case,

the parameter of interest is Ω, or its inverse Σ depending on the type of dependence, conditional or marginal, under study; more specifically, we want Ω to be sparse, for the reasons explained earlier, so that some of its entries must be identically zero.
We indicate with G the graph which constraints the structure of Ω: $\omega_{ij} = 0$ if and only if there is no edge in G connecting variables i and j. In a Bayesian setting, G is a random variable as well: we need therefore to specify a prior for it; however, this is the hardest and most challenging step when defining the model.
First thing to point out is that the space of $p \times p$ binary and symmetric matrices, over which G is defined, is very high-dimensional and grows super-exponentially with the number of features p. Indeed, given p variables, the underlying complete graph consists of $\frac{p(p-1)}{2}$ edges, and each of them can be either present or absent; therefore, there are $2^{\frac{p(p-1)}{2}}$ possible graphical models. When $p = 10$, for example, this number is in the order of 10^{13}. An exhaustive search over the entire space is computationally impossible.
Even assuming it is not, there are still some drawbacks we will outline for the easiest case of uniform prior over all the $2^{\frac{p(p-1)}{2}}$ possible models: being the probability distribution so spread, we will not have confidence on the Maximum A Posteriori (MAP) estimator, in the sense that its posterior probability will be still very small.
In order to overcome this problem, MAP estimator is usually replaced by a weighted average over all the models: G will then contain the edges with the highest posterior probability of inclusion. In any case, the model space must be necessarily narrowed and explored stochastically for the search procedure to be computationally feasible, but it is generally difficult to identify a good subspace, either small enough to be handled and rich enough to contain non-trivial graphical structures.
In Section 3.1, we will briefly describe a fully Bayesian algorithm for the inference of multiple Gaussian graphical models by Peterson et al. [2015]: the precision of their estimates gained through a complex hierarchical model comes at the prize of heavy computations, so that they can not go much further than a hundred features.
The method we will introduce in the next section is Bayesian as well, but it relies on a prior which is conjugate, so that the posterior distribution can be derived analytically, without resorting to MCMC algorithms: thanks to that, the algorithm scales very well with the number of features. The only way to remain conjugate is to estimate a dense matrix; sparsity will be achieved afterwards through element-wise hypothesis testing: the null hypothesis is $\omega_{ij} = 0$ (for partial independence, $\sigma_{ij} = 0$ for marginal independence), i.e.

the edge is not present in the network. All the edges for which H_0 is not rejected will be removed from the network and the corresponding element ω_{ij} in the precision matrix (or, analogously, σ_{ij} in the correlation matrix) will be set to 0.

1.5 Fast Bayesian inference in large Gaussian graphical models

This section is based on the work of Dr G. Leday and Prof S. Richardson, presented in Leday and Richardson [2018]. The paper proposes to perform graphical model selection by means of multiple testing of hypotheses about pairwise (marginal or conditional) independence, using closed-form Bayes factors. This approach has two strong points: it allows control of the type I error (which seems more difficult to achieve in the standard framework) and it is computationally very efficient.

1.5.1 The Gaussian conjugate model

Let Y be the $n \times p$ observation matrix; the probability distribution associated to it is the matrix normal, which is the generalization of the multivariate normal distribution to random matrices. Matrix and multivariate distributions are related in the following way:

$$Y \sim \mathcal{MN}_{n,p}(M, U, \Sigma) \quad \text{if and only if} \quad vec(Y) \sim \mathcal{N}_{n \times p}\left(vec(M), \Sigma \otimes U\right),$$

where $\otimes$ denotes the Kronecker product, $vec(M)$ the vectorization of M, and U and Σ are positive-definite real $n \times n$ and $p \times p$, respectively, covariance matrices.
In this setting, M is a zero matrix and U the n-dimensional identity I_n.
The prior for Σ is chosen to be the inverse-Wishart distribution, because it is conjugate, i.e. the posterior distribution of Σ is Inverse-Wishart, as well. This allows the derivation of closed-form Bayes estimators for Σ and Ω. Summing up, the overall Bayesian model is defined as follows:

$$\begin{array}{rcl} vec(Y)|\Sigma & \sim & \mathcal{N}_{n \times p}(0, \Sigma \otimes I_n) \\ \Sigma|M, \delta & \sim & \mathcal{IW}_p(M, \delta) \end{array} \tag{1.2}$$

Proposition 1.2. *The model described in 1.2 is conjugate and the posterior distribution is*

$$\Sigma|M, \delta, Y \sim \mathcal{IW}_p(M + S, \delta + n),$$

where $S = Y^T Y$.

1.5.2 Hyperparameters elicitation

To complete the model, prior hyperparameters must be elicited. They are the degrees of freedom δ and the scale matrix M of the inverse-Wishart distribution modelling Σ, as reported in Equation 1.2.
The expected value of a inverse-Wishart distributed $p \times p$ random matrix X with parameters M and δ, $X \sim IW_p(M, \delta)$ is given by

$$\mathbb{E}[X] = \frac{M}{\delta - p - 1}.$$

Let $T := M/(\delta - p - 1)$ be the target matrix, i.e. the matrix towards which the MLE estimator is shrunk.
The data Y are standardized ($Y_j^T I_n = 0$ and $Y_j^T Y_j / n = I_n$ for $1 \leq j \leq p$), therefore Σ will contain the correlation between the variables; in absence of prior information, the most reasonable choice is to set the target matrix T equal to the identity and, consequently, the hyperparameter M equal to $(\delta - p - 1)I_p$.
Hyperparameter δ needs to be chosen carefully, since it has a regularization effect, as we can see by writing down the posterior expectation of Σ:

$$\begin{aligned}\mathbb{E}[\Sigma|X] &= \frac{M+S}{\delta+n-p-1} = \frac{1}{\delta+n-p-1}M + \frac{n}{\delta+n-p-1}\frac{S}{n} \\ &= \frac{\delta-p-1}{\delta+n-p-1}T + \frac{n}{\delta+n-p-1}\hat{\Sigma}_{\text{mle}}.\end{aligned}$$

$\mathbb{E}[\Sigma|X]$ is a convex linear combination between the prior expectation $\mathbb{E}[\Sigma] = T$ and the maximum likelihood estimator $\hat{\Sigma}_{\text{mle}}$, weighted by $\alpha = (\delta - p - 1)/(\delta + n - p - 1) \in (0, 1)$, which indeed depends on δ.
Following previous papers, e.g. Hannart and Naveau [2014], δ is set to the value which maximizes the log-likelihood of the model. It can be obtained in closed-form and efficiently optimized, being convex:

$$\hat{\delta} = \underset{\delta}{\operatorname{argmax}} \log p_\delta(X),$$

where

$$\begin{aligned}\log p_\delta(X) = &-\frac{np}{2}\log\pi + \log\Gamma_p\left(\frac{\delta+n}{2}\right) - \log\Gamma_p\left(\frac{\delta}{2}\right) \\ &+ \frac{\delta}{2}log|M| - \frac{\delta+n}{2}\log|M+S|,\end{aligned} \tag{1.3}$$

being $S = X^T X$, $\Gamma_p(\cdot)$ the multivariate gamma function and the vertical lines $|\cdot|$ representing the determinant operator.

Setting $M = (\delta - p - 1)I_p$, Equation 1.3 can be simplified:

$$\begin{aligned} \log p_\delta(X) = &-\frac{np}{2}\log\pi + \log\Gamma_p\left(\frac{\delta+n}{2}\right) - \log\Gamma_p\left(\frac{\delta}{2}\right) \\ &+\frac{\delta p}{2}\log(\delta-p-1) - \frac{\delta+n}{2}\sum_{i=1}\log(\delta-p-1+d_i), \end{aligned} \tag{1.4}$$

being d_i the eigenvalues of S.

1.5.3 Bayes factors

Bayes factors for marginal independence

In order to build the graphical model associated with the data, it is necessary to test the presence of each single edge (i, j):

$$H^M_{0,ij} : \phi_{ij} = 0 \quad \text{against} \quad H^M_{1,ij} : \phi_{ij} \neq 0,$$

where $\phi_{ij} = \sigma_{ij}(\sigma_{ii}\sigma_{jj})^{-1/2}$ is the marginal correlation between variables i and j. The Bayes factor evaluating evidence in favour of the alternative hypothesis H_1 is

$$BF^M_{ij} = \frac{\int p_1(Y|\Sigma)p_1(\Sigma)d\Sigma}{\int p_0(Y|\Sigma^0)p_0(\Sigma^0)d\Sigma^0},$$

where Σ^0 is the covariance matrix Σ with $\sigma_{ij} = 0$. Through some reparametrizations and adopting a conditional approach to derive the distribution under the null hypothesis from that under the alternative, the following closed-form expression for the marginal Bayes factor is obtained:

$$BF^M_{ij} = \frac{\Gamma_2(\frac{\delta+n-p+2}{2})\Gamma^2(\frac{\delta-p+3}{2})}{\Gamma_2(\frac{\delta-p+2}{2})\Gamma^2(\frac{\delta+n-p+3}{2})}\frac{\left(1-r^2_{m,ij}\right)^{\frac{\delta-p+2}{2}}}{\left(1-r^2_{h,ij}\right)^{\frac{\delta+n-p+2}{2}}}\left(\frac{h^2_{ii}h^2_{jj}}{m^2_{ii}m^2_{jj}}\right)^{\frac{1}{2}},$$

where $m^2_{ii}, m^2_{jj}, r_{m,ij}, h^2_{ii}, h^2_{jj}$ and $r_{h,ij}$ are such that

$$\begin{aligned} M_{aa} &= \begin{bmatrix} m^2_{ii} & m_{ii}m_{jj}r_{m,ij} \\ m_{ii}m_{jj}r_{m,ij} & m^2_{jj} \end{bmatrix} \\ H_{aa} = M_{aa} + S_{aa} &= \begin{bmatrix} h^2_{ii} & h_{ii}h_{jj}r_{h,ij} \\ h_{ii}h_{jj}r_{h,ij} & h^2_{jj} \end{bmatrix}. \end{aligned} \tag{1.5}$$

The subscripts aa indicate the 2×2 submatrix of M and S respectively, with the elements associated to coordinates i and j.
As a final note, we can interpret H_{aa} as the Bayesian update of the prior matrix M_{aa}: from this perspective, $m^2_{ii}, m^2_{jj}, h^2_{ii}, h^2_{jj}, r_{m,ij}$ and $r_{h,ij}$ can be seen as prior marginal and posterior variances and correlations.

Bayes factors for conditional independence

Similarly to the marginal case, in order to evince the conditional independence structure it is necessary to define the proper hypothesis test for the partial correlation $\rho_{ij} = -\omega_{ij}(\omega_{ii}\omega_{jj})^{-1/2}$:

$$H^C_{0,ij} : \rho_{ij} = 0 \quad \text{against} \quad H^C_{1,ij} : \rho_{ij} \neq 0.$$

Again, through block-wise partition of the involved matrices induced by the partition of Y in (Y_a, Y_b), where $a = i, j$ and $b = V \backslash a$, and through a conditional approach for deriving the prior distribution under the null hypothesis, the Bayes factor in favour of $H^C_{1,ij}$ can be computed. Its expression is the following:

$$BF^C_{ij} = \frac{\Gamma(\frac{\delta+n}{2})\Gamma(\frac{\delta+n-1}{2})\Gamma^2(\frac{\delta+1}{2})}{\Gamma(\frac{\delta}{2})\Gamma(\frac{\delta-1}{2})\Gamma^2(\frac{\delta+n+1}{2})} \frac{(1-r^2_{g,ij})^{\frac{\delta}{2}}}{(1-r^2_{k,ij})^{\frac{\delta+n}{2}}} \left(\frac{k^2_{ii}k^2_{jj}}{g^2_{ii}g^2_{jj}}\right)^{\frac{1}{2}},$$

where $g^2_{ii}, g^2_{jj}, r_{g,ij}, k^2_{ii}, k^2_{jj}$ and $r_{k,ij}$ are such that

$$M_{aa.b} = \begin{bmatrix} g^2_{ii} & g_{ii}g_{jj}r_{g,ij} \\ g_{ii}g_{jj}r_{g,ij} & g^2_{jj} \end{bmatrix} \quad \text{and} \quad H_{aa.b} = \begin{bmatrix} k^2_{ii} & k_{ii}h_{jj}r_{k,ij} \\ k_{ii}k_{jj}r_{k,ij} & k^2_{jj}. \end{bmatrix}$$

with $M_{aa.b} = M_{aa} - M_{ab}M^{-1}_{bb}M_{ba}$ and $H_{aa.b} = H_{aa} - H_{ab}H^{-1}_{bb}H_{ba}$.

Scaled Bayes factors

Both marginal and conditional Bayes factors presented in the previous sections are not scale-invariant, due to their last multiplicative term. Since the next step of the method will involve their comparison in the multiple testing context, from now on their scaled version will be considered, thus defined:

$$sBF^M_{ij} = \frac{\Gamma_2(\frac{\delta+n-p+2}{2})\Gamma^2(\frac{\delta-p+3}{2})}{\Gamma_2(\frac{\delta-p+2}{2})\Gamma^2(\frac{\delta+n-p+3}{2})} \frac{\left(1-r^2_{m,ij}\right)^{\frac{\delta-p+2}{2}}}{\left(1-r^2_{h,ij}\right)^{\frac{\delta+n-p+2}{2}}} \tag{1.6}$$

for marginal independence, and

$$sBF^C_{ij} = \frac{\Gamma(\frac{\delta+n}{2})\Gamma(\frac{\delta+n-1}{2})\Gamma^2(\frac{\delta+1}{2})}{\Gamma(\frac{\delta}{2})\Gamma(\frac{\delta-1}{2})\Gamma^2(\frac{\delta+n+1}{2})} \frac{(1-r^2_{g,ij})^{\frac{\delta}{2}}}{(1-r^2_{k,ij})^{\frac{\delta+n}{2}}} \tag{1.7}$$

for conditional independence.

1.5.4 Structure estimation via multiple testing

So far, for each edge (i.e. couple of features) the estimation of marginal and partial correlation and the scaled Bayes factor have been computed: the higher this quantity, the higher the evidence towards the presence (if it is positive, absence if it is negative) of the edge in the network. Dealing with scaled quantities, the comparison between the $p(p-1)/2$ edges is facilitated, but they are still difficult to interpret, because the scale itself is not so meaningful. To address this problem, tail probabilities are derived using the null distribution of the Bayes factors; these new quantities are precisely probabilities, bounded between 0 (maximum evidence in favour of H_1) and 1 (maximum evidence in favour of H_0), to which standard adjustments can be applied, in order to account for multiplicity and control family-wise error or false discovery rate. The tail probabilities can be computed analytically: the most relevant considerations about how to derive them in the marginal case are here reported.
First of all, the only data-dependent term in the Bayes factor 1.6 is $r_{h,ij}$; then, recalling from 1.5 that $H_{aa} = M_{aa} + S_{aa}$, it can be evinced that $r_{h,ij} = (m_{ii}m_{jj}r_{m,ij} + s_{ii}s_{jj}r_{s,ij})(h_{ii}h_{jj})^{-1}$. When the prior matrix M is diagonal, $r_{m,ij} = 0$ and consequently $r_{h,ij}$ is proportional to $r_{s,ij}$, therefore the tail probability can be obtained from the distribution under the null hypothesis of $r_{s,ij}$:

$$pr(sBF^M_{ij})_{H_0} > (sBF^M_{ij})_{obs} = pr(r^2_{s,ij})_{H_0} > (r^2_{s,ij})_{obs}.$$

The following proposition will prove in a while to be useful in deriving the distribution of $r_{s,ij}$ (and, consequently, of the tail probabilities):

Proposition 1.3 (Proposition 3 of Leday and Richardson [2018]). *Suppose* $\Phi \sim W_2(\Sigma, d)$, *where*

$$\Phi = \begin{bmatrix} \phi_1^2 & \phi_1\phi_2\varphi \\ \phi_1\phi_2\varphi & \phi_2^2 \end{bmatrix} \quad \textit{and} \quad \Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & \sigma_2^2 \end{bmatrix}$$

are parametrised in terms of their correlations φ *and* ρ. *Then,*

$$(\varphi^2|\rho = 0) \sim Beta(1/2, (d-1)/2).$$

Recalling that, according to the proposed model, $S_{aa} \sim W_2(\Sigma_{aa}, n)$, the distribution of its squared correlation $r^2_{s,ij}$ under the null hypothesis of marginal independence can be derived, thanks to Proposition 1.3:

$$r^2_{s,ij}|H^M_{0,ij} \sim Beta(1/2, (n-1)/2).$$

Summing up, the tail probability of the Bayes factor for marginal independence can be computed exactly using $Beta(1/2, (n-1)/2)$. A similar result

is obtained in the conditional case.
To conclude, the underlying graph structure, starting from the dense estimations of Σ (marginal independence) and Ω (conditional independence), is retrieved by means of multiple hypothesis testing: the $p(p-1)/2$ tail probabilities, exactly derived from the Bayes factors, are adjusted to account for multiplicity using standard correction procedures (Bonferroni, Benjamini-Hockberg, Holm, ...) to control, say, family-wise error or false discovery rates.

Chapter 2

R package 'beam'

The method proposed by Leday and Richardson [2018] and presented in Chapter 1.5 has been implemented in the R package `beam` (Leday and Speranza [2018]), available on the CRAN repository.
This method shows two main innovation points with respect to others devoted to the same task of estimating graphical models: first of all, the entire procedure is closed-form, thus avoiding the sampling step, normally present in Bayesian algorithms. The sampling phase is computationally expensive, that is why fully Bayesian algorithms requiring MCMC sampling usually cannot deal with a large number of features. On the contrary, this algorithm scales very well in a high-dimensional setting. Secondly, having derived the tail probabilities using the exact distribution under the null hypothesis of the Bayes factors, type I error can be controlled, say, with family-wise error or false discovery rates. In the standard framework, such control would be more difficult to achieve.

2.1 Structure of the package

In this section the original structure of R package `beam` is presented. During the optimization process it has been slightly modified, as explained in the next section. The documentation of the optimized version of `beam`, available on the CRAN, can be found in Appendix A.
The package is composed by two main functions:

- `beam`: performs the first step of the algorithm, i.e. the estimation of the covariance matrix Σ and of its inverse Ω. For each entry of the two matrices Bayes factors, in logarithmic scale, and tail probabilities are also returned. This step of the algorithm is parameter-free, therefore

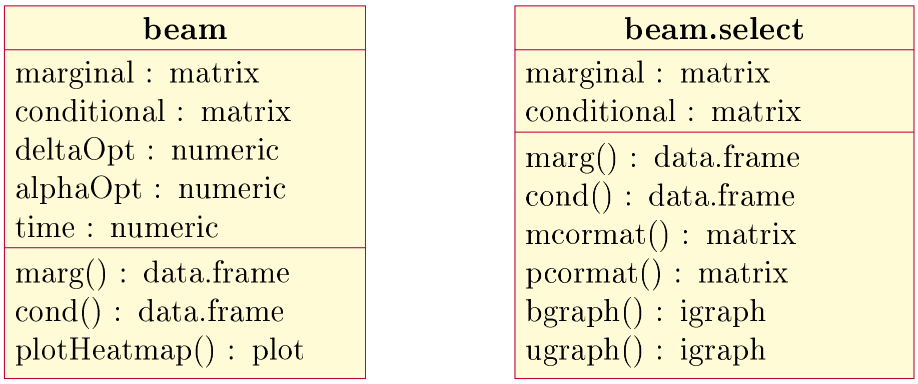


Figure 2.1: Simplified UML class diagrams for the S4 objects provided in `beam` package, with core slots and methods displayed.

no tuning phase is required, contrary to many other algorithms. Hyperparameter tuning can be either slow, if interested in an exhaustive search over the parameter space, and tricky, since it is often difficult to pinpoint the optimal values.

- `beam.select`: performs the selection, testing edge by edge whether it is present or not in the network. The user can set one or two parameters, depending on the multiplicity adjustment (Bonferroni, Holm, Benjamini-Hockberg, ...) he chooses: `thres` is the overall level of the test, by default set to 0.1, while $\mathtt{p}_0$, used by blfdr (Bayesian local false discovery rate) and BFDR (Bayesian false discovery rate), is the prior belief about the proportion of true null hypotheses (close to 1 for very sparse graphs, close to 0 for dense graphs).

The output of `beam` function is an S4 object of class `beam-class`. The most interesting quantities contained in this object are two matrices with $p(p-1)/2$ rows, corresponding to the number of edges present in the complete network (in this step no selection is performed yet), and the estimates of marginal and partial correlation (respectively), scaled Bayes factors and tail probabilities along the columns. Function `beam.select` takes the `beam` object as input and returns in output another S4 object of class `beam.select-class` containing only the edges which were selected as actually present in the network representing marginal and conditional independence structures, respectively. In addition to that, some methods acting on the S4 output objects, `beam` and `beam.select`, are provided: in general, they take the quantities of interest contained in the output and return them to the user in a useful and readable format. As an example, methods `bgraph` and `ugraph` take a `beam.select` object as input and return the marginal and conditional independence struc-

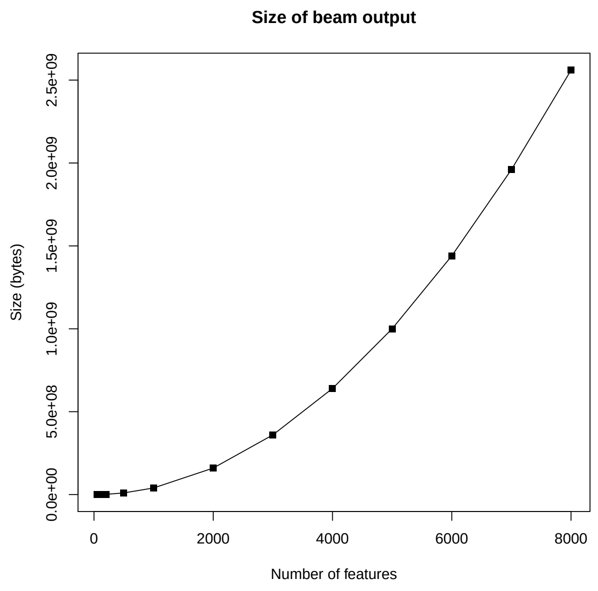


Figure 2.2: Size of beam object when $n = 71$ and $p \in [50, 8000]$

tures, respectively, as `igraph` objects, a format provided by the homonym R package `igraph` (Csardi and Nepusz [2006]), a broad library with lots of tools for network analysis and a good interface with other applications, like Cytoscape for network visualization.
The main slots and methods of `beam-class` and `beam.select-class` are reported in Figure 2.1.

2.2 Code optimization

2.2.1 Original version

The first version of `beam` package had some issues: besides few secondary bugs, mainly related to a lack of error handling, the main problem was the extremely large size of the output object `beam`. In Figure 2.2 we show how the size of the output increases with the number of features p, while the number of instances n is kept fixed at 71: as expected, we see that the growth is quadratic, like the number of edges with respect to p. When $p = 8000$, the size of the output is around 2.5 Gigabytes, and R can fail in storing it if in its environment there are already other big objects.
Therefore, the first goal of code optimization was to reduce the amount of memory needed for `beam` output. Many tips for achieving a better code have

```
     row col      m_cor    m_logBF  m_tail_prob
[1,]   1   2 -0.3499239 10.049308 3.897943e-08
[2,]   1   3  0.4387400 18.201709 1.260282e-12
[3,]   2   3 -0.2974358  6.326299 4.049561e-06
[4,]   1   4 -0.2740645  4.900681 2.389228e-05
[5,]   2   4  0.1193434 -1.345668 7.196609e-02
[6,]   3   4 -0.0550267 -2.454343 4.085861e-01
```

(a) slot "marginal"

```
     row col        p_cor     p_logBF p_tail_prob
[1,]   1   2  0.004152242 -0.8375797  0.87009330
[2,]   1   3  0.078558793  0.3446349  0.01889859
[3,]   2   3 -0.020700522 -0.7588028  0.44481223
[4,]   1   4 -0.009233312 -0.8245544  0.79493445
[5,]   2   4  0.010220344 -0.8208769  0.72228012
[6,]   3   4 -0.037582710 -0.5702007  0.32148008
```

(b) slot "conditional"

Figure 2.3: An example of slots "marginal" (a) and "conditional" (b) contained in S4 object `beam`.

been taken from the book about advanced programming in R by Wickham [2014].
Let us start by investigating in more detail the structure of the two tables returned in output and reported in Figure 2.3, since they are by far the biggest objects returned in output: each one is made of $p(p-1)/2$ rows and 5 columns, with two of them ('row' and 'col') in common.

2.2.2 Improvements and comparison

The first improvement simply consisted in returning a unique table, instead of two, in order not to duplicate the columns related to the indexing. Moreover, the data structure initially chosen to represent the tables in R was the matrix, which is not efficient for situations where the columns are not type-homogeneous, as in this case: indeed, all the elements inside a matrix must be of the same type, thus forcing the first two columns, containing only positive integers, to be treated as floats. In order to allow each column to use the best data type to represent its values, we decided to convert the matrix into a data frame, thus reducing the object size of 30%.
The last optimization step consisted in completely removing columns 'row' and 'col': that was possible after realizing that there is a one-to-one relation

$$\begin{bmatrix} - & 1 & 2 & 4 & 7 \\ - & - & 3 & 5 & 8 \\ - & - & - & 6 & 9 \\ - & - & - & - & 10 \\ - & - & - & - & - \end{bmatrix} \Leftrightarrow \begin{bmatrix} - & (1,2) & (1,3) & (1,4) & (1,5) \\ - & - & (2,3) & (2,4) & (2,5) \\ - & - & - & (3,4) & (3,5) \\ - & - & - & - & (4,5) \\ - & - & - & - & - \end{bmatrix}$$

(a) Matrix format

col	number of elems	elems indexes	first elem index	last elem index
1	0	-	-	-
2	1	1	1	1
3	2	2 3	2	3
4	3	4 5 6	4	6
5	4	7 8 9 10	7	10
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
p	$p-1$	...	$(p^2-3p+4)/2$	$p(p-1)/2$

(b) Table format

Figure 2.4: Relationship between (i, j) indexing and column-wise order of the upper triangular part of a square matrix.

between the row number and the position in the matrix. Indeed, we only store the upper triangular part, diagonal excluded being always equal to 1, of the matrix we are estimating, since it is symmetric. R command `upper.tri()` stores the upper triangular part of a matrix into a vector, following column-wise order, that is preserved in the output (first row corresponds to the first element in the upper triangular part, i.e. (1,2), the second corresponds to the second element according to the column-wise order, i.e. (1,3), etc.). The mapping described so far is shown in Figure 2.4a for a 5×5 matrix.
Our goal is to find the formula that, given the vector index n (upper triangular part stored in column-wise order), returns the corresponding pair of indexes (i, j) in the original matrix. In Table 2.4b we list in order the column index, how many elements of the upper triangular part the column contains, which are the indexes of the elements, in particular of the first and last ones. First of all, we notice that column j contains $j-1$ elements; this implies that n is in column j if

$$\sum_{k=1}^{j-1}(k-1) < n \leq \sum_{k=1}^{j}(k-1). \tag{2.1}$$

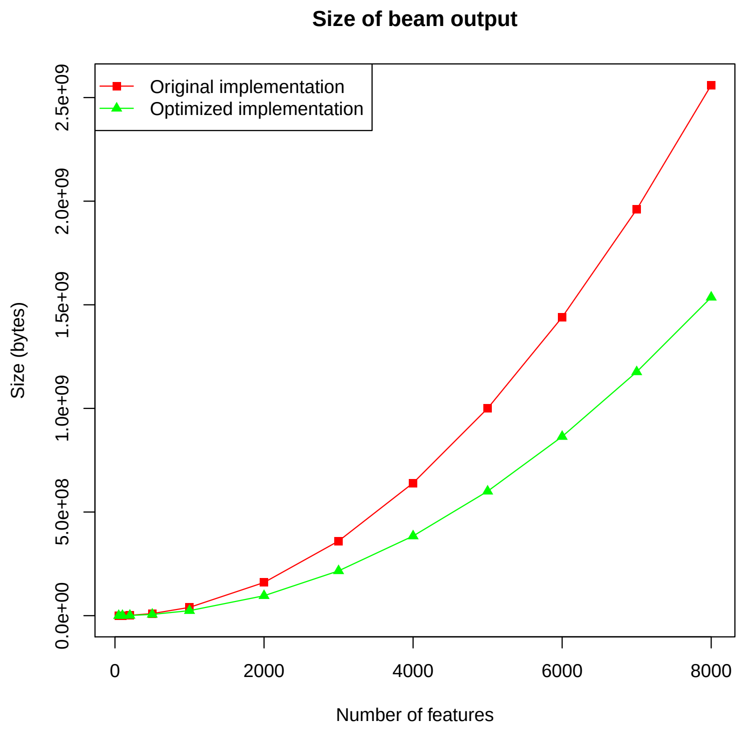


Figure 2.5: Comparison between original and optimized version of 'beam' code on the same data as figure 2.2.

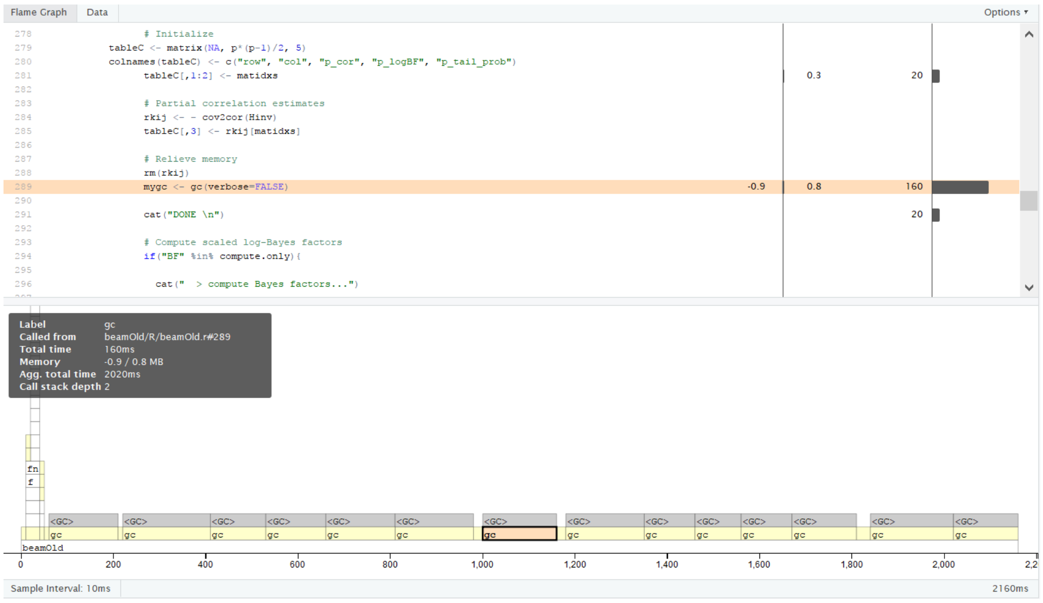
Solving the inequality we obtain:

$$j = \left\lceil \frac{1+\sqrt{1+8n}}{2} \right\rceil.$$

Now that we identified the column, we are left to find the row index i: recalling Equation 2.1, we know that the first element in column j is $\sum_{k=1}^{j-1}(k-1)+1 = \frac{(j-1)(j-2)}{2}+1$; the row index will therefore equal the difference between n and the first element of its column and is given by:

$$i = n + 1 - \frac{j^2 - 3j + 4}{2}.$$

Summing up, we moved from two 5-columns matrices to a unique 6-columns data frame, whose size is around 60% of the original solution, as displayed in Figure 2.5.
Our first goal was to save as much memory as we could, therefore we stored only the minimal information in the lightest format; the user will then access



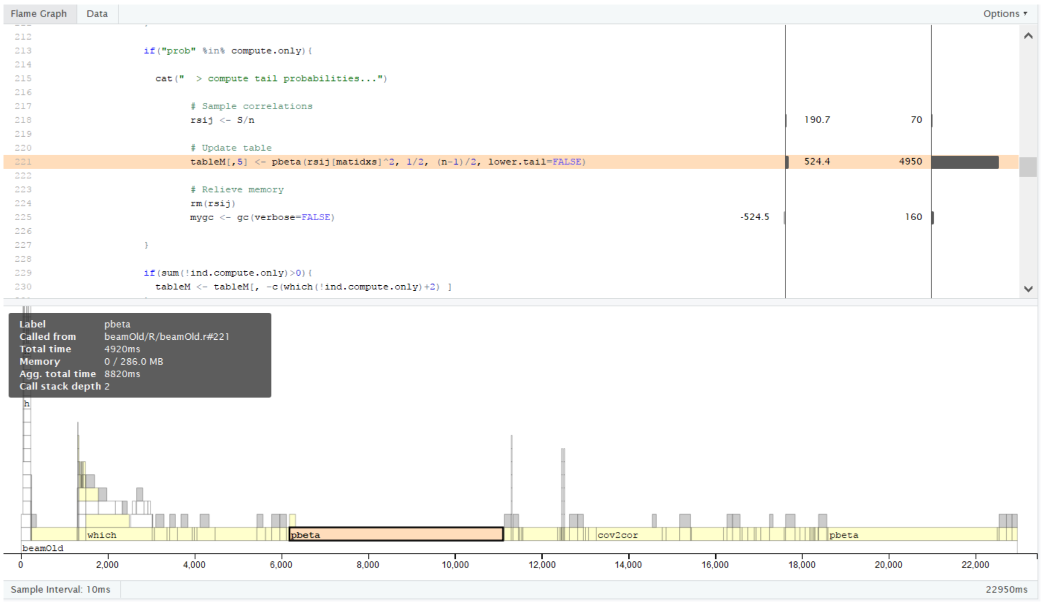
```
          # Initialize
      tableC <- matrix(NA, p*(p-1)/2, 5)
      colnames(tableC) <- c("row", "col", "p_cor", "p_logBF", "p_tail_prob")
          tableC[,1:2] <- matidxs                                           0.3    20

          # Partial correlation estimates
          rkij <- - cov2cor(Hinv)
          tableC[,3] <- rkij[matidxs]

          # Relieve memory
          rm(rkij)
          mygc <- gc(verbose=FALSE)                                  -0.9   0.8   160

          cat("DONE \n")                                                         20

          # Compute scaled log-Bayes factors
          if("BF" %in% compute.only){

            cat("  > compute Bayes factors...")
```

```
Label            gc
Called from      beamOld/R/beamOld.r#289
Total time       160ms
Memory           -0.9 / 0.8 MB
Agg. total time  2020ms
Call stack depth 2
```

Figure 2.6: profvis output, $n = 71$, $p = 100$

the output by using specific methods, provided in the package, which will display it in a readable way, adding for instance the indexing.
Moreover, the output is now without duplicates and type-homogeneous, since we removed the two columns regarding the indexing. Therefore, the most optimized data type to store it is the matrix: actually, we tried to come back to this kind of representation, but there was not any significant improvement, so we decided to stick to the data frame, which allows a slightly easier data manipulation.

After considerably improving the code under the memory point of view, we tried to optimize it also in time; to do so, we profiled the code with the R package `profvis` (Chang and Luraschi [2017]): this tool measures the time spent inside each function, so that it is easy to pinpoint the most time-consuming ones. This preliminary analysis is very important, because it is useless to waste energy in optimizing parts of code which take a negligible bit of time, while it is worth trying to improve, even a little, those functions which represent the bottlenecks of the code.
In Figure 2.6 we show an example of `profvis` output: on top, the code is displayed with information about time spent (right column), measured in milliseconds, and memory allocated/deallocated (left column) line by line. On the bottom is a flame graph, whose horizontal direction represents the time, while in vertical we have the call stack: in the figure we can read the total time in the bottom right corner, 2160 ms, and the correspondence be-



```
        if("prob" %in% compute.only){

          cat("  > compute tail probabilities...")

                # Sample correlations
                rsij <- S/n                                                          190.7        70

                # Update table
                tableM[,5] <- pbeta(rsij[matidxs]^2, 1/2, (n-1)/2, lower.tail=FALSE)  524.4      4950

                # Relieve memory
                rm(rsij)
                mygc <- gc(verbose=FALSE)                                   -524.5                160

        }

        if(sum(!ind.compute.only)>0){
          tableM <- tableM[, -c(which(!ind.compute.only)+2) ]
```

```
Label           pbeta
Called from     beamOld/R/beamOld.r#221
Total time      4920ms
Memory          0 / 286.0 MB
Agg. total time 8820ms
Call stack depth 2
```

Figure 2.7: profvis output, $n = 71$, $p = 5000$

tween upper and lower part of the screen enlighten in red, with the black box giving additional information on the selected function. In case of small p, as that presented in the example, the whole function is really fast and almost all time is spent in the several calls to the R function `gc()`, acronym for garbage collection: it forces R to return immediately the unused memory to the operating system. The repeated calls to this function inside `beam` code were meant to unload the memory burden as much as possible; however, many R guides claim that this procedure is useless, since R handles the process well by itself.
When p increases (Figure 2.7) this function is not the real bottleneck anymore, although we decided to remove it, after assuring that the code worked fine (and even better) without it. In the new example we can see that there are 3 functions which take a long time during `beam` execution: `which`, `pbeta` and `cov2cor`. The first two have been optimized, while `cov2cor` has not been modified, since it performs a necessary and unchangeable step of the algorithm.

- `which` function is used in the original version of `beam` to extract the indexes of a squared matrix corresponding to its upper triangular part. Indeed, being the matrices we want to estimate symmetric, it is enough to store one of the two halves.
  This is the line of code which performed the operation:

```
matidxs <- which(upper.tri(diag(p), diag=FALSE), arr.ind=
    TRUE)
```

We can see that a new $p \times p$ matrix, `diag(p)`, is created only for this specific purpose and then for each element of this matrix command `which` checks whether it belongs or not to the upper triangular part. As a matter of fact, no check is needed since the indexes of interest can be computed in advance, with the dimension of the matrix as the only required input. This is the new function:

```
.upperTriIdxs <- function(p){
z <- sequence(p)
cbind(
row = unlist(lapply(1:(p-1), function(x) 1:x), use.names
    = FALSE),
col = rep(z[-1], times = tail(z, -1)-1))
}
```

and below the modified line of code:

```
matidxs <- .upperTriIdxs(p)
```

- `pbeta` function is used twice during the algorithm for tail probabilities in marginal and conditional estimation, on each edge of the network, i.e. $p(p-1)/2$ times, therefore the time taken increases quadratically with the number of features. Per se, `pbeta` function cannot be optimized further, however we propose an approximated version, which can be employed when p gets bigger. In order to speed-up the computation, we evaluate `pbeta` only on a dense grid of 100001 values, equally spaced between 0 and 1, and then return for each edge the linear interpolation, as it is shown in the code below:

```
.approxBeta <- function(u, shp1, shp2, h=.00001){
x <- seq(from = 0, to = 1, by = h)
approx(x, pbeta(x, shape1=shp1, shape2=shp2, lower.tail =
     FALSE), yleft = 0, yright = 1, xout = u)$y
}

results$m_tail_prob <- .approxBeta(rsij^2, 1/2, (n-1)/2)

results$p_tail_prob <- .approxBeta(rfij2, 1/2, (n-1)/2)
```

Some simulations we run confirm that the edges selected are exactly the same with and without the approximation. As a final note, this

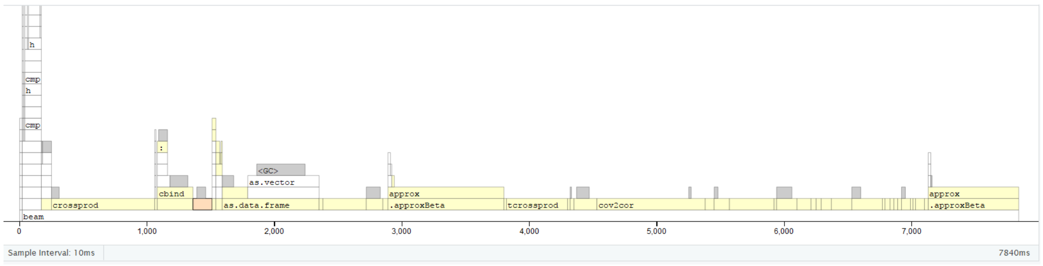
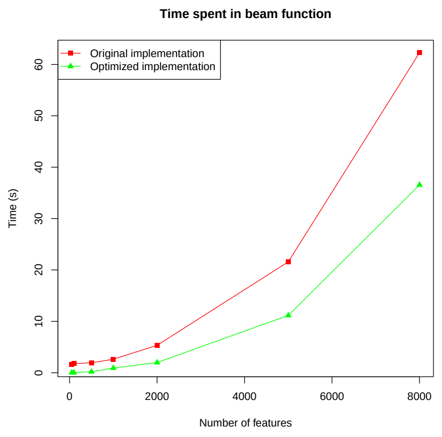


Figure 2.8: profvis output for optimized `beam` function, $n = 71$, $p = 5000$



p	original (SE)	optimized (SE)
50	1.651(0.138)	0.007(0.001)
100	1.771(0.073)	0.017(0.005)
500	1.935(0.061)	0.192(0.036)
1000	2.628(0.069)	0.906(0.020)
2000	5.337(0.544)	1.982(0.050)
5000	21.580(2.445)	11.153(0.213)
8000	62.286(13.073)	36.548(9.680)

Figure 2.9: Running time comparison between original and optimized `beam` versions with different number of features p. In the table, standard error (SE) over 10 simulations is also shown.

approximated version is faster than the original one when $p > 448$, as this is minimum integer value satisfying $p(p-1)/2 > 100001$.

With the optimizations just described, the algorithm runs much faster, as we can see from the new profiled code in Figure 2.8: now it takes about 10 seconds to complete (against the 20 of the original version) and the bottlenecks have been greatly reduced. In addition to that, we can notice that `gc` function is now automatically called by R in many points (grey rectangles at the top of the stack) in a much more optimized way.
In Figure 2.9 we show how the code has been improved in terms of running time for different values of p.

Chapter 3

Joint inference of multiple Gaussian graphical models

So far, we discussed how to deal with graphical models concerning a single set of data and we illustrated an algorithm based on Gaussian conjugate model and hypothesis testing. Let us now consider the case in which we have data related to multiple conditions (they can be cancer types, human tissues, healthy and ill patients). If for each of them we analysed the same features, e.g. we measured the expression of the same genes, it might be interesting to investigate not only the network structure underlying each condition, but also which are differences and commonalities across the conditions.
When multiple conditions are involved, two "easy" ways to analyse the data can be considered: overall estimation merging all conditions and group-by-group estimation. Regarding the former, even if on one side the number of observations increases (being the sum of the observations in each group), on the other, putting all data together, there is no possibility to perform any differential analysis and the heterogeneity is lost; the latter has instead the opposite problem, because it considers each condition independently, neglecting the underlying shared structure.
Starting from these unsuitable solutions for the multi-group network estimation, we derived two necessary characteristics any algorithm must have to deal properly with this problem: first of all, observations belonging to different conditions must be kept separate, so that commonalities and differences across groups can be appreciated; secondly, interaction between conditions must be present, to allow the expected underlying shared structure to emerge and act as anchor for the individual estimations.

3.1 State of the art

Before starting with the implementation of our method, we considered several papers to see which were the techniques used so far to deal with multi-network estimation; in particular, we investigated two in detail, since they were presenting very different methodologies, each with its points of strength and weakness.
In the paper by Danaher et al. [2014] Joint Graphical Lasso (JGL) is presented: the sparse and interdependent across groups network estimation is obtained by maximizing a double penalized version of the likelihood of the model. The first tuning parameter, present also in the single group case, controls the sparsity of each network, the higher the sparser; the second one, instead, penalizes the differences between corresponding elements in the precision matrices: the higher, the more the estimations across groups will be forced to be similar one to each other.
Compared to other algorithms, JGL scales quite well with the number of features (in the paper they claim they can get up to 20000), though quite slowly and only for specific ranges of the tuning parameters.
The main issue we met when trying this method (implemented in the R package `JGL`, Danaher [2013]) is related to the choice of the tuning parameters λ_1, for the individual sparsity, and λ_2, for the across-groups similarity: even if their intrinsic meaning is clear, in many real cases it is not known a priori how sparse and how similar the conditions under study are. The authors propose to tune, when possible, the two parameters according to practical criteria, such as network interpretability and stability; for instance, a sparse estimation is usually preferred.
When the knowledge about the data is not enough or a more objective method is desirable, λ_1 and λ_2 can be found by minimizing an approximated version of AIC score; when p gets larger, though, this second approach becomes quite slow and it might also happen that the provided values for the tuning parameters lead to an unreliable estimation with a (too) high false discovery rate.

The other paper we considered is by Peterson et al. [2015]: here, as in our framework, a Bayesian method to infer Gaussian graphical models is presented; however, it is not in closed-form, thus having strong limitations in terms of scalability.
In order to assess a sparse estimation, the prior on precision matrices Ω_k is distributed as a G-Wishart,

$$\Omega_k|G_k, b, D \sim W_G(b, D),$$

for $k = 1, 2, ..., K$. This prior constraints the non-zero elements of Ω_k in

correspondence of the edges of graph G_k; in other words, if edge $(i, j) \notin G_k$ then $\omega_{k,ij} = 0$. Every graph structure G_k is in turn random, and a Markov Random Field (MRF) prior is defined on it, in order to encourage common structure in related graphs. A further layer in the model is set to learn graph relatedness by defining a $K \times K$ symmetric matrix and by placing on it a spike-and-slab prior, so that graphs in non-related conditions will not be encouraged to share any common structure.
The method presented in Peterson et al. [2015] is very complex and accounts for all the issues generally arising in multiple network inference, like sparsity of the estimation and sharing of information across conditions, but this very complexity makes the algorithm unusable on data with more than 50 features, maximum: the examples shown in the paper deal with 20 features in the simulation study and with 17 on real data, 4 groups in both cases, and the algorithm takes already more than 2 hours.

3.2 Proposed model

Let us introduce the new setting and highlight the analogies with the unidimensional case by Leday and Richardson [2018] we are going to extend. We will refer to the number of conditions, also called groups, with K; p will be the number of features, which must be the same in all conditions under study, and n_k the number of observations for group k, $k = 1, ..., K$.
The model within each group is again the conjugate Gaussian-inverse-Wishart:

$$\begin{array}{rcl} vec(Y_k)|\Sigma_k & \sim & \mathcal{N}_{n_k \times p}(0, \Sigma_k \otimes I_{n_k}) \\ \Sigma_k|M, \delta & \sim & \mathcal{IW}_p(M, \delta) \end{array} \tag{3.1}$$

with $k = 1, ..., K$.
As before, scale matrix M can be expressed as a function of the target matrix T, $M = (\delta - p - 1)T$: we will use both notations.
It is important to note that the prior on the covariance matrices is independent of condition k. In this way we allow the sharing of information across groups by borrowing of strength and joint shrinkage estimation. This can be seen by writing down the posterior expectation of Σ_k

$$\mathbb{E}\left[\Sigma_k|Y_k\right] = \frac{\delta - p - 1}{\delta + n_k - p - 1} T + \frac{n_k}{\delta + n_k - p - 1} \hat{\Sigma}_k^{\text{mle}}, \tag{3.2}$$

where $\hat{\Sigma}_k^{\text{mle}} = n_k^{-1} S_k$ is the maximum likelihood estimator of Σ_k and $S_k = Y_k^T Y_k$: it is a convex linear combination between the maximum likelihood estimator $\hat{\Sigma}_k^{\text{mle}}$ and the prior expectation $\mathbb{E}\left[\Sigma_k\right] = T$. This means that all

the single Σ_k are jointly shrunk towards the common prior expectation T. Summing up, all estimates of Σ_k are anchored to a common target T with intensity regulated by hyperparameter δ.
Like the other methods described in the previous section, our model should output more precise estimations with respect to one-by-one estimation, especially when the number of observations is unbalanced between the groups.

3.3 Hyperparameter elicitation

Our first attempt for the choice of hyperparameters M and δ has been to use the same strategy used by Leday in the univariate case and here discussed in Subsection 1.5.2: we set $M = (\delta - p - 1)I_p$ and δ equal to the value maximizing the (logarithm of the) joint marginal likelihood of the model, available in closed-form thanks to prior conjugacy. We write here its general formulation

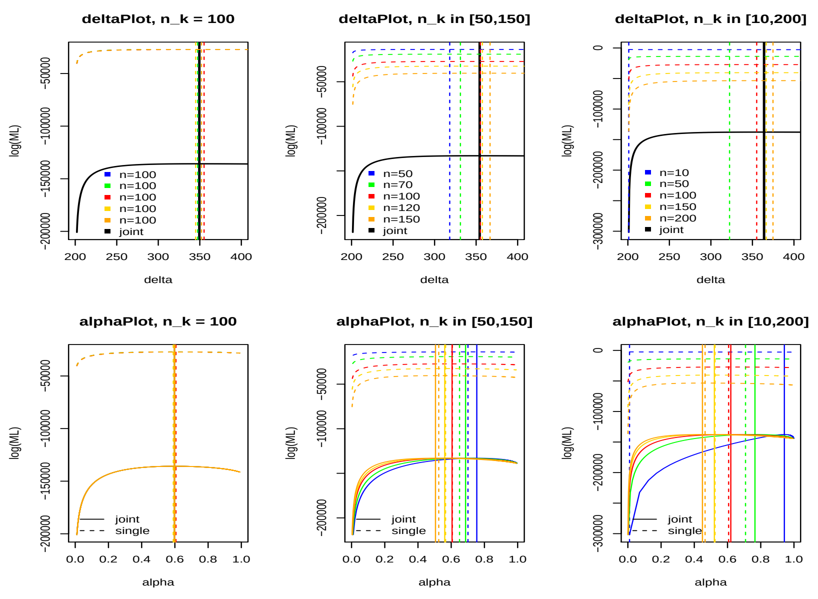
$$\begin{aligned}\sum_{k=1}^{K} \log p_\delta(Y_k) = &- \frac{p \log \pi}{2} \sum_{k=1}^{K} n_k + \sum_{k=1}^{K} \log \Gamma_p \left(\frac{\delta + n_k}{2} \right) + \frac{K\delta}{2} \log|M| \\ &- K \log \Gamma_p \left(\frac{\delta}{2} \right) - \sum_{k=1}^{K} \frac{\delta + n_k}{2} \log|M + S_k|,\end{aligned} \tag{3.3}$$

and that specific to our current case, with $T = I_p$:

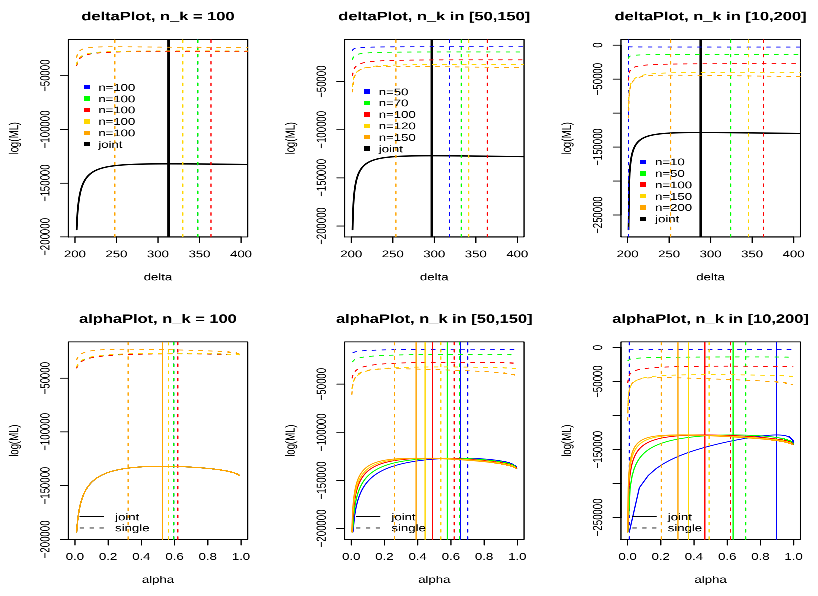
$$\begin{aligned}\sum_{k=1}^{K} \log p_\delta(Y_k) = &- \frac{p \log \pi}{2} \sum_{k=1}^{K} n_k + \sum_{k=1}^{K} \log \Gamma_p \left(\frac{\delta + n_k}{2} \right) \\ &+ \frac{K\delta p}{2} \log(\delta - p - 1) - K \log \Gamma_p \left(\frac{\delta}{2} \right) \\ &- \sum_{k=1}^{K} \frac{\delta + n_k}{2} \sum_{i=1}^{p} \log \left(\delta - p - 1 + d_i^{(k)} \right),\end{aligned} \tag{3.4}$$

where $d_i^{(k)}$ are the eigenvalues of $S_k = Y_k^T Y_k$.
Choosing M such that the prior expectation for Σ is equal to the identity has several advantages: first of all, it acts as regularizer for the MLE estimator which is usually ill-conditioned, specially when $n_k \ll p$; secondly, being diagonal, the derivation of tail probabilities is relatively straightforward. We refer to Subsection 1.5.4 for the details in the single-condition framework, which hold in the same way in the current joint inference setting.
These nice properties come at a prize: the prior is completely uninformative

and brings all posteriors close to the identity, i.e. to the empty graph. In the single-condition case, this shrinkage does not affect the quality of the performance; though, in our setting, by bringing all groups closer to the identity, we mask somehow the appearance of their common underlying structure. In other words, borrowing information across conditions by means of δ only, might be not enough to appreciate significant improvements with respect to the independent inference on each group individually.
Our concerns were confirmed by some numerical experiments: for a set of simulated data representing the multiple conditions (the simulation procedure is detailed in Section 4.2), we plot simultaneously the logarithm of each individual marginal likelihood (see Equation 1.4) and the joint one (Equation 3.4). We make the same comparison on the more interpretable scale given by $\alpha = (\delta - p - 1)/(\delta + n_k - p - 1)$. Results are shown in Figure 3.1: in both scenarios, totally (3.1a) and partially (3.1b) shared network structure, we consider three cases in which we vary the number of observations inside the conditions. We start on the left with constant $n_k = 100$, then, as we move to the right, we increase the imbalance between conditions ($n_k \in [50, 150]$ in the second column and $n_k \in [10, 200]$ in the third). To draw our conclusions, let us focus on the α scale, which is bounded between 0 and 1 and more interpretable, since the higher the value, the more the influence of the prior (see Equation 3.2).
When the underlying structure is exactly the same and the conditions are balanced (leftmost column in 3.1a), we can see that estimating individually or jointly is the same. As we increase the heterogeneity between conditions, by varying the number of observations and the underlying network structure, we start to appreciate some differences between the two estimations: however, they are significant only in very extreme scenarios (see for instance the blue line in the rightmost column, corresponding to the condition with 10 observations: from the alpha-plots we evince that this group borrows a lot of information from the others, and its corresponding optimal α_k moves from almost 0 to almost 1).
We will come back on this comparison between individual and joint estimation in Section 5.1: for the time being, we just point out that, even if it does not prove to be "practically" beneficial, still the joint approach is the proper way to model the multi-condition framework under a theoretical point of view.
After realizing that setting hyperparameter M equal to the identity was not optimal, we looked for new solutions, which could be more data-oriented: this is for instance achieved by using an empirical Bayes estimate also for M (we use it already for the estimation of hyper-parameter δ). Empirical Bayes approach, proposed by Morris [1983], allows the prior to be data dependent,



(a) Same covariance structure



(b) Partially shared covariance structure

Figure 3.1: Plots of log(marginal likelihood) as a function of δ (upper row) and $\alpha = (\delta - p - 1)/(\delta + n_k - p - 1)$ (lower row) when estimated independently group by group and jointly. In delta-plots, colored lines correspond to the 5 simulated conditions, the black one to the joint marginal likelihood; in alpha-plots, dashed lines refer to the individual estimation, solid one to the joint. Vertical lines are in correspondence of the optimal value, i.e. the maximum of the function.

in particular the aim in a multi-condition framework is to borrow strength from the ensemble.
Hypothetically, we could decide to optimize the joint marginal likelihood (Equation 3.3) in both its variables δ and M, but this solution is impracticable: the estimation of M requires in truth the estimation of $p(p-1)/2$ parameters, with the constraint of positive definiteness, therefore the problem is too high-dimensional to be numerically handled. Moreover, the joint marginal likelihood is proved to be concave only as a function of δ: the same property does not hold when considering also M variable (see Proposition 3.1).

3.3.1 EM algorithm for hyperparameters estimation

A method developed by Bilgrau et al. [2015] helped in this framework. Their model is Normal-inverse-Wishart as ours and they also estimate the hyperparameters of the inverse-Wishart distribution by maximizing the log-likelihood.
Before illustrating the method, we briefly introduce their framework, which is meta-analysis of covariance matrices. Nowadays, independent studies about gene expression are carried out by multiple research groups: each of them collects lots of data but only on relatively few subjects. It then becomes very useful to gather information across studies, so that the overall sample size increases. Doing that by simply merging all the data might lead to erroneous results, due to severe batch effects.
Bilgrau et al. [2015] propose a maximum likelihood estimator for the underlying common covariance matrix. As previously mentioned, they model the data as we do, i.e. according to the normal-inverse-Wishart distribution, reported in Equation 3.1. Accordingly, also the expression of the (log-) marginal likelihood will be the same we show in Equation 3.3: they propose a expectation-maximization (EM) algorithm for the joint optimization of M and δ. It consists in an iterative algorithm, where, in turn, $\hat{M}$ is updated keeping $\hat{\delta}$ fixed (expectation step) and $\hat{\delta}$ is optimized, given the current value of $\hat{M}$ (maximization step). This procedure continues until convergence, which is guaranteed under the only constraint $\sum_k n_k \geq p$.
We present here the relative propositions: for the proofs we refer to their paper.

Proposition 3.1. *For a fixed δ, the log-likelihood function 3.3 is not concave in M.*

Proposition 3.2. *For a fixed positive definite M, the log-likelihood function 3.3 is concave in δ.*

Proposition 3.3. *The log-likelihood 3.3 has a unique maximum in M for fixed δ and $N = \sum_{k=1}^{K} n_k \geq p$.*

At iteration $t+1$, EM algorithm updates the estimations for M and δ, $\hat{M}$ and $\hat{\delta}$ respectively, in the following way:

$$\hat{M}_{(t+1)} = \left(\frac{1}{K\hat{\delta}_{(t)}} \sum_{k=1}^{K} \left(n_k + \hat{\delta}_{(t)} \right) \left(\hat{M}_{(t)} + S_k \right) \right)^{-1}$$
$$\hat{\delta}_{(t+1)} = \underset{\delta}{\operatorname{argmax}} \sum_{k=1}^{K} \log p_\delta(Y_k, \hat{M}_{(t+1)}),$$

being K the number of groups and $S_k = Y_k^T Y_k$ the scatter matrix.
We implemented this algorithm and we tried it on several simulated datasets: it usually converges in less than 100 iterations and it is very fast, even in relatively high-dimension (we tried it up to $p = 500$ and $K = 10$).
However, we observed a severe drawback: the estimated $\hat{M}$ is often ill-conditioned or rather it is not well-conditioned enough to regularize the scatter matrices S_k, when summed-up in the posterior. To be clearer: contrary on the method proposed by Bilgrau et al. [2015], in our case the estimation of M is required for the hyperparameters elicitation of a Bayesian model. After that, our interest lies in the posterior distribution, especially of precision matrices Ω_k, which we recall to be

$$\Omega_k | Y_k \sim W_p \left((M + S_k)^{-1}, \delta + n_k \right).$$

We see that M has to be summed up to S_k and the resulting matrix $M_{post} = M + S_k$ has to be inverted. Our simulations show that M_{post} is generally ill-conditioned, specially in the groups with low sample size. When inverting it, a lot of numerical error is added to the estimation, which is consequently not reliable anymore. Numerical results are presented in Section 5.1.

3.3.2 Improved estimation of hyperparameters

Up to now, we analysed two methods for the estimation M: the first sets it equal to $(\delta - p - 1)I_p$, the second to the estimated common covariance structure. The former shrinks too much towards the empty graph, overcoming the borrowing of information, the latter regularizes too little, with consequent lack of accuracy in the estimation, due to ill-conditioning.

Since the goal is to estimate the underlying common structure and then use it as a prior for the joint inference, we decided to run `beam` function on all data merged together. Recalling that in the single group case the target matrix T was set to the identity (see 1.5.2), we expect the estimated overall covariance matrix to be well-conditioned and to reveal that common underlying structure we want to embed in M.
Let us see in more details the steps that bring to the estimation of M. As just mentioned, we run `beam` function on all data $Y_{tot} = Y_1 \cup ... \cup Y_K$; from it, we obtain the estimate of posterior expectation for the common covariance matrix $\hat{\Psi}$:

$$\hat{\Psi} := \mathbb{E}\left[\Sigma_{comm}|Y_{tot}\right].$$

We now recall that the expected value of an inverse-Wishart distributed random variable, $X \sim IW_p(M, \delta)$ is given by $M/(\delta - p - 1)$. By equalling it with the posterior expectation we get the following expression for M:

$$M = (\delta - p - 1)\hat{\Psi}.$$

Standard univariate (M is now expressed as a function of δ) optimization of log-likelihood function follows to find δ:

$$\delta_{opt} = \underset{\delta}{\operatorname{argmax}} \sum_{k=1}^{K} \log p_\delta(Y_k, \hat{\Psi}),$$

with

$$\begin{aligned}\log p_\delta(Y_k, \hat{\Psi}) = &-\frac{pn_k \log \pi}{2} + \log \Gamma_p\left(\frac{\delta + n_k}{2}\right) + \frac{\delta p}{2} \log|(\delta - p - 1)| \\ &+ \frac{\delta}{2}\log|\hat{\Psi}| - \log \Gamma_p\left(\frac{\delta}{2}\right) - \frac{\delta + n_k}{2} \log|(\delta - p - 1)\hat{\Psi} + S_k|.\end{aligned}$$

It is interesting to notice that this approach differs from the starting one only for the target matrix T, which here is equal to $\hat{\Psi}$, instead of the identity: this means that the posteriors will be now shrunk towards a target that somehow represents the underlying common structure.
As we will see in Section 5.1, the simulations show that this approach to the estimation of M is very promising: the results are well-conditioned and generally better than the other two.
However, being it not diagonal, the expression of tail probabilities has yet to be derived: indeed, the derivation of tail probabilities reported in 1.5.4 is under the assumption of diagonal prior matrix.
For this reason, in this Master thesis we will adopt the first approach, with $T = I_p$.

3.4 Group comparisons using measures from information theory

Even though our final goal is the network estimation inside each group, we can already, with no selection carried out yet, provide useful information regarding the relative distance between groups. Being Bayesian and working with conjugate prior, we have closed-form inference for the posteriors of Σ_k and, analogously, of Ω_k:

$$\begin{aligned}\Sigma_k &\sim IW_p(M + S_k, \quad \delta + n_k)\\ \Omega_k &\sim W_p((M + S_k)^{-1}, \quad \delta + n_k).\end{aligned} \tag{3.5}$$

We can, therefore, rely on some statistical distances defined between probabilistic distributions, rather than simply comparing the posterior expectations. We will focus on two of them, broadly used also in information theory: cross entropy and Kullback–Leibler (KL) divergence, which are closely related, as we will see in a while. We give the definitions of the two distances in the continuous case, since this is our current setting, although both of them were originally thought for discrete distributions.

Definition 3.1. *Let P and Q be the distributions of a continuous random variable, and p and q their density functions. Then the cross entropy between P and Q is*

$$H(P, Q) = -\int_{-\inf}^{+\infty} p(x) \log q(x) dx = \mathbb{E}_P[-\log q].$$

Definition 3.2. *Given distributions P and Q of a continuous random variable, the Kullback-Leibler divergence from Q to P is defined as*

$$D_{KL}(P\|Q) = \int_{-\infty}^{+\infty} p(x) \log \frac{p(x)}{q(x)} dx,$$

being p and q the densities of P and Q.

Remark 3.1. *Taking advantage of logarithm properties, it is immediate to evince the following relationship between cross entropy and KL divergence:*

$$D_{KL}(P\|Q) = H(P) - H(P, Q),$$

where $H(P) \equiv H(P, P)$ is the entropy of P.

Remark 3.2. *None of the two statistical distances just introduced is symmetric. This is reasonable in many typical contexts, where P represents the "true" (or theoretical) distribution and Q its approximation. In these cases the interest lies in finding the approximation Q that minimizes cross entropy or KL divergence (they are identical up to an additive constant) with respect to P, since it encodes how much information is lost when using the approximation. On the contrary, we would like to deal with a symmetric distance, because the estimations we are going to compare are exchangeable: in order to ensure this property we will simply sum the two asymmetric quantities, as it is proposed by Kullback and Leibler [1951] themselves:*

$$KL(P,Q) = KL(P\|Q) + KL(Q\|P).$$

Let us now derive the expressions for entropy, cross-entropy and KL divergence between random variables distributed according to Wishart and Inverse-Wishart distributions, which are the cases we are interested in, being Ω_k and Σ_k distributed likewise.

Proposition 3.4. *Let X_1 and X_2 be $p \times p$ random matrices, Wishart distributed, with parameters (δ_1, V_1) and (δ_2, V_2), respectively. Then cross entropy $H(X_1, X_2)$, entropy $H(X_1)$ and KL divergence $KL(X_1\|X_2)$ are given by:*

$$\begin{aligned} H(X_1, X_2) = {} & -\frac{\delta_2}{2}\log|V_2^{-1}V_1| + \frac{p+1}{2}\log|V_1| + \frac{\delta_1}{2}\operatorname{Tr}(V_2^{-1}V_1) \\ & + \log\Gamma_p\left(\frac{\delta_2}{2}\right) - \frac{\delta_2 - p - 1}{2}\Psi_p\left(\frac{\delta_1}{2}\right) + \frac{p(p+1)}{2}\log 2; \end{aligned} \tag{3.6}$$

$$\begin{aligned} H(X_1) = {} & \frac{p+1}{2}\log|V_1| + \log\Gamma_p\left(\frac{\delta_1}{2}\right) - \frac{\delta_1 - p - 1}{2}\Psi_p\left(\frac{\delta_1}{2}\right) \\ & + \frac{\delta_1 p}{2} + \frac{p(p+1)}{2}\log 2; \end{aligned} \tag{3.7}$$

$$\begin{aligned} KL(X_1\|X_2) = {} & -\frac{\delta_2}{2}\log|V_2^{-1}V_1| + \frac{\delta_1}{2}\operatorname{Tr}(V_2^{-1}V_1) + \log\frac{\Gamma_p\left(\frac{\delta_2}{2}\right)}{\Gamma_p\left(\frac{\delta_1}{2}\right)} \\ & + \frac{\delta_1 - \delta_2}{2}\Psi_p\left(\frac{\delta_1}{2}\right) - \frac{\delta_1 p}{2}. \end{aligned} \tag{3.8}$$

Ψ_p is is the multivariate digamma function, i.e. the derivative of the logarithm of multivariate gamma function Γ_p.

Proposition 3.5. *Let X_1 and X_2 be $p \times p$ random matrices, Inverse-Wishart distributed, with parameters (δ_1, V_1) and (δ_2, V_2), respectively. Then, the cross* *entropy $H(X_1, X_2)$, entropy $H(X_1)$ and KL divergence $KL(X_1\|X_2)$ are given by:*

$$\begin{aligned} H(X_1, X_2) = &-\frac{\delta_2}{2}\log|V_1^{-1}V_2| - \frac{p+1}{2}\log|V_1^{-1}| + \frac{\delta_1}{2}\operatorname{Tr}(V_1^{-1}V_2) \\ &+ \log\Gamma_p\left(\frac{\delta_2}{2}\right) - \frac{\delta_2+p+1}{2}\Psi_p\left(\frac{\delta_1}{2}\right) - \frac{p(p+1)}{2}\log 2; \end{aligned} \tag{3.9}$$

$$\begin{aligned} H(X_1) = &-\frac{p+1}{2}\log|V_1^{-1}| + \log\Gamma_p\left(\frac{\delta_1}{2}\right) - \frac{\delta_1+p+1}{2}\Psi_p\left(\frac{\delta_1}{2}\right) \\ &+ \frac{\delta_1 p}{2} - \frac{p(p+1)}{2}\log 2; \end{aligned} \tag{3.10}$$

$$\begin{aligned} KL(X_1\|X_2) = &-\frac{\delta_2}{2}\log|V_1^{-1}V_2| + \frac{\delta_1}{2}\operatorname{Tr}(V_1^{-1}V_2) + \log\frac{\Gamma_p\left(\frac{\delta_2}{2}\right)}{\Gamma_p\left(\frac{\delta_1}{2}\right)} \\ &+ \frac{\delta_1-\delta_2}{2}\Psi_p\left(\frac{\delta_1}{2}\right) - \frac{\delta_1 p}{2}. \end{aligned} \tag{3.11}$$

Remark 3.3. *KL divergence is invariant to parametrization. This consideration leads to the following equality*

$$KL(\Sigma_i\|\Sigma_j) = KL(\Omega_i\|\Omega_j) \qquad \forall i \neq j,$$

which can be easily checked by comparing 3.8 and 3.11.

With the closed-form expression of KL divergence and cross entropy between both covariance and precision matrices we have, at a low computational price, a first exploratory tool which gathers in a single positive number a notion of distance between each pair of groups. The a posteriori distances we computed could be also useful for some cluster algorithms, which might detect interesting sets of similar conditions, among those under study.
Obviously, there are also some drawbacks: first of all, as already mentioned, this is supposed to be only a very first, qualitative, information about group similarities, since we are trying to summarize differences and commonalities between high-dimensional matrices in a single number, which, as a result, cannot carry all the information we are interested in. Moreover, the distances we compute are meaningful only when put on a relative scale and compared between them: per se, they are not a good indicator of the effective similarity. To be clearer, we will get extremely high values for KL divergence even between almost equal matrices. This happens exactly for the reason we

were discussing earlier: even a slight perturbation in a very high-dimensional object has a huge impact on its "projection" on a one-dimensional space, as KL divergence and cross entropy are.

3.5 Structure estimation

Up to this point we dealt with the estimation of marginal and partial correlation matrices. As in the single-condition case, this is only an intermediate step, because users' interest lies in a more interpretable result, like a network (in the multi-condition case K networks) where only significant connections between variables are reported.
To find a proper and working way to extend the hypothesis testing to the multi-condition case has been very hard; we struggled particularly in dealing with double multiplicity (multiple edges within multiple conditions), since we had to think about which kind of error we could control and how to do it.
Before starting illustrating our results, let us briefly recap what we have done so far: by setting hyperparameter δ to the value maximizing the joint marginal likelihood, we obtained a joint estimation of marginal and partial correlation in each of the K groups, together with tail probabilities and Bayes factors related to the probability of inclusion in the network, in perfect analogy with the univariate case we presented in Chapter 1.5. For the time being, we stored either Bayes factors and tail probabilities, because both these quantities could be useful for the selection algorithm: indeed, we will see that some approaches we tried rely on Bayes factors and some others on tail probabilities.
Regarding the selection procedure, we now make a distinction between confirmatory and exploratory analysis, recalling a paper by Tukey [1980]. In a nutshell: as the name suggests, exploratory analysis aims at exploring data, in order, for instance, to identify their general structure, the key variables and the questions worth to be investigated. Goeman et al. [2011] outline three characteristics for exploratory research, which describe its open-minded nature: mild (some false positive are allowed among the selected hypotheses), flexible (no a-priori prescription of which precise hypotheses to select or not select) and post-hoc (choices inherent the procedure can be made after seeing the data). Vice versa, confirmatory analysis is about the rigorous evaluation of the evidence coming from the data by means of statistical tools such as significance, inference and confidence. Goeman et al. [2011] position multiple hypotheses testing at the border between the two approaches, especially when thousands of variables are involved: although somewhat structured,

such experiments take also into account criteria as convenience and completeness when selecting a collection of hypotheses to be tested. Indeed, the rejected hypotheses are usually not to be considered as final results to be reported, but rather as a pool of significant variables, on which it is worth to follow up with validation experiments.
Thanks to the analytical derivation of Bayes factors and tail probabilities, the method we developed provides the tools for proper confirmatory analysis. However, there are two considerations we need to take into consideration: first of all, even though tail probabilities and Bayes factors are available, it is not straightforward to understand how to manipulate them, in order to account for double multiplicity. The second consideration is about the type of information we want to return in output: confirmatory analysis is performed to answer a specific question, say, the network structure made of edges present only in condition 1. Conversely, we think that it is often more useful to return the network estimated in each condition: a rigorous formulation of the corresponding hypothesis test is difficult to obtain, therefore we will try a more exploratory approach. In this section we will discuss Union-Intersection tests using Bayes factors, pertaining confirmatory analysis; after that, we will move to the broader and more exploratory goal of associating each edge to the conditions in which it is present, if any.

3.5.1 Confirmatory analysis with Bayes factors

The confirmatory procedure for structure estimation we developed takes inspiration from the work of Van Deun et al. [2009] and it is based on the following two observations:

1. The Bayes factor is the ratio between the marginal likelihoods under the alternative hypothesis and under the null: the higher, the more likely is the alternative. If we take the reciprocal (or change the sign if we are working with its logartihm) we will have a measure of how likely the null is with respect to the alternative.

2. Exploiting the a priori independence across groups, we can easily formulate global hypothesis tests for, say, the detection of edges present in all groups: to be in common, an edge must be present in all groups, i.e. the null hypothesis must be rejected in each group. Bayes factors quantify the likeliness of the alternative versus the null in each group: thanks to independence, we can multiply (or sum if we consider the logarithms) them to obtain the Bayes factor for the global test.

We call these global tests Union-Intersection (UI) tests, since the global alternative is the intersection of many individual hypotheses, and the null its complement, i.e. the union of the complement individual hypotheses. Denoting $\rho_{ij}^{(k)}$ the partial correlation between variables i and j in condition k, we formulate the null hypothesis of conditional independence by $H_{0,ij}^{(k)} : \rho_{ij} = 0$ against the alternative $H_{1,ij}^{(k)} : \rho_{ij} \neq 0$. The test we propose on each edge, in order to identify the common network structure is the following:

$$H_{0,ij} : \bigcup_{k=1}^{K} H_{0,ij}^{(k)} \quad \text{against} \quad H_{1,ij} : \bigcap_{k=1}^{K} H_{1,ij}^{(k)}$$

Recalling what discussed in consideration 1, we can actually test whatever global hypothesis we may be interested in by simply considering the reciprocal of the Bayes factors corresponding to the conditions where we want to test the edge to be absent.
We illustrate this concept with an example, where we have 3 conditions and we want to investigate the network structure specific to group 2, that is composed by those edges which are present exclusively in group 2. The corresponding UI test is

$$H_{0,ij} = H_{1,ij} \cup H_{0,ij} \cup H_{1,ij} \quad \text{against} \quad H_{1,ij} : H_{0,ij} \cap H_{1,ij} \cap H_{0,ij}.$$

The global Bayes factor expressing the evidence in favour of the alternative is then obtained by multiplying the Bayes factor in group 2 by the reciprocals of Bayes factors in groups 1 and 3.
The potentiality of this method relies on the easy formulation of a global test for any differential or common network structure we are interested to investigate; however, several issues arose when we implemented and applied it to simulated data:

- Only one test at a time. UI-tests require the a priori statement of the network structure of interest, either the common one or that specific to one or more conditions. In any case, it is not possible to estimate the network structure relative to each condition, therefore the purpose of this method is limited.

- p_0 estimation. Selection methods based on Bayes factors, blfdr and BFDR, need the tuning of parameter p_0 which, in short, converts the Bayes factors into probabilities. To them, standard multiplicity adjustment can be applied in order to find the edges in the network significantly different from 0. The theory states that p_0 is the proportion of tests for which H_0 is not rejected (true null). In many real cases, this information is unknown; furthermore, even trying on simulated data to set p_0 to the exact value, the selection was not optimal.

Aiming at solving the problem of p_0, we used HC threshold on tail probabilities to estimate the proportion of true null. HC stands for "Higher Criticism", term that was coined by Tukey [1976]. We refer to a more recent work by Klaus and Strimmer [2012], who propose to apply an empirical HC threshold based on p-values (tail probabilities in our case) for signal identification in high-dimensional settings.
Setting $N := p(p-1)/2$ equal to the total number of edges, the HC approach to signal identification proceeds as follows:

1. Arrange the p-values in increasing order $p_{(1)}, ..., p_{(N)}$. The empirical distribution function of the p-values is given by:
$$\hat{F}(x) = i/N \quad \text{for} \quad p_{(i)} \leq x < p_{(i+1)},$$
with $x \in [0; 1]$, $p_{(0)}$ and $p_{(N+1)}$.
2. Compute the empirical HC objective function:
$$\hat{HC}(x) = \frac{|\hat{F}(x) - x|}{\sqrt{\hat{F}(x)(1 - \hat{F}(x))/N}}.$$
3. Obtain the HC statistics
$$\hat{HC}^* = \max_i \hat{HC}(p_{(i)}) = \hat{HC}(x^{HC}).$$
4. Take x^{HC} as the HC decision threshold for signal identification: all edges whose $p_{(i)} < x^{HC}$ are considered "significant".

We then estimate p_0 as the proportion of p-values greater than the HC threshold:
$$p_0 = card\left(\{p_{(i)} : p_{(i)} \geq x^{HC}\}\right)/N.$$
With this approach, we assess in the simulations a good approximation of the real value, i.e. the true proportion of edges not present in the network.

3.5.2 Exploratory analysis with tail probabilities

Our next goal was to answer to wider questions, instead of focusing on a single network structure of interest. As just discussed, UI test is not suited for this more general purpose. Neither are Bayes factors: indeed, we could multiply them as long as at the alternative we had the intersection of elemental hypotheses. When trying to address more general problems, this property of Bayes factors is not useful anymore; in addition to that, their numeric value is not directly interpretable, since the scale is not clearly defined. Considering these issues, we decided to use tail probabilities for group allocation, which are well-suited for standard multiplicity adjustments.

Intersection-Union test and hierarchical testing

Our first idea to detect a broader set of edges was to switch from UI to IU test, standing for intersection union. This global test detects all the edges that are present in at least one condition:

$$H_{0,ij} : \bigcap_{k=1}^{K} H_{0,ij}^{(k)} \quad \text{against} \quad H_{1,ij} : \bigcup_{k=1}^{K} H_{1,ij}^{(k)}.$$

At this stage, we have the opposite problem, that is the question is too general, since all we know about the edges for which the null was rejected is that they are present in at least one group, but we don't know neither how many nor which ones.
From a computational point of view, in order to perform this test, we select for each edge its minimum tail probability across all conditions. This means that for each edge we are testing whether it is present or not in the condition where it is most likely to appear. After this procedure we end up with $p(p-1)/2$, which we adjust to account for multiplicity. If we terminate here, we expect to detect lots of edges, but still the information we have on them is very weak: the idea is to add a further step to the procedure in which we allocate the edges we found to be significant to the conditions they belong to.
The considerable problem we had to face when trying to think about this additional step pertains the error control. During selection we controlled a specific error type, at a specific level: it is not straightforward to come up with a solution for the group allocation step such that the same error control is preserved, or even any form of error control.
We decided to use the approach proposed by Goeman et al. [2011], dealing with multiple testing for exploratory research. Their final goal is slightly different from ours, since they aim at pinpointing one or more subset of variables which prove to be significant, so that it is reasonable to follow up with a confirmatory analysis on them. We aim instead at a more rigorous group allocation of those edges we already pinpointed to be the significant ones.
Nevertheless, the method they propose has the peculiarity that family-wise error rate (FWER) is always controlled at the same level, no matter which subset out of the entire collection is selected: this is exactly the characteristic we were looking for.
Let us present the methodology in more detail. As the name, hierarchical testing, suggests, it consists of a hierarchical procedure: at the top, the global intersection hypothesis test $H_1 \cap H_2 \cap ... \cap H_K$ is performed. In our framework, it corresponds to test whether a specific edge is present in all conditions. If it

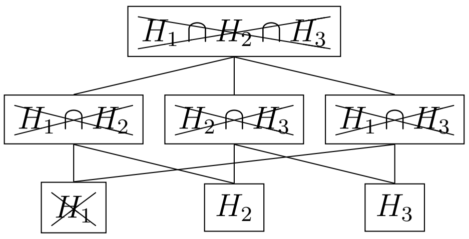


Figure 3.2: Example of hierarchical testing, taken from the paper by Goeman et al. [2011]. Rejected hypotheses are marked with a cross.

is rejected, we go down of one level and we perform the $K-1$ tests, with all possible combinations of $K-1$ variables: $H_2 \cap ... \cap H_K$, $H_1 \cap H_3 \cap ... \cap H_K$, ..., $H_1 \cap ... \cap H_{K-1}$. Where the null hypothesis is not rejected, we do not explore further; conversely, we keep testing the underlying leaves whenever we reject the null hypothesis. An example with three variables is shown in Figure 3.2: the first two levels are all rejected, on the third only H_1 is rejected. This result is somehow counter-intuitive: since $H_2 \cap H_3$ is rejected, we expect at least one of H_2 and H_3 to be rejected as well. For this reason, the procedure is said to be non-consonant: the authors claim that in the framework of exploratory research, which aims at being milder than family-wise error-based one, non-consonant testing procedures need not to be avoided.
In our case, though, we would prefer to have this property: we will see in a while how we will guarantee it.
We now observe that, in its standard form, this hierarchical testing procedure requires about $2^n - 1$ tests to be performed, being n the number of variables under study: with more than 20-30 variables, it becomes computationally intractable. Our problem is in a much higher-dimensional setting, being n the number of edges whose null hypothesis was rejected during IU test .
For such cases, the authors propose two shortcuts, which can be applied under specific assumptions: one of them is based on Fisher's combination method (Fisher [1925]), which can be adopted when the null hypotheses are independent, as ours are. Given the p-values $p_1, ..., p_n$ of the tests of the elemental hypotheses $H_1, ..., H_N$, Fisher's combination method rejects the intersection hypothesis I if

$$-2 \sum_{i \in I} \log(p_i) \geq g_{\#I},$$

being g_r the $(1-\alpha)$-quantile of a χ^2-distribution with $2r$ degrees of freedom. Together with the paper, the authors provided the R package `cherry` (Goeman et al. [2015]), available on CRAN repository, where hierarchical testing, in its standard form and with shortcuts, is implemented.

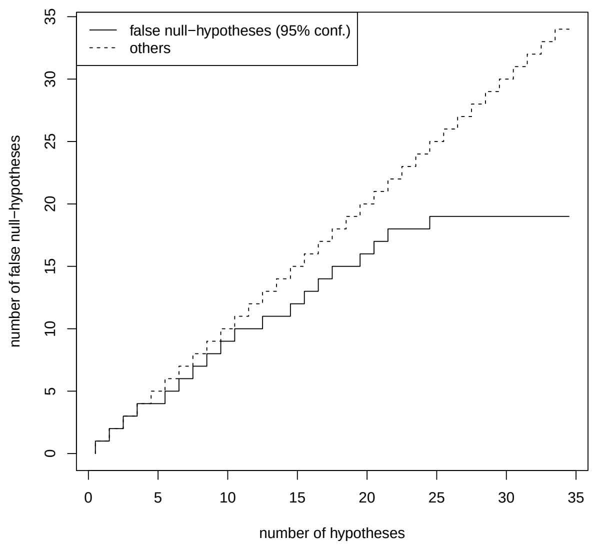


Figure 3.3: Output of function `curveFisher` in package `cherry`, performing hierarchical testing with Fisher's combination method.

In Figure 3.3 the graphical output of `curveFisher` is shown: the input data is a vector of 34 p-values provided in the package itself. Given a value on the x-axis (corresponding to how many hypotheses, increasingly ordered according to their p-value, we are considering), the corresponding y-value on the solid line represents the number of correct rejections we are making, with 95% confidence. With respect to Figure 3.3, we can reject the 4 most significant hypotheses, being confident at 95% of not making any mistake. If we want to include also the fifth, we do not have evidence enough to reject all five: with 95% confidence, we could state that we are making 4 correct rejections out of 5. Moving to the right, it is interesting to notice that there is no advantage in including any of the last 9 variables, since we have no benefit in terms of number of true rejected.
This tool is very useful when dealing with thousands of variables and it is necessary, due to budget constraints for instance, to pinpoint a subset on which it is reasonable to focus further analyses.
As already pointed out, our main goal is to allocate the edges selected by IU test to the conditions they belong to: this is achieved by considering each edge present in the conditions located before the point where the solid line stops overlapping with the dotted one. That's the point beyond which the number of true rejected hypotheses is strictly less than the number of

included variables. If we went beyond it, the only statement we could do would be something similar to: "With 95% confidence, this edge is present in 4 out of the 5 following conditions". Such a result would be difficult to communicate to the final user.
To conclude, we discuss some drawbacks that led us to look for a better-suited method. Recalling that we focus only on the edges rejected by IU test, i.e. present in at least one condition, we expect, accordingly, each edge to be allocated in at least one condition by `curveFisher` function: this does not always happen, due to the non-consonance of the procedure.
Moreover, the error type that is controlled is family-wise error rate, which is very conservative: in our framework, where the signal is often weak, we struggle to have fairly good power.

Hierarchical BH

In order to deal with the weakness of signal and to solve the lack of power, it might be useful to apply a less stringent error control than family-wise error rate, when performing multiple testing, such as False Discovery Rate (FDR), introduced by Benjamini and Hochberg [1995].
We will here present some results from a more recent work by Benjamini and Bogomolov [2014]: they adapt FDR error control to the case of multiple families of hypotheses, which is exactly our framework. Indeed, we have in total $p(p-1)/2 \times K$ tail probabilities (we will refer to them as p-values in this section), corresponding to $p(p-1)/2$ edges in K networks. In our model we consider each edge independent from the others, therefore we can think of it as a family, whose components refer to the presence or absence of that edge in each condition $k = 1, ..., K$. Summing up, we have $p(p-1)/2$ families/edges and K hypotheses inside each of them.
When dealing with multiple families, a common approach is to select promising families first (in our case, this was achieved by Intersection-Union test) and then apply a multiple testing procedure in each selected family separately. Unfortunately, this strategy is proved not to guarantee any level of confidence about the filtration of errors within the selected families.
In general, it could be interesting to assure some level of confidence for discoveries within the selected families: a natural request could be the control of the expected value of some measure of error $\mathcal{C}$ in each selected family i: $\mathbb{E}\left[\mathcal{C}_i | i \text{ is selected}\right]$. This, however, is often difficult to achieve; for this reason, a more modest goal is presented: the control of the expected average value of $\mathcal{C}$ over the selected families.
Formally, let P_i be the set of p-values in family i, $i = 1, ..., N$, being $N = p(p-1)/2$ the number of families/edges. P is the ensemble of these sets:

$P = \{P_i\}_{i=1}^{N}$. Let $\mathcal{S}$ be the selection procedure, which, given in input the p-values P, returns the indexes of the selected families. Let $|\mathcal{S}(P)|$ be the number of selected families.
The error criterion under study is:

$$\mathbb{E}\left[\mathcal{C}_{\mathcal{S}}\right] = \mathbb{E}\left[\sum_{i=1}^{N} \mathcal{C}_i / N\right].$$

The authors claim that, in many cases, the control over the expected average measure of errors is also more appropriate for the interpretation of the results than the global control of an error rate over the selected family, because it gives some confidence in the discoveries within the selected families.
The main result of the paper by Benjamini and Bogomolov [2014] we are presenting is that, in order to guarantee the control of $\mathbb{E}\left[\mathcal{C}_{\mathcal{S}}\right]$, $\mathbb{E}\left[\mathcal{C}\right]$ should be controlled in each selected family i at a more stringent level: the nominal level q must be multiplied by the proportion of selected families among all the families. The adjustment procedure (*procedure 1*) required on the testing levels within the selected families is given by the following steps:

1. apply the selection rule $\mathcal{S}$ to the ensemble of sets P. Let R be the number of selected families: $R = |\mathcal{S}(P)|$.
2. apply the $\mathbb{E}\left[\mathcal{C}\right]$ controlling procedure in each selected family separately at level Rq/N.

Theorem 1 (Theorem 1 of Benjamini and Bogomolov [2014])**.** *For any error rate $\mathbb{E}\left[\mathcal{C}\right]$ such that $\mathcal{C}$ takes values in a countable set, suppose that we have a testing procedure that can control $\mathbb{E}\left[\mathcal{C}\right]$ at any desired level α under the dependence structure of the p-values within a family. If the p-values in each family are independent of the p-values in any other family then for any simple selection rule $\mathcal{S}(P)$ the selection-adjusted procedure guarantees $\mathbb{E}\left[\mathcal{C}_{\mathcal{S}}\right] \leq q$.*

Proof. See Benjamini and Bogomolov [2014] □

An additional useful result presented in the paper states that, if the considered error rate is FDR, then the control of $\mathbb{E}\left[\mathcal{C}\right]$ by the selection-adjusted procedure yields control of FDR at the family level. Thus, the control is guaranteed both for the erroneous discoveries within each family and at the first stage where families are selected. Building upon the work of Benjamini and Bogomolov [2014] presented so far, Peterson et al. [2016] propose a procedure which controls vFDR (FDR for the discovery of families) and sFDR (average FDR on the selected families) at levels q_1 and q_2, respectively.

Testing is carried out on the basis of the p-values corresponding to each individual hypothesis H_{it}, with $i = 1, ..., N$ indexing the family and $t = 1, ..., K$ each hypothesis within it. The p-values for the intersection hypothesis $H_{i\bullet} = \bigcap_{t=1}^{K} H_{it}$ are defined as the Simes's p-values (Simes [1986]):

$$p_{i\bullet} = \min_t \frac{K p_{i(t)}}{t},$$

where $p_{i(t)}$ is the t-th element of the increasingly ordered vector $\{p_{it}, t = 1, ..., K\}$.
The hierarchical procedure (we will refer to it as *hierarchical BH*), is defined by the following steps:

0. Use Simes's method to obtain the p-values $p_{i\bullet}$ for the intersection hypotheses $H_{i\bullet}$.

1. apply the Benjamini-Hochberg (BH) method (Benjamini and Hochberg [1995]) to the collection of p-values $p_{i\bullet}$ with an FDR target level q_1. Let $\mathcal{S}(P)$ be the set of rejected hypotheses $H_{i\bullet}$.

2. Test the individual hypotheses H_{it} only in families $\mathcal{F}_i \in \mathcal{S}(P)$. Within such families, apply BH with target level $q_{2,adj}$ properly adjusted to account for the selection bias introduced in stage 1:

$$q_{2,adj} = q_2 \times \frac{|\mathcal{S}(P)|}{N}.$$

Remark 3.4. *Setting $q_1 \leq q_2$ consonance is guaranteed, i.e. if the global null corresponding to an edge is rejected, then it will be allocated to at least one condition.*

We implemented the hierarchical BH procedure in the R function `hierBH`, reported in Appendix B.4. As we will see in Section 5.1, where we will test on simulated data the three methods presented for structure estimation, hierarchical BH proves to have good performances. Moreover, it was originally developed to deal with a problem very similar to ours, i.e. multiple families (edges in our case) of hypotheses (about the allocation in the K conditions).

Chapter 4

Computational development

4.1 R package beamDiff

In this project, theoretical and computational development proceeded side by side. This was necessary because, at each step, we needed to check if what seemed correct and sound under a theoretical point of view also worked in practice. As in the unidimensional case, our method should prove to be much faster than other Bayesian ones and less parameter dependent than those using a Lasso approach. In our multi-condition framework, good memory management is fundamental: during the estimation phase, for instance, we need to store $K \times p(p-1)/2$ for the K correlation matrices and as many for the partial ones. It is therefore worth making an effort to find a light way to store the data.
After setting hyperparameter δ by maximizing the joint marginal likelihood, the estimation procedure boils down at independently computing the posterior distribution in each condition: this step is identical to the single group case. Thus, it now becomes clearer why the first goal of the thesis project was the optimization of R package `beam`: most of the code of `beam` function will be reused for the estimation of each condition.
We called the R package `beamDiff`, to highlight the strong relationship with `beam` and suggest its application to differential networks. The package is not complete: currently, it only consists of the estimation function `beamDiff` and of some methods acting on its output.
All the procedures discussed in Section 3.5 have been implemented, for the time being, as standalone functions. Similarly, of the three approaches to hyperparameter elicitation presented in Section 3.3, we included in the package only the simplest ($M = (\delta - p - 1)I_p$), since the theory about the derivation of tail probabilities for the other two has yet to be completed.

We report here the header of `beamDiff` function, `beamDiff(..., type, return.only, approx.tail, verbose)`: the body of the function is reported in Appendix B.1. The first argument is an ellipsis, to allow more flexibility in the input data format: the method accepts both K separated datasets or a single list, where each element correspond to a dataset. With parameter `type` the user can specify if interested in both marginal and partial estimation, or only in one of the two; `return.only` is used to set which quantities to compute (correlations, Bayes factors and/or tail probabilities). `approx.tail` is a binary flag, regulating whether or not the approximation of `pbeta` function, explained in Section 2.2 must be put in place, in order to speed-up the function. If `verbose` is set to `true`, some information about the execution are displayed, while the code is running.

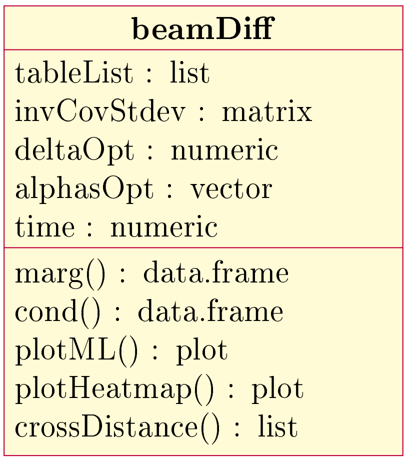
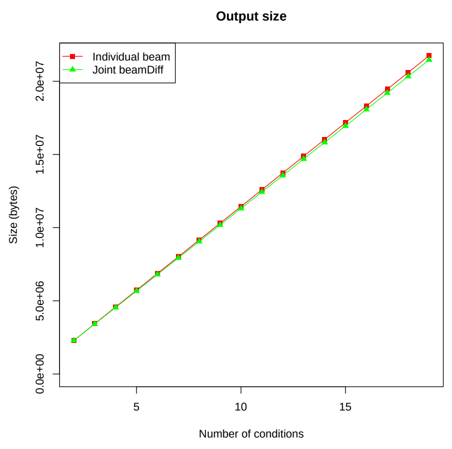
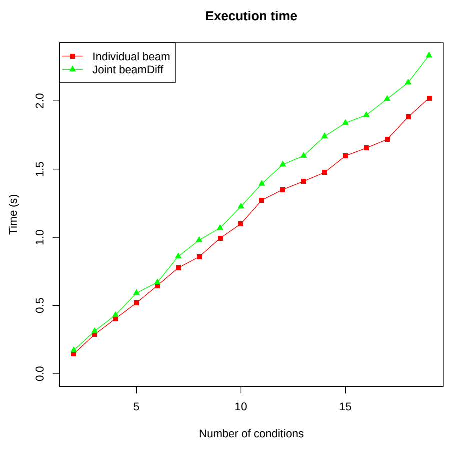


Figure 4.1: Simplified UML class diagram for the S4 object provided in `beamDiff` package. Core slots and methods are displayed.

As for `beam` function, `beamDiff` outputs a S4 object of class `beamDiff-class`: its more relevant quantities and methods are displayed in Figure 4.1.
`tableList` is a list of K elements: each of them is a matrix with $p(p-1)/2$ rows; the number of columns varies from 2 to 6, depending on `type` and `return.only` parameters. `invCovStdev` is a $p \times K$ matrix: its columns contain the diagonal of the estimated posterior inverse covariance matrix, which is necessary (and expensive) to compute, but not to store, because the correlation matrix is returned. However, we need to recover the covariance matrix and its inverse to compute KL-divergence (see Section 3.4), therefore the diagonal is needed.
We implemented for `beamDiff` the same methods that were acting on `beam`, adjusted accordingly. In addition to them, we also implemented the method `crossDistance`: it outputs a list with three $K \times K$ matrices containing,



(a) Memory allocation



(b) Execution time

Figure 4.2: Comparison between `beam` and `beamDiff` estimations in terms of memory allocation of the respective outputs (a) and of their running time (b). The results are averaged on 5 simulations.

respectively, KL divergence and marginal and conditional cross-entropy. As already discussed in Section 3.4, these quantities are very useful to have a first insight about which conditions have similar network structures and which are expected to differ more; these distances can also be used in several clustering algorithms. In input to the function, users can set the flag `symmetric`: if `true`, a symmetrised version of these distances, obtained by summing each matrix with its transpose, is returned.
Let us now see how `beamDiff` compares to `beam` in terms of memory and time: we run both of them on a set of data made of 19 conditions. We will present it more details in Section 5.2. We vary the number of conditions to consider, from 2 to 19: for each of these settings we run `beamDiff` to get the joint inference estimations and `beam` on each single condition. We then compare the execution time and memory allocation of `beamDiff` with execution times and object sizes of `beam`, summed all together.
The results are displayed in Figure 4.2. From the memory allocation point of view, `beamDiff` performs slightly better than `beam`, and the improvement increases as we increase the number of conditions to analyse: this might be enabled by the decision to change the structure to store the results, from data.frame to matrix. As already observed in Section 2.2, matrix representation is lighter than data.frame when data are type-homogeneous, as in our case: the advantage is really small (indeed for `beam` is negligible), but, when summed up over multiple conditions, it becomes slightly evident

(when analysing all 19 conditions, `beamDiff` memory allocation is 1% less than `beam`). As expected, `beamDiff` takes longer than the individual `beam` execution times summed up together: the overhead might be given by the repeated calls to `Map` function, through which we simultaneously apply the same function to all conditions. However our main concern is memory, and there `beamDiff` proves to perform better; moreover, even if a bit slower (15% when considering all 19 conditions), we are still very fast, being able to analyse all 19 groups with 217 features in less than 3 seconds.

4.2 Data simulation

In order to assess the performance of our method, we had to run it on simulated data, where the underlying graph structure was known. In this way we could compute some useful indicators:

- distance between true and estimated precision and covariance matrices. It highlights how precise we are in the estimation, i.e. how close we get to our target;
- ROC curves, where we use as predictors the absolute values of the estimated partial correlation and as labels a binary vector indicating the presence (1) or absence (0) of each edge in the graph. This tool helps us understanding how well the original ordering is preserved in the estimation: the better, the more accurate the recovery of the graph structure will be;
- contingency table (containing the number of true positives, true negatives, false negatives and false positives): this indicator, contrary to the previous two, is computed after selection step has been performed. We are therefore measuring how good is our thresholding algorithm: we might achieve a very good ROC curve, but then "cut" it at a non-optimal point (being, for instance, too conservative or too liberal).

Our focus has been on the performance in recovering the conditional dependence structure, being it more informative and interpretable than the marginal one. For this reason, we aimed at simulating the precision matrix Ω, which by definition has to be symmetric positive definite; moreover, representing a network, we need it to be sparse: the non-zero elements will correspond to the edges present in the graph.
In addition to these structural characteristics, we also want to control the magnitude of the non-zero entries and the condition number: the former because if they are too small (say under 0.001) it will be very hard, nay

impossible, for the method to pinpoint them among the random noise generated during the sampling phase; the latter because Ω needs to be inverted in order to obtain Σ from which we will sample from the Gaussian distribution. When inverting a ill-conditioned matrix, extra noise is added, therefore the results will not reflect the real performances of the method.
Lastly, we remark that we want to simulate a framework with multiple conditions, whose underlying graph structure must be partly in common, since we want to simulate a situation where we have information to share across groups.

4.2.1 Review of existing methods

First of all, we revised how in similar papers present in the literature simulations have been carried on.
In the paper by Peterson et al. [2015], 4 conditions are simulated: the first matrix Ω_1 has a band structure, with non-zero entries that guarantee symmetric positive definiteness; the other three are obtained by removing some edges from the first one and adding some new. In order to assess positive definiteness in the last three groups each off-diagonal element is divided by the sum of the off-diagonal elements in its row, and then the matrix is averaged with its transpose.
This way of simulating has two issues: first of all, the number of nodes included is 20 (the method is Bayesian so, as previously discussed, it cannot scale) and, if we increase it, the magnitude of non-zero entries in Ω_2, Ω_3 and Ω_4 will become smaller and smaller, if we want to keep the same sparsity level, due to the regularization necessary to make them positive definite. The other issue is in the simulation itself, where the magnitude of the entries in Ω_2, Ω_3 and Ω_4 is smaller than in Ω_1: this responds a merely structural constraint and we believe it does not reflect a realistic situation.
Danaher et al. [2014] simulate 3 conditions with 500 features each. Ω_1 has a block structure, composed by ten equally sized unconnected subgraphs; each subgraph follows a power law degree distribution, which is thought to mimic the biological networks. Ω_2 is set equal to Ω_1, except for one of the ten blocks that is removed; the same happens for Ω_3, but in this case the removed blocks are two. Also in this case, positive definiteness is not automatically guaranteed: similarly to the previous example, all the elements are divided by the sum of the elements present in their row, then the matrix thus obtained is averaged with its transpose, to make it symmetric. This regularization procedure makes most of the entries really small (a simulation example is reported in Table 4.1): the authors themselves considered, when assessing the performance of their method, the edges with magnitude under

Summary abs. partial correlation							
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Spars.	Cond.
9.09e-14	1.85e-05	2.57e-04	0.0124	0.0028	0.3421	9.81	26.75
9.10e-14	1.76e-05	2.44e-04	0.0124	0.0027	0.3404	8.84	23.21
9.10e-14	1.64e-05	2.44e-04	0.0125	0.0029	0.3404	7.86	23.05

Table 4.1: Distribution of non-zero entries (absolute value), sparsity and condition number of the 3 precision matrices simulated according to the method presented in Danaher et al. [2014].

a certain threshold as not actually present in the graph. We do believe that it is more fair to avoid such a shortcut.

4.2.2 Our method

Aware of the difficulties others met before us when simulating realistic high-dimensional graphical models, we tried to keep the structures, band and blocks, of the examples illustrated in the previous section and only improve the problematic aspects.
Concerning the block structure, the big advantage is that we can simulate each block independently, so that the dimensionality of the problem reduces. For instance, we can obtain a 200×200 matrix by simulating 5 blocks with 40 features each. As we increase the number of blocks we simplify the simulation, but simultaneously we also increase the sparsity, since the maximum density we can get is given by $1/B$, being B the number of blocks. Like Danaher et al. [2014], we reproduce the multi-condition framework by replicating the same matrix in all groups, and in turn remove one or more blocks.
Regarding the band structure, the heterogeneity across conditions is obtained by permuting a subset of rows of columns: in this way, there will be a core of edges shared across all groups, plus some group-specific edges.
For both structures we implemented a function (see Appendix B.3) with several parameters that can be set, in order to obtain a simulated matrix with the desired properties.

```
simBlockMatrix <- function(sparsity, nBlocks, val_lb = 0, val
   _ub = 1, p, max_iter = 1000, maxCond = 500, minCorThres =
   0, minMedThres = 0, shrink = 0)
```

Listing 4.1: header of function `simBlockMatrix`

Function `simBlockMatrix`, whose header is reported in Listing 4.1, simulates one precision matrix with *nBlocks* independent blocks along the main

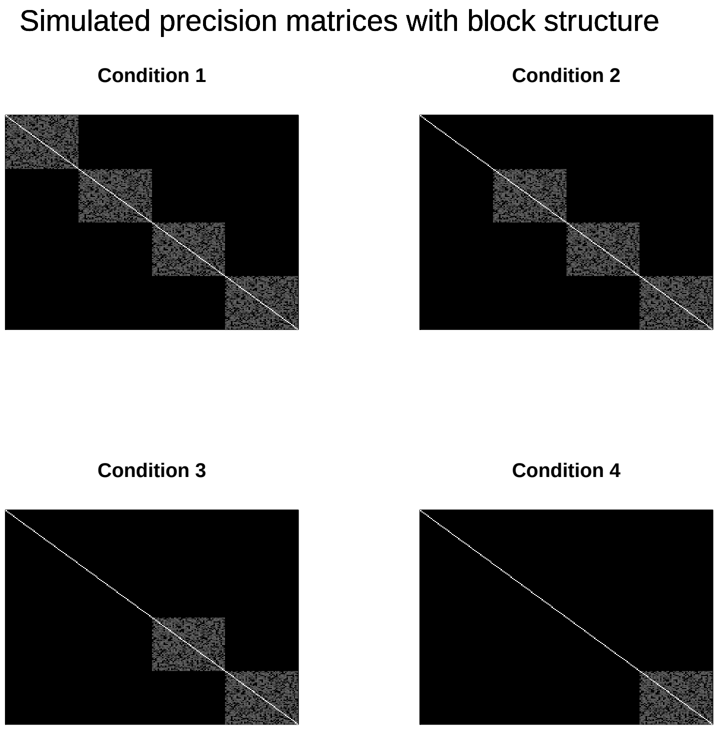


Figure 4.3: Simulation example with $p = 200$ and block structure. The absolute value of non-zero entries is displayed. The grey color scale goes from 0-black to 1-white.

diagonal. With parameter *sparsity* we control the overall sparsity, recalling that its maximum value is $1/nBlocks$. For convenience, we decided to simulate only one squared block of dimension $p/nBlocks$ and then replicate it the required number of times. For that block, first the position of the non-zero entries is chosen randomly, then their values are uniformly sampled from $[-val\_ub, -val\_lb] \cup [val\_lb, val\_ub]$. If the resulting matrix thus obtained is already positive definite and its condition number is under *maxCond*, the algorithm stops and returns the matrix. If, instead, positive definiteness is not assessed, the quantity *shrink* is added on the main diagonal and the matrix is then scaled accordingly to reset the values on the diagonal to 1. This regularization process makes the minimum eigenvalue increase, so that, if the shrinking is enough, it will become positive; as countereffect, the non-zero elements will be shrunk towards 0: through parameters *minCorThres* and *minMedThres* we can control their minimum magnitude and/or the minimum value of their median. This whole procedure is iterated until a matrix fulfilling all constraints is found, up to a maximum number of iterations specified by parameter *max_iter*: in this second case an error message is displayed, with information about the best result achieved so far. Based on that, the function can be called again with a new setting for the parameters. In Figure 4.3 an example of simulation with 4 conditions is shown. The first matrix is generated by the following call to the function: `simBlockMatrix(sparsity = .2, nBlocks = 4, p = 200, shrink = 0.05, val_lb = .1, val_ub = 1, max_iter = 1000)`.

```
simBandShuffle <- function(p, width, shuffle.from.list,
   shuffle.to.list, K, lb = 0, ub = 1)
```

Listing 4.2: header of function `simBandShuffle`

In Listing 4.2 we display the header of function `simBandShuffle`: parameter p sets the dimension of the matrix, *width* specifies the bandwidth, *lb* and *ub* lower and upper bounds for the uniform distribution from which the non-zero entries are sampled. Unlike the block simulation, in this case we directly simulate K matrices, corresponding to as many conditions: we will start simulating the matrix for the first condition, then it will be copied in the other groups with some rows and columns, specified by *shuffle.from.list* and *shuffle.to.list*, randomly permuted. In Figure 4.4 we display the output matrices produced by the following code: `simBandShuffle(p=200, width=20, shuffle.from.list=list(1,1,1), shuffle.to.list=list(30,60,90), K=4, lb=.3, ub=1)`.

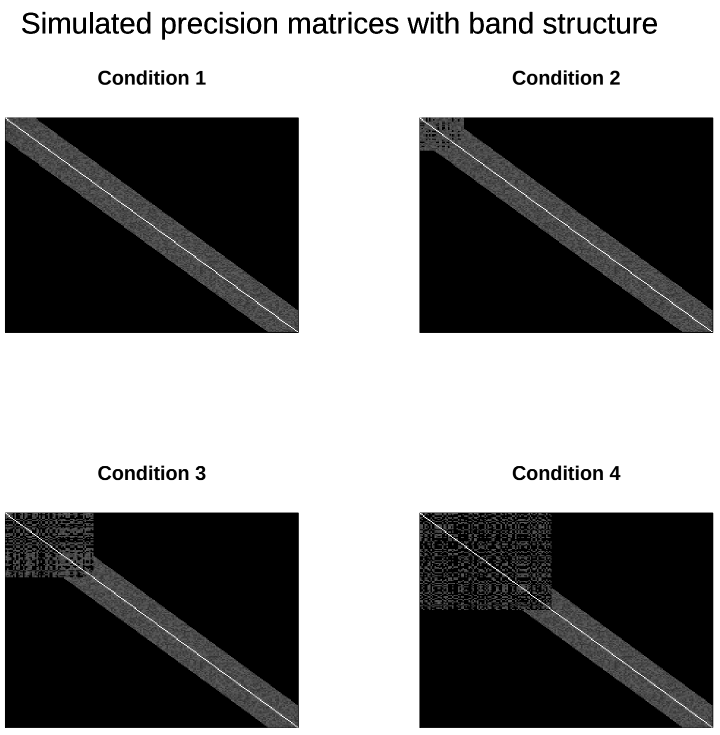


Figure 4.4: Simulation example with $p = 200$ and band structure of width 20. The absolute value of non-zero entries is displayed. The grey colour scale goes from 0-black to 1-white. The number of permuted rows and columns in condition 2,3 and 4 is 30,60 and 90 respectively.

Chapter 5

Results on simulated and real data

In the last two chapters we discussed some methods for estimating marginal and partial correlation (Section 3.3), some for structure selection (Section 3.5) and data simulation (Section 4.2), discussing issues and possible solutions. In this chapter we will see how the algorithms we presented perform on simulated data; after that, we will use our method to jointly infer multiple network structures on a real dataset.

5.1 Internal comparison

Let us start by introducing the simulated data we are going to use. We set the number of features p to 200 in all simulations: though not extremely high, we think it is enough to appreciate the behaviour of our method. We tried to be as exhaustive as possible by let several parameters under our control in the simulation change. We summarize here the various settings we are going to analyse in the next plots:

- Network structure. We simulate both band and block structure for our precision matrices. This to ensure our method's performance is independent of the underlying structure it has to infer.

- Information shared across groups. We simulate either the ideal case of same underlying structure in all groups and the case of a core shared structure plus some group-specific edges. The former, although not realistic, is the situation in which there is maximum information to share across groups and it is interesting to see how the different methods behave in this context.

We did not analyse the other extreme case of no similarity between conditions: we already expect a poor performance in this case, because our method has been tailored to encourage commonalities across groups, under the assumption of some information to be shared. If there is none, the method will wrongly bring the conditions close to each other.

- Number of groups. We will simulate the setting with 2,5 and 10 conditions.
- Number of observations. We will consider the case in which all conditions have the same number of observations n_k and the case in which they differ. As already pointed out in Section 3.3, this imbalance is a factor of heterogeneity: in this setting, we expect the methods performing joint inference (i.e. allowing the conditions to borrow information from the others) to perform better than `beam`, i.e. than the independent inference in each condition.

Practically, our simulations proceeded in the following way: with functions `simBandShuffle` and `simBlockMatrix` we simulate 4 lists of 10 sparse precision matrices, 2 with band and 2 with block structure. For each structure, one list contains the same matrix replicated ten times; the other represents the case of partially shared information.
In Figures 5.1 and 5.2 we display the precision matrices we are going to use throughout all the simulations. For each of the four scenarios, we select the first $K = 2, 5, 10$ groups and we create the correspondent simulated dataset by sampling n_k observations from the multivariate Gaussian distribution. This step is repeated for 10 times, in order to have more stable results.

5.1.1 Correlation and precision matrices estimation

We start comparing the three strategies for hyperparameters elicitation discussed in Section 3.3 for the joint inference, among them and with the individual estimation of each group independently.
For the individual estimation we use function `beam`, provided by the homonym R package; the case $M = (\delta - p - 1)I_p$ is currently implemented in R package `beamDiff`. The function performing the joint optimization in M and δ using a EM algorithm is named `beamDiff_boost`. Lastly, in function `beamDiff_new` we estimate M by running `beam` on all data merged together.
We remind that these simulations pertain only the estimation of correlation and precision matrices, because the theory on how to derive the tail probabilities when M is not diagonal has yet to be developed.

Method	cond(H)	dist_marg	dist_part	dist_part_nz	AUC
beam	-	9.645	4.799	3.789	0.872
beamDiff	-	9.643	4.788	3.790	0.873
beamDiff_boost	555.55	8.041	9.548	**2.046**	0.926
beamDiff_new	45.2	**5.779**	**4.021**	2.074	**0.989**
beam	-	9.642	4.784	3.789	0.877
beamDiff	-	9.638	4.787	3.786	0.875
beamDiff_boost	561.99	8.001	9.564	**2.064**	0.925
beamDiff_new	46.07	**5.777**	**4.020**	2.074	**0.988**
beam	-	9.615	4.793	3.786	0.862
beamDiff	-	9.616	4.790	3.789	0.862
beamDiff_boost	587.99	8.027	9.584	**2.054**	0.929
beamDiff_new	45.74	**5.777**	**4.021**	2.074	**0.989**
beam	-	9.600	4.780	3.789	0.869
beamDiff	-	9.588	4.784	3.782	0.870
beamDiff_boost	505.03	7.974	9.581	**2.049**	0.922
beamDiff_new	44.26	**5.774**	**4.020**	2.074	**0.988**
beam	-	9.621	4.778	3.774	0.877
beamDiff	-	9.626	4.775	3.778	0.877
beamDiff_boost	580.2	8.020	9.553	**2.019**	0.926
beamDiff_new	44.73	**5.776**	**4.020**	2.074	**0.989**

Table 5.1: Some performance indicators for the estimation of marginal and partial correlation matrices. Simulated setting: same band structure in all 5 conditions, $p = 200$, $n_k = 100$ (same as figure 5.3). From left to right: method used for estimation, condition number of $H = M + S_k$, L^2-distance between true and estimated correlation matrix, same for precision matrix and for non-zero entries in the precision matrix, area under the curve (AUC).

Method	cond(H)	dist_marg	dist_part	dist_part_nz	AUC
beam	-	10.927	4.993	4.331	0.793
beamDiff	-	10.969	4.957	4.379	0.795
beamDiff_boost	486.06	7.838	9.264	**2.016**	0.928
beamDiff_new	38.69	**5.834**	**4.012**	2.117	**0.985**
beam	-	10.352	4.916	4.096	0.836
beamDiff	-	10.392	4.886	4.132	0.837
beamDiff_boost	371.70	7.965	9.423	**2.056**	0.923
beamDiff_new	40.36	**5.833**	**4.017**	2.114	**0.986**
beam	-	9.642	4.795	3.790	0.868
beamDiff	-	9.652	4.787	3.798	0.869
beamDiff_boost	509.43	7.912	9.698	2.125	0.914
beamDiff_new	40.86	**5.826**	**4.024**	**2.108**	**0.986**
beam	-	9.260	4.729	3.627	0.890
beamDiff	-	9.243	4.740	3.614	0.890
beamDiff_boost	456.96	7.843	9.817	2.144	0.913
beamDiff_new	37.29	**5.825**	**4.029**	**2.103**	**0.986**
beam	-	8.718	4.635	3.391	0.901
beamDiff	-	8.687	4.661	3.363	0.901
beamDiff_boost	496.59	7.711	9.827	2.140	0.910
beamDiff_new	40.62	**5.821**	**4.034**	**2.097**	**0.986**

Table 5.2: Some performance indicators for the estimation of marginal and partial correlation matrices. Simulated setting: same band structure in all 5 conditions, $p = 200$, $n_k = 50$, 70, 100, 120, 150 (same as figure 5.4). From left to right: method used for estimation, condition number of $H = M + S_k$, L^2-distance between true and estimated correlation matrix, same for precision matrix and for non-zero entries in the precision matrix, area under the curve (AUC).

Method	cond(H)	dist_marg	dist_part	dist_part_nz	AUC
beam	-	9.406	4.547	3.722	0.841
beamDiff	-	9.391	4.553	3.713	0.841
beamDiff_boost	1263.04	8.077	9.504	**2.217**	0.874
beamDiff_new	39.32	**6.246**	**4.042**	2.489	**0.955**
beam	-	9.370	4.553	3.720	0.840
beamDiff	-	9.365	4.555	3.717	0.840
beamDiff_boost	1228.12	8.101	9.487	**2.203**	0.877
beamDiff_new	41.36	**6.255**	**4.043**	2.491	**0.954**
beam	-	9.532	4.558	3.707	0.852
beamDiff	-	9.543	4.551	3.714	0.852
beamDiff_boost	1029.29	8.199	9.517	**2.276**	0.871
beamDiff_new	40.80	**6.464**	**4.075**	2.536	**0.947**
beam	-	9.401	4.556	3.724	0.845
beamDiff	-	9.392	4.560	3.720	0.845
beamDiff_boost	1205.06	8.166	9.634	**2.411**	0.860
beamDiff_new	42.31	**6.492**	**4.174**	2.653	**0.940**
beam	-	9.421	4.553	3.706	0.843
beamDiff	-	9.432	4.547	3.712	0.844
beamDiff_boost	1043.54	8.274	9.631	**2.451**	0.856
beamDiff_new	40.06	**6.719**	**4.251**	2.730	**0.926**

Table 5.3: Some performance indicators for the estimation of marginal and partial correlation matrices. Simulated setting: partially shared band structure in all 5 conditions, $p = 200$, $n_k = 100$ (same as figure 5.5). From left to right: method used for estimation, condition number of $H = M + S_k$, L^2-distance between true and estimated correlation matrix, same for precision matrix and for non-zero entries in the precision matrix, area under the curve (AUC).

Method	cond(H)	dist_marg	dist_part	dist_part_nz	AUC
beam	-	10.455	4.721	4.148	0.770
beamDiff	-	10.534	4.679	4.198	0.771
beamDiff_boost	802.00	8.060	9.269	**2.335**	0.870
beamDiff_new	35.57	**6.436**	**4.122**	2.631	**0.934**
beam	-	9.962	4.646	3.937	0.811
beamDiff	-	10.030	4.611	3.979	0.812
beamDiff_boost	879.18	8.178	9.428	**2.353**	0.865
beamDiff_new	37.07	**6.430**	**4.125**	2.622	**0.933**
beam	-	9.462	4.572	3.707	0.836
beamDiff	-	9.488	4.557	3.723	0.836
beamDiff_boost	1044.50	8.200	9.657	**2.390**	0.868
beamDiff_new	37.01	**6.580**	**4.134**	2.612	**0.937**
beam	-	9.078	4.506	3.593	0.844
beamDiff	-	9.038	4.527	3.570	0.843
beamDiff_boost	1147.41	8.053	9.827	**2.446**	0.857
beamDiff_new	37.00	**6.504**	**4.181**	2.656	**0.934**
beam	-	8.601	4.429	3.376	0.890
beamDiff	-	8.556	4.455	3.348	0.890
beamDiff_boost	1118.28	7.900	9.930	**2.425**	0.859
beamDiff_new	38.51	**6.600**	**4.197**	2.639	**0.935**

Table 5.4: Some performance indicators for the estimation of marginal and partial correlation matrices. Simulated setting: partially shared band structure in all 5 conditions, $p = 200$, $n_k = 50, 70, 100, 120, 150$ (same as figure 5.6). From left to right: method used for estimation, condition number of $H = M + S_k$, L^2-distance between true and estimated correlation matrix, same for precision matrix and for non-zero entries in the precision matrix, area under the curve (AUC).

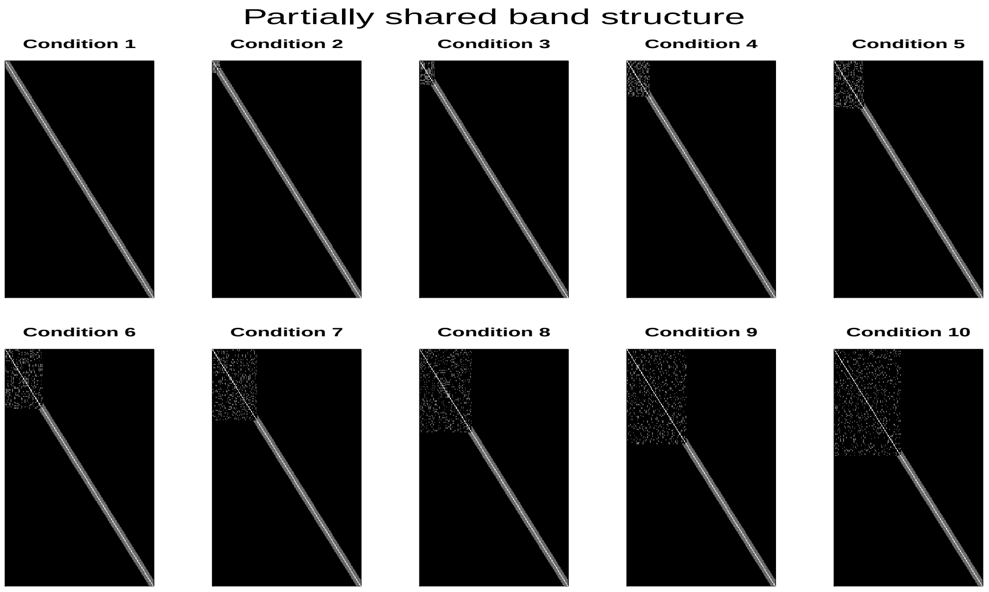


Figure 5.1: Simulated precision matrices with band structure, obtained with function `simBandShuffle`. The magnitude of the non-zero entries is between 0.12 and 0.22. The sparsity is at 5% and its condition number is 94.30. The number of shuffled rows is, respectively: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90.

In the next plots and tables we compare Receiver Operating Characteristic (ROC) curves, condition number of $H_k = M + S_k$ and L^2-distance between true and estimated matrices. For the ROC curves we used the absolute values of estimated partial correlations as predictors and the binary indicators of presence or absence of each edge as labels.
We report here all the scenarios (totally/partially shared network structure, same/different number of observations) and discuss the results for the band structure case with $K = 5$ conditions. Afterwards, we will present one case for $K = 2$, one for $K = 10$ and one relative to the block structure.
In Figure 5.3 and Table 5.1 we report, respectively, the ROC curves and some indicators about the method's performance relative to the case with same underlying structure and same number of observations. From the plot we notice that the ROC curves relative to `beam` and `beamDiff` (red and blue) overlap almost perfectly. This is in accordance with what we already discussed in section 3.3: borrowing information only by means of hyperparameter δ does not lead to any significant improvement with respect to the separate inference on the single conditions. Table 5.1 confirms it: indeed, the first two lines of each group, are almost identical. We appreciate some differences between the two only in cases with strong heterogeneity across conditions, like in the

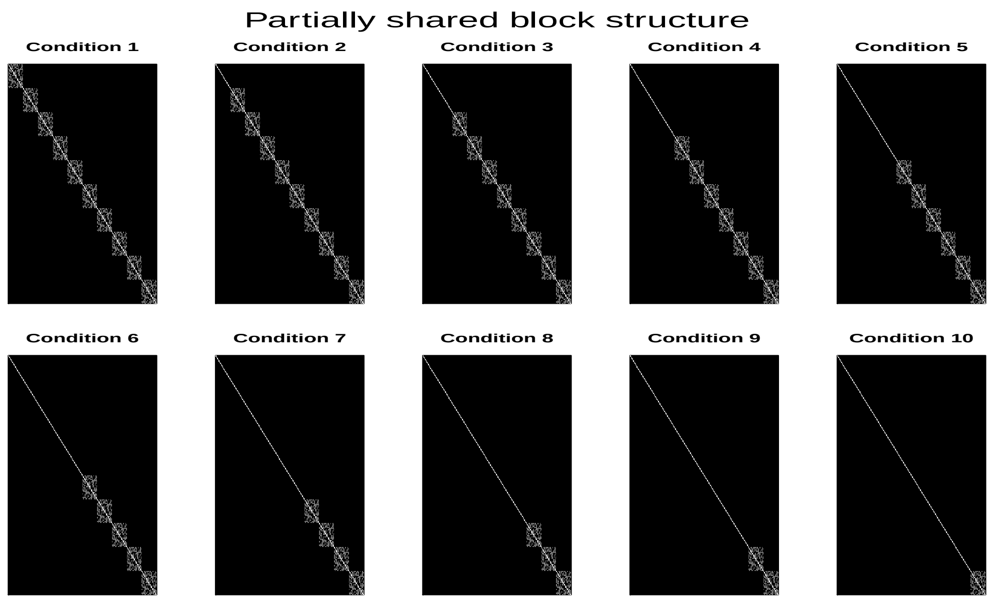


Figure 5.2: Simulated precision matrices with block structure, obtained with function `simBlockMatrix`. The magnitude of the non-zero entries is between 0.14 and 0.22. The full matrix (with all 10 blocks) is 5% sparse and its condition number is 171.54. The number of blocks in the matrices is, respectively: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.

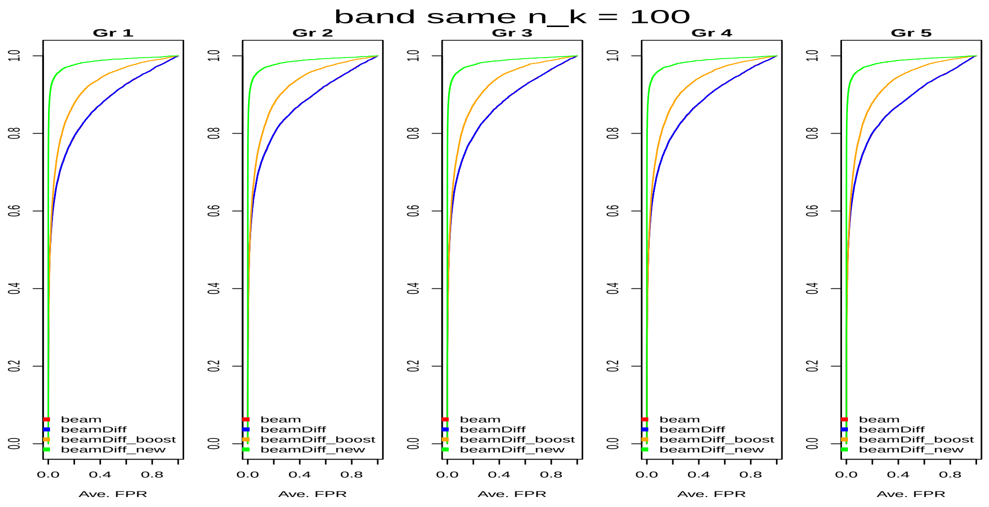


Figure 5.3: ROC curves for the correlation matrix estimation. 5 conditions, same band structure, $p = 200$ and $n_k = 100$ in all conditions.

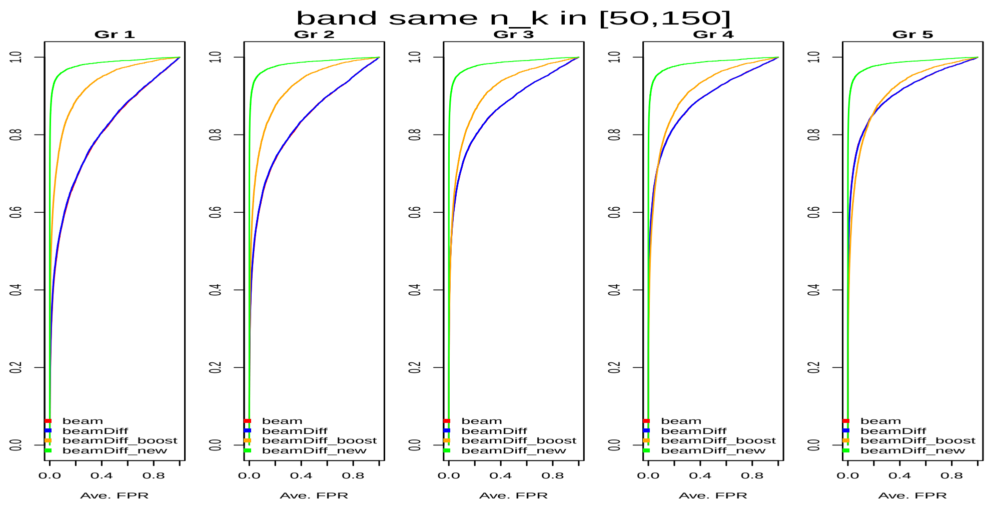


Figure 5.4: ROC curves for the correlation matrix estimation. 5 conditions, same band structure, $p = 200$, $n_k = 50$, 70, 100, 120, 150.

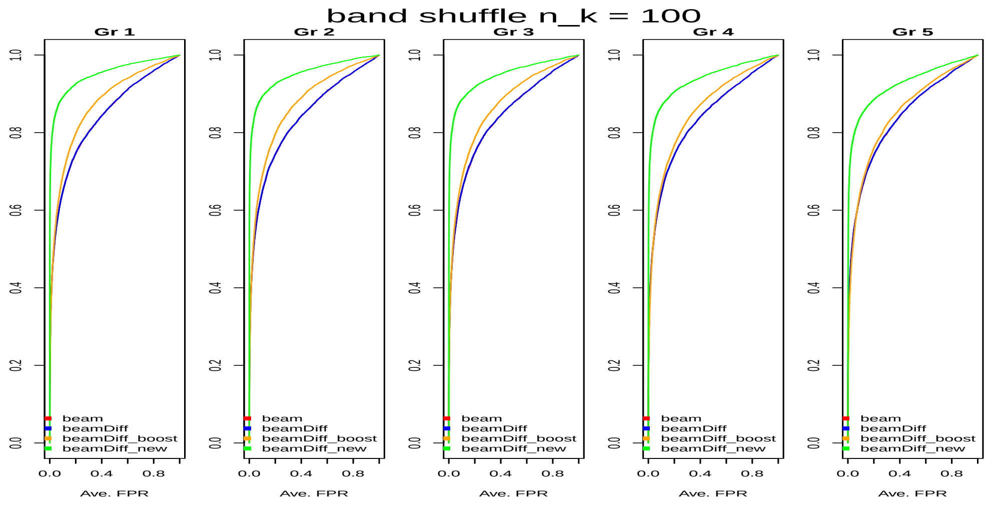


Figure 5.5: ROC curves for the correlation matrix estimation. 5 conditions, shuffled band structure, $p = 200$, $n_k = 100$ in all conditions.

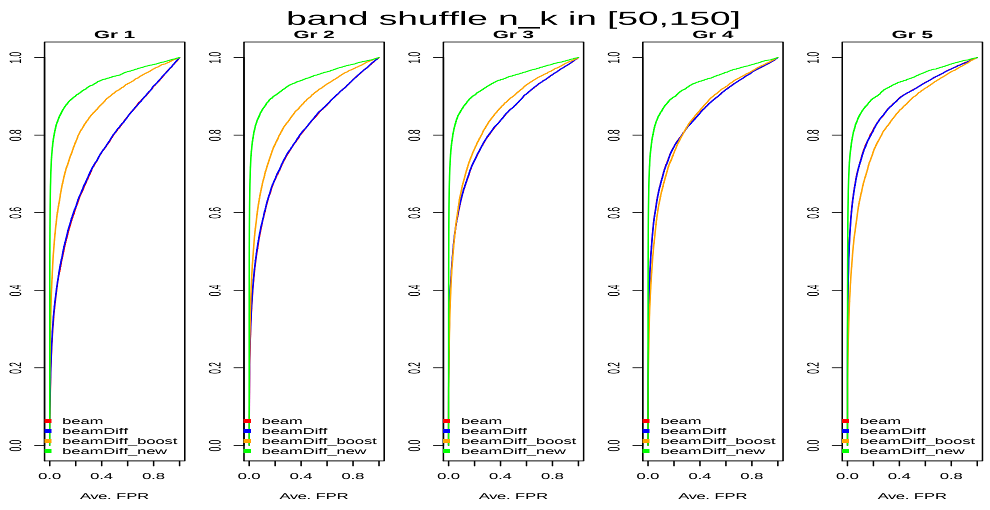


Figure 5.6: ROC curves for the correlation matrix estimation. 5 conditions, shuffled band structure, $p = 200$, $n_k = 50$, 70, 100, 120, 150.

case presented in Figure 5.9, which we will discuss in more details later.
When number of observations and structure are identical in all conditions (Figure 5.3), we observe, as expected, that the ROC curves are pretty much the same in all conditions; by varying the number of observations (Figure 5.4), we notice that `beam` and `beamDiff` have poor performances in the conditions with small sample size (on the left) and improve as it increases. On the contrary, `beamDiff_boost` and `beamDiff_new`, the latter in particular, manage to borrow a good amount of information through hyperparameter M: in this way, they perform pretty well also on the conditions with small n_k. If, instead, we vary the underlying structure, keeping constant the number of observations, we observe (Figure 5.5) that all methods struggle more than before: the performance gets worse as we increase the heterogeneity. `beamDiff_new` is particularly affected as we can see in the rightmost plot: this is due to the fact that we use as target M the matrix representing the common underlying structure, obtained by merging data from all conditions together. The more heterogeneous the data, the more the target will not be optimal. We notice, however, that its performance is still better than the others, at least in this case.
Lastly, we vary both structure and number of observations (Figure 5.6): again, we observe that the two methods using the identity matrix as target, worsen when the sample size decreases, while `beamDiff_boost` and `beamDiff_new` are less affected.
Besides ROC curves, we also computed some numeric indicators, reported

in Tables 5.1 (same structure, same n_k), 5.2 (same structure, different n_k), 5.3 (different structure, same n_k) and 5.4 (different structure, different n_k). For methods `beamDiff_boost` and `beamDiff_new` we report the condition number of $H_k = M + S_k$, which is the matrix that needs to be inverted to compute the partial correlation estimates. As expected, the condition number relative to `beamDiff_boost` is way bigger than that of `beamDiff_new` and in general quite high, like in Table 5.3.
The consequences of bad conditioning can be observed in the next two columns of the table, where the L^2-distance between true and estimated correlation and precision matrices are reported. As regards the correlation matrix, `beamDiff_new` performs better than `beamDiff_boost`, which in turn outperforms `beam` and `beamDiff`. When considering the precision matrix, `beamDiff_new` still performs (slightly) better, while `beamDiff_boost` has a much bigger error. This is caused by the numerical noise introduced when inverting bad-conditioned H_k. As further proof of it, we report in the next column the L^2-error restricted to the non-zero entries of the true precision matrices: we notice in this case `beamDiff_boost` has good performances again. At selection phase, this will likely translate into a good recall (many true positives) but insufficient precision (too many false positives).
In the last column the area under the curve (AUC) is reported: it refers to the ROC curves displayed in the corresponding plots.

Let us now discuss the impact of K, i.e. the number of conditions on the estimations: for this purpose we present a case with 2 conditions and one with 10. In Figure 5.7 we display the ROC curves relative to the precision matrix estimation in the setting with 2 conditions, different network structure and same number of observations: while `beamDiff_new` performs well, `beamDiff_boost` does as bad as random allocation. The explanation for this behaviour is again in the condition number (Table 5.5), which is in the order of 10^8. Both the estimations of marginal and partial correlation are completely disguised by the numerical noise due to ill-conditioning. When, instead, we consider the setting with 10 conditions (Figure 5.8), we observe that the performances of `beamDiff_new` and `beamDiff_boost` are similar: the more the conditions, the more the estimation of hyperparameter M provided by the EM algorithm is regularized. It also interesting to notice how the performances of both `beamDiff_new` and `beamDiff_boost` deteriorate as we increase the heterogeneity: for instance, in condition 10 approximately 50% of the edges are group-specific (see Figure 5.1), therefore there is no benefit, actually it might even be detrimental, in using a data-driven prior matrix, which shrinks towards the common structure, rather than the uninformative identity.
We give a graphical representation of the effect of M on the estimations in

the block structure case, where it is easier to be seen. The ROC curves relative to the setting with 10 conditions, partially shared network structure and same number of observations are displayed in Figure 5.9: in condition 10, where only one block is present (see Figure 5.2 for the true structure), the two methods using a data-driven prior perform much better, while in condition 1 (all 10 blocks present) the four methods perform quite similarly. In Figure 5.10 we display the estimations of condition 1 (5.10a) and condition 10 (5.10b). First of all, we can glimpse the noise affecting zero entries in `beamDiff_boost`'s estimation (third column) of the precision matrices. In condition 1, the best method for partial correlation estimation is `beamDiff`, indeed we can distinguish all 10 blocks, while in `beamDiff_new` estimation the last blocks mix up with the noise on zero entries. Conversely, in condition 10 the only block actually present is better identified by `beamDiff_new`, where we can also glimpse some other blocks: this is the effect of shrinking towards the structure inferred by pooling together all data, which will be a sort of average, containing therefore approximately 5 blocks. In light of all these simulations, we can draw some conclusions:

- `beamDiff` often performs as good as `beam`. When the underlying network structure is expected to be similar across the conditions, it proved beneficial to borrow information also through hyper-parameter M (methods `beamDiff_boost` and `beamDiff_new`). However, if conditions are quite heterogeneous, it might be convenient to use a non-informative prior, to avoid the shrinking towards a common structure that does not really exist.

- In case of imbalance in the number of observations, `beamDiff_boost` and `beamDiff_new` perform better than `beamDiff` in the conditions with small sample size, since they are able to borrow more information from the others.

- `beamDiff_boost` provides noisy estimates, unless we are dealing with many conditions and reasonably high sample sizes. Before applying this algorithm, it is recommended to compute the condition number of $H_k = M + S_k$, for each condition k, and check if it is reasonably low.

- In general, `beamDiff_new` proved to outperform (or at least perform as well as) all the other methods. Therefore, we think that it is worth extending the derivation of tail probabilities also to the case of non-diagonal prior matrix. In this way, we could use `beamDiff_new` at estimation phase and get better results also in the following step when we retrieve the network structures.

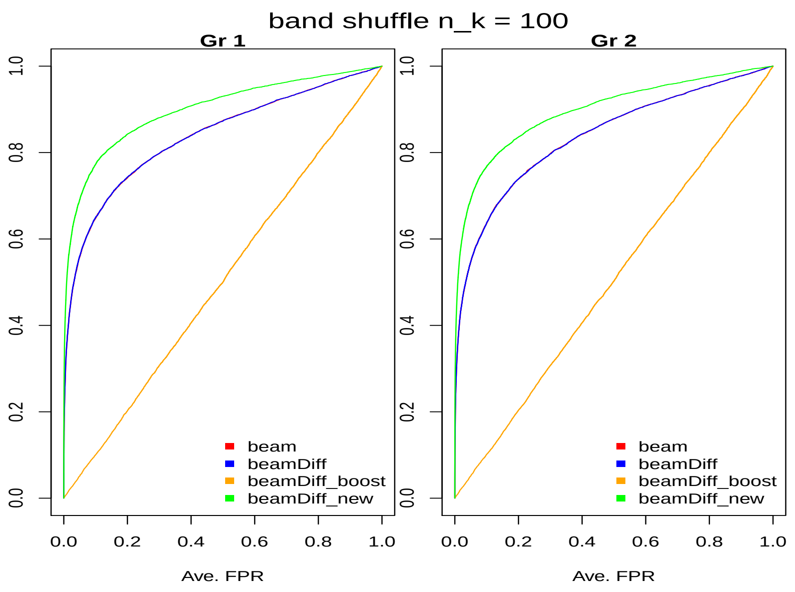


Figure 5.7: ROC curves for the correlation matrix estimation. 2 conditions, shuffled band structure, $n_k = 100$ in all conditions.

Method	cond(H)	dist_marg	dist_part	dist_part_nz	AUC
beam	-	9.369	4.557	3.710	0.840
beamDiff	-	9.374	**4.553**	3.713	0.841
beamDiff_boost	2.98E+08	10.024	97.087	21.366	0.503
beamDiff_new	33.81	**7.918**	4.600	**2.976**	**0.903**
beam	-	9.379	**4.556**	3.723	0.844
beamDiff	-	9.371	4.559	3.719	0.843
beamDiff_boost	3.11E+08	10.010	97.249	21.415	0.505
beamDiff_new	31.65	**7.912**	4.605	**2.981**	**0.901**

Table 5.5: Some performance indicators for the estimation of marginal and partial correlation matrices. Simulated setting: partially shared band structure in both conditions, $p = 200$, $n_k = 100$ (same as figure 5.7). From left to right: method used for estimation, condition number of $H = M + S_k$, L^2-distance between true and estimated correlation matrix, same for precision matrix and for non-zero entries in the precision matrix, area under the curve (AUC).

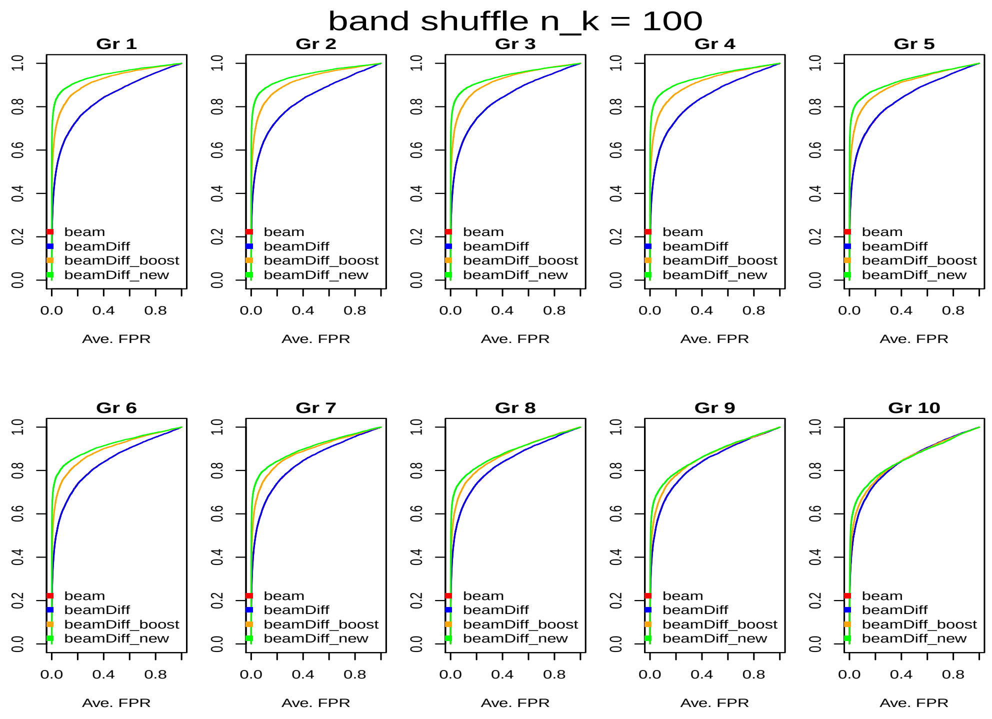


Figure 5.8: ROC curves for the correlation matrix estimation. 10 conditions, shuffled band structure, $p = 200$ and $n_k = 100$ in all conditions.

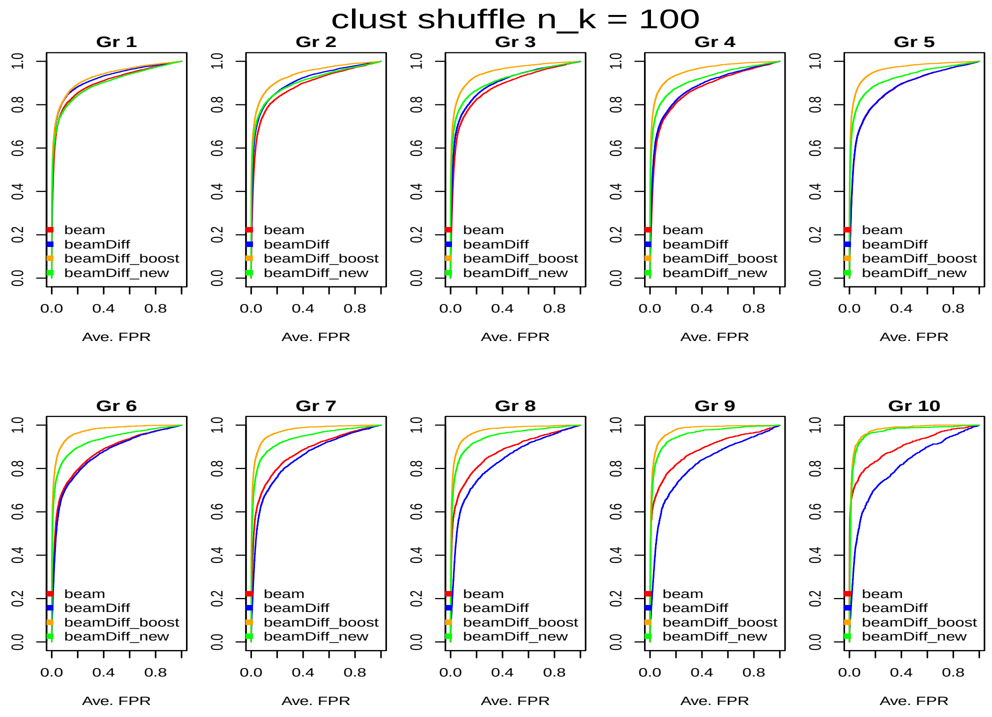
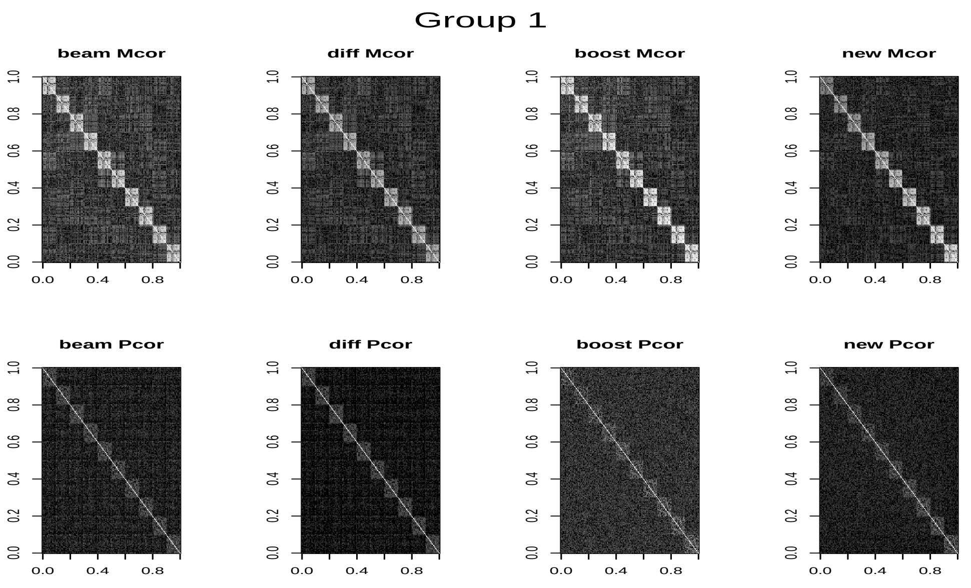
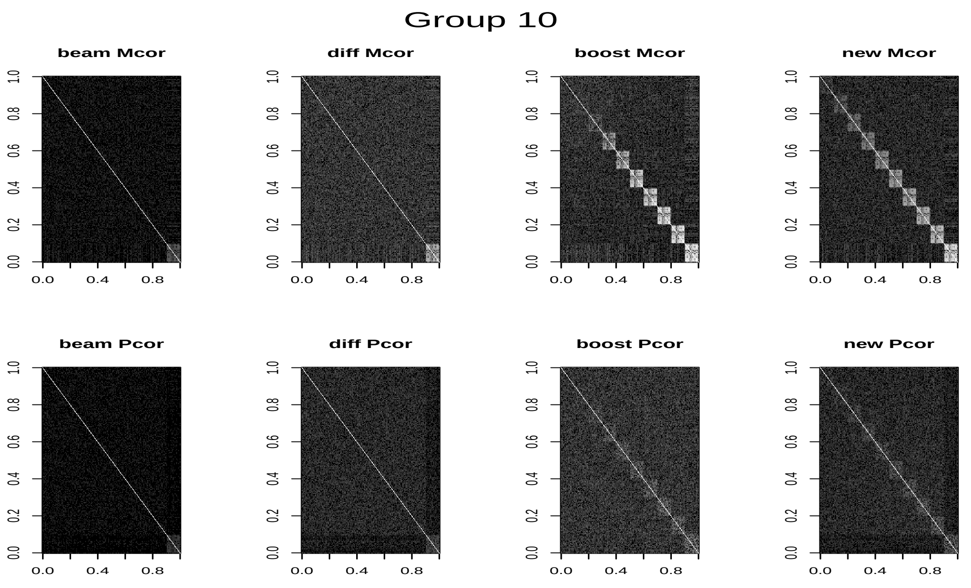


Figure 5.9: ROC curves for the correlation matrix estimation. 10 conditions, partially shared block structure, $p = 200$ and $n_k = 100$ in all conditions.



(a) Condition 1



(b) Condition 10

Figure 5.10: Plot of marginal (upper row) and partial (lower row) correlation matrices estimated by methods `beam`, `beamDiff`, `beamDiff_boost` and `beamDiff_new` (from left to right) for condition 1 (a) and 10 (b) in the following setting: 10 conditions, partially shared block structure, $p = 200$ and $n_k = 100$ in all conditions.

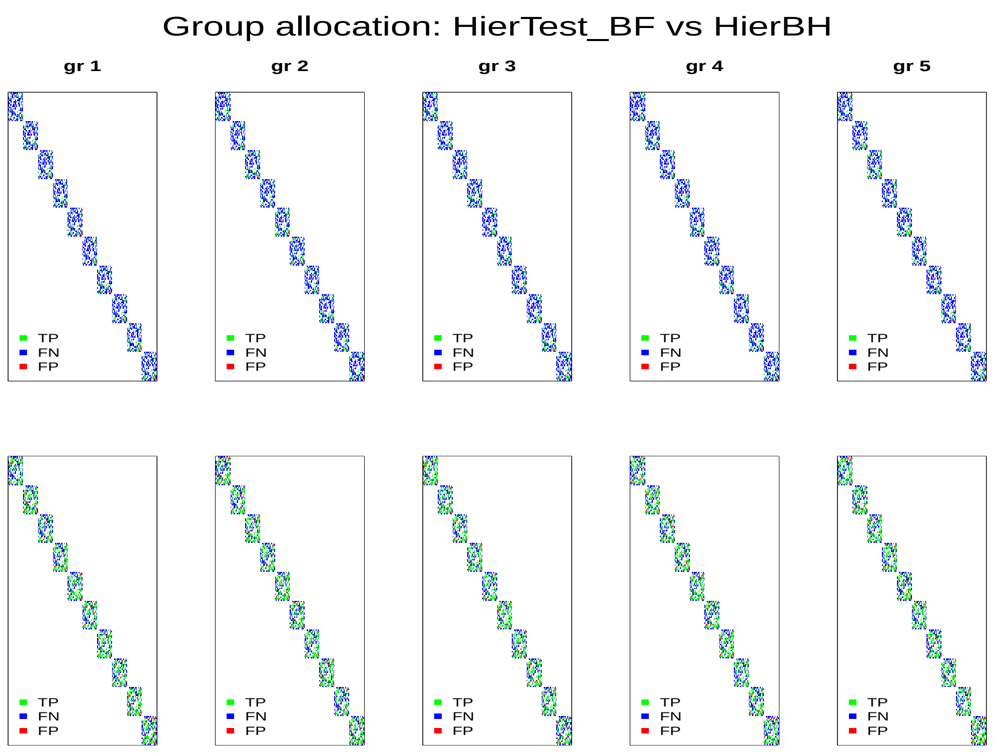


Figure 5.11: Network inference performed by methods `hierTest_BF` (upper row) and `hierBH` (lower row). 5 conditions, same block structure, $p = 200$ and $n_k = 100$ in all conditions. True Positives (green), False Negatives (blue) and False Positives (red) are displayed.

5.1.2 Network structure inference

Before presenting the simulations results, we recall the methods for retrieving the network structure, introduced in section 3.5, we are going to test.
In the context of confirmatory analysis, we proposed the Union-Intersection (UI) test, which is based on the product of the Bayes factors. UI test can be used to test whatever hypothesis about a specific network structure, e.g. that containing edges present only in a given condition or that shared across all conditions. In the simulations, we focus on this last case: we will compare it with the other two methods, where we will consider as common an edge that has been allocated to all conditions.
Switching to a more exploratory setting, we discussed hierarchical testing by Goeman et al. [2011] and hierarchical BH by Peterson et al. [2016], which we implemented in functions `hierTest_BF` and `hierBH` (see Appendix B.4), respectively.

Method	TP	FN	FP	TN	TPR (%)	FPR (%)
hierTest_BF	199	751	**11**	**18939**	20.95	**0.0580**
hierBH	**507**	**443**	76	18874	**53.37**	0.4011
hierTest_BF	211	739	**10**	**18940**	22.21	**0.0528**
hierBH	**511**	**439**	73	18877	**53.79**	0.3852
hierTest_BF	198	752	**10**	**18940**	20.84	**0.0528**
hierBH	**519**	**431**	75	18875	**54.63**	0.3958
hierTest_BF	196	754	**10**	**18940**	20.63	**0.0528**
hierBH	**509**	**441**	74	18876	**53.58**	0.3905
hierTest_BF	203	747	**10**	**18940**	21.37	**0.0528**
hierBH	**512**	**438**	72	18878	**53.89**	0.3799
hierTest_BF	71	879	**2**	**18948**	7.47	**0.0106**
hierBH	340	610	12	18938	35.79	0.0633
UI test	**568**	**382**	52	18898	**59.79**	0.2744

Table 5.6: Performance indicators for methods `hierTest_BF`, `hierBH` and `UI_test` (only for common edge detection). 5 conditions, same block structure, $p = 200$ and $n_k = 100$ in all conditions. A single horizontal line separates the conditions; under the double line, same performance indicators for common edge detection.

In its original context, hierarchical testing was thought to be applied on a vector of p-values: we have a matrix, instead, made of $N = p(p-1)/2$ rows, corresponding to the edges, and K columns, one for each condition. As discussed when presenting this method, we first apply Intersection-Union (IU) test to identify the "promising" edges; after that we run independently hierarchical testing on each of them, in order to pinpoint the conditions where they are present. In practice, we implemented the procedure in the following way: in order to verify if an edge is significant or not, we consider the maximum tail probability on its row, representing the evidence of it being present in at least one condition. If it is lower than the given threshold α, corrected for multiplicity with Bonferroni (α/N), then hierarchical testing is applied on it. The choice of adjusting for multiplicity using Bonferroni approach is dictated by the necessity of coherency between the two steps: since hierarchical testing controls FWER (Family-Wise Error Rate), we need to ensure the same type of control also at the first level.
For all the methods we set the threshold α at 0.1; hierarchical BH allows different test levels α_1 and α_2 for the two selection steps: we set them both equal to 0.1.
As we did for in the previous comparison, we will comment in details all four

Method	TP	FN	FP	TN	TPR (%)	FPR (%)
hierTest_BF	170	780	**10**	**18940**	17.89	**0.0528**
hierBH	**469**	**481**	149	18801	**49.37**	0.7863
hierTest_BF	184	766	**10**	**18940**	19.37	**0.0528**
hierBH	**483**	**467**	118	18832	**50.84**	0.6227
hierTest_BF	196	754	**10**	**18940**	20.63	**0.0528**
hierBH	**511**	**439**	86	18864	**53.79**	0.4538
hierTest_BF	206	744	**10**	**18940**	21.68	**0.0528**
hierBH	**526**	**424**	61	18889	**55.37**	0.3219
hierTest_BF	215	735	**8**	**18942**	22.63	**0.0422**
hierBH	**553**	**397**	34	18916	**58.21**	0.1794
hierTest_BF	49	901	**3**	**18947**	5.16	**0.0158**
hierBH	331	619	12	18938	34.84	0.0633
UI test	**568**	**382**	43	18907	**59.79**	0.2269

Table 5.7: Performance indicators for methods `hierTest_BF`, `hierBH` and `UI_test` (only for common edge detection). 5 conditions, same block structure, $p = 200$ and $n_k = 50, 70, 100, 120, 150$. A single horizontal line separates the conditions; under the double line, same performance indicators for common edge detection.

scenarios regarding the simulation setting with 5 conditions. Here, we will consider the simulations with block structure (displayed in Figure 5.2).
In Figure 5.11 a graphical representation of how methods `hierTest_BF` (upper row) and `hierBH` (lower row) perform is shown. Colour green denotes the True Positives (true edges correctly detected), blue the False Negatives (true edges not detected) and red the False Positives (edges wrongly detected as present). White colour denotes the others, i.e. the True Negatives. At a glance, we immediately spot a lack of power in method `hierTest_BF`, whose estimations are dominated by colour blue. For a more quantitative insight, we refer to Table 5.6: as expected, `hierBH` detects many more edges than `hierTest_BF`. However, we notice that False Postive Rate (FPR) increased, but it is still quite low.
When considering the case with different number of observations (Table 5.6), we notice, unsurprisingly, that the detection ability improves with the sample size: indeed, TPR gets higher and, simultaneously, FPR gets lower as we increase the number of observations, from 50 in the first condition (first block in the table) to 150 (last block before the double horizontal line).
Moving to the case with partially shared network structure (Tables 5.8 and 5.9), we observe the same dependence on the sample size and a slight improve-

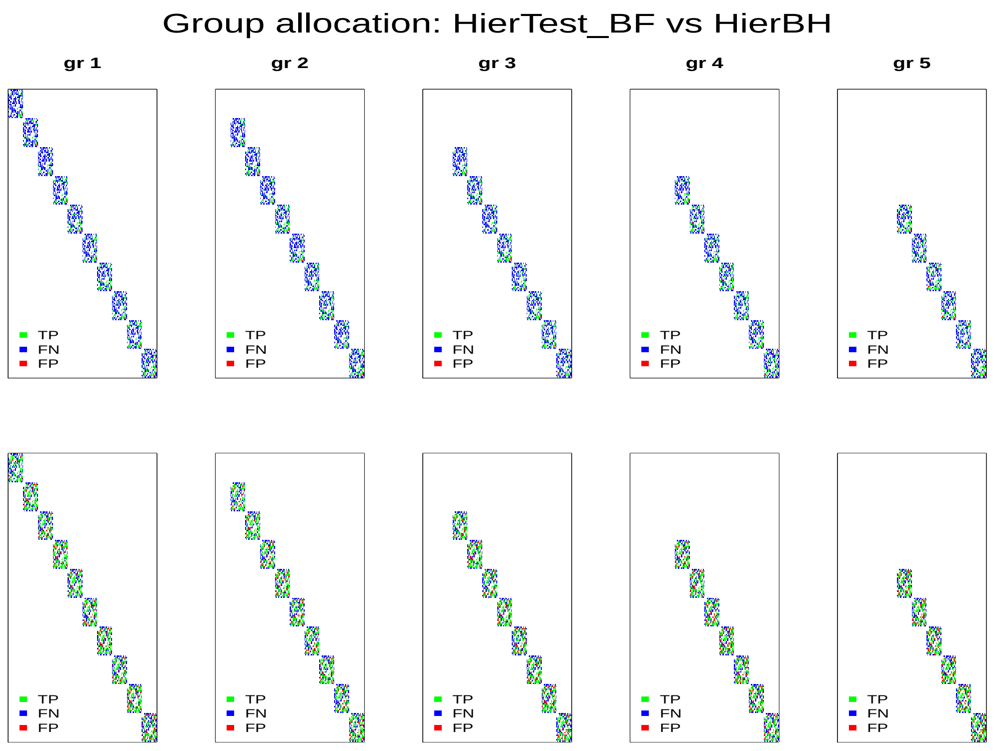


Figure 5.12: Network inference performed by methods `hierTest_BF` (upper row) and `hierBH` (lower row). 5 conditions, partially shared block structure, $p = 200$ and $n_k = 100$ in all conditions. True Positives (green), False Negatives (blue) and False Positives (red) are displayed.

ment of the performances for both methods, which becomes more significant as the number of blocks contained in the matrix decreases. This might be a peculiarity of this specific simulation setting: indeed, the heterogeneity is here simulated by gradually removing some blocks form the true precision matrices. This means that there are no edges specific to a single condition (expect the first block of the first condition): they are all shared between at least two conditions. This setting seems to be favourable for both methods. We perform the same comparison (5 conditions, same number of observations, totally vs partially shared network structure) in the case with band matrices (Table 5.10 and 5.11, respectively): in this setting we notice the opposite trend, i.e. both methods, `hierBH` in particular, perform better when all conditions have the same network structure.
In Tables 5.12 and 5.13 we report the results for the setting with same block structure and same number of observations, relative to 2 and 10 conditions, respectively.

Method	TP	FN	FP	TN	TPR (%)	FPR (%)
hierTest_BF	234	716	**12**	**18938**	24.63	**0.0633**
hierBH	**549**	**401**	168	18782	**57.79**	0.8865
hierTest_BF	230	625	**10**	**19035**	26.90	**0.0525**
hierBH	**505**	**350**	157	18888	**59.06**	0.8244
hierTest_BF	227	533	**13**	**19127**	29.87	**0.0679**
hierBH	**449**	**311**	157	18983	**59.08**	0.8203
hierTest_BF	220	445	**15**	**19220**	33.08	**0.0780**
hierBH	**395**	**270**	143	19092	**59.40**	0.7434
hierTest_BF	196	374	**20**	**19310**	34.39	**0.1035**
hierBH	**345**	**225**	139	19191	**60.53**	0.7191
hierTest_BF	18	77	**89**	**19716**	18.95	**0.4494**
hierBH	**44**	**51**	277	19528	**46.32**	1.3986
UI test	36	59	291	19514	37.89	1.4693

Table 5.8: Performance indicators for methods `hierTest_BF`, `hierBH` and `UI_test` (only for common edge detection). 5 conditions, partially block structure, $p = 200$ and $n_k = 100$ in all conditions. A single horizontal line separates the conditions; under the double line, same performance indicators for common edge detection.

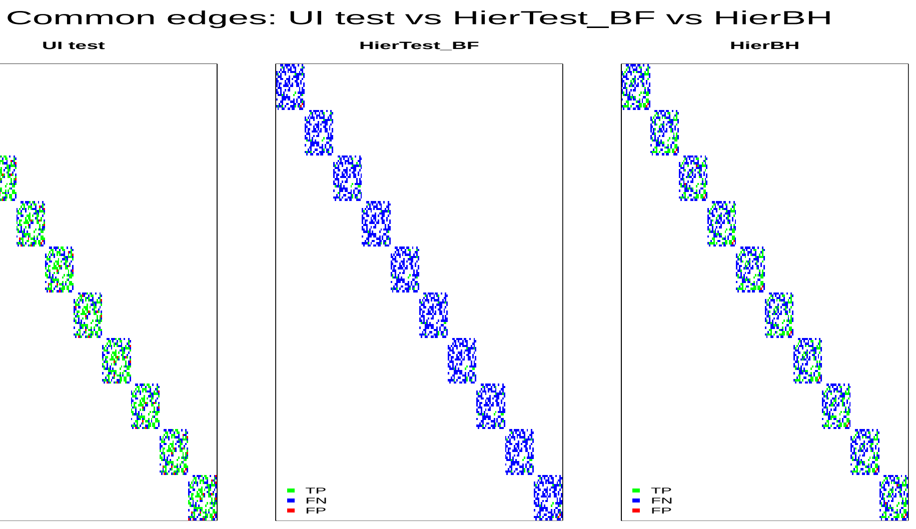


Figure 5.13: Common edge detection performed by methods `UI test` (left), `hierTest_BF` (center) and `hierBH` (right). 5 conditions, same block structure, $p = 200$ and $n_k = 100$ in all conditions. True Positives (green), False Negatives (blue) and False Positives (red) are displayed.

Method	TP	FN	FP	TN	TPR (%)	FPR (%)
hierTest_BF	201	749	**20**	**18930**	21.16	**0.1055**
hierBH	**502**	**448**	231	18719	**52.84**	1.219
hierTest_BF	229	626	**29**	**19016**	26.78	**0.1523**
hierBH	**475**	**380**	227	18818	**55.56**	1.1919
hierTest_BF	244	516	**26**	**19114**	32.11	**0.1358**
hierBH	**459**	**301**	190	18950	**60.39**	0.9927
hierTest_BF	239	426	**21**	**19214**	35.94	**0.1092**
hierBH	**411**	**254**	165	19070	**61.80**	0.8578
hierTest_BF	238	332	**22**	**19308**	41.75	**0.1138**
hierBH	**364**	**206**	135	19195	**63.86**	0.6984
hierTest_BF	16	79	**94**	**19711**	16.84	**0.4746**
hierBH	**44**	**51**	302	19503	**46.32**	1.5249
UI test	35	60	226	19579	36.84	1.1411

Table 5.9: Performance indicators for methods `hierTest_BF`, `hierBH` and `UI_test` (only for common edge detection). 5 conditions, partially shared block structure, $p = 200$ and $n_k = 50, 70, 100, 120, 150$. A single horizontal line separates the conditions; under the double line, same performance indicators for common edge detection.

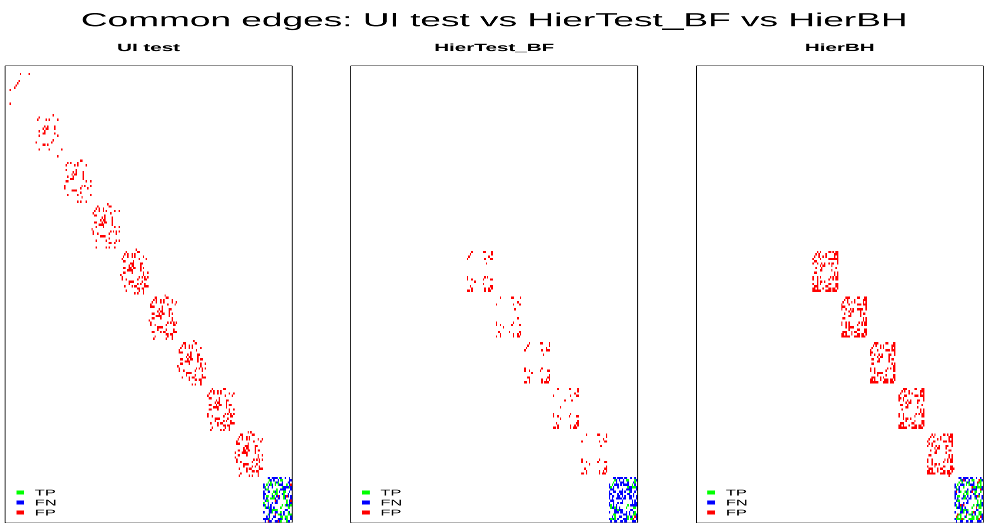


Figure 5.14: Common edge detection performed by methods `UI test` (left), `hierTest_BF` (center) and `hierBH` (right). 5 conditions, partially shared block structure, $p = 200$ and $n_k = 100$ in all conditions. True Positives (green), False Negatives (blue) and False Positives (red) are displayed.

Method	TP	FN	FP	TN	TPR (%)	FPR (%)
hierTest_BF	52	933	**0**	**18915**	5.28	**0**
hierBH	**256**	**729**	12	18903	**25.99**	0.0634
hierTest_BF	52	933	**0**	**18915**	5.28	**0**
hierBH	**259**	**726**	10	18905	**26.29**	0.0529
hierTest_BF	53	932	**0**	**18915**	5.38	**0**
hierBH	**264**	**721**	11	18904	**26.80**	0.0582
hierTest_BF	55	930	**0**	**18915**	5.58	**0**
hierBH	**263**	**722**	13	18902	**26.70**	0.0687
hierTest_BF	57	928	**0**	**18915**	5.79	**0**
hierBH	**266**	**719**	9	18906	**27.01**	0.0476
hierTest_BF	14	971	**0**	**18915**	1.42	**0**
hierBH	105	880	2	18913	10.66	0.0106
UI test	**244**	**741**	**0**	**18915**	**24.77**	**0**

Table 5.10: Performance indicators for methods `hierTest_BF`, `hierBH` and `UI_test` (only for common edge detection). 5 conditions, same band structure, $p = 200$ and $n_k = 100$ in all conditions. A single horizontal line separates the conditions; under the double line, same performance indicators for common edge detection.

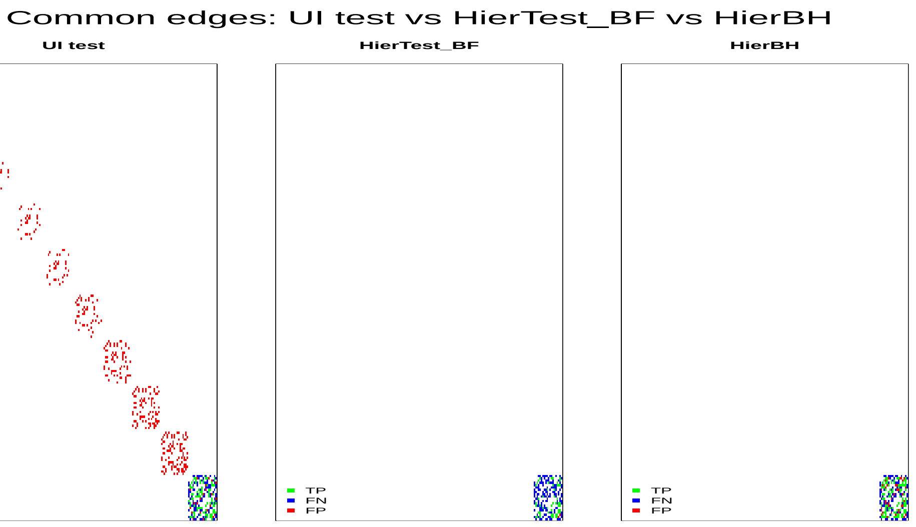


Figure 5.15: Common edge detection performed by methods `UI test` (left), `hierTest_BF` (center) and `hierBH` (right). 10 conditions, partially shared block structure, $p = 200$ and $n_k = 100$ in all conditions. True Positives (green), False Negatives (blue) and False Positives (red) are displayed.

Method	TP	FN	FP	TN	TPR (%)	FPR (%)
hierTest_BF	56	929	**3**	**18912**	5.69	**0.0159**
hierBH	**145**	**840**	17	18898	**14.72**	0.0899
hierTest_BF	58	927	**3**	**18912**	5.89	**0.0159**
hierBH	**141**	**844**	19	18896	**14.31**	0.1004
hierTest_BF	61	924	**2**	**18913**	6.19	**0.0106**
hierBH	**148**	**837**	19	18896	**15.03**	0.1004
hierTest_BF	70	915	**1**	**18914**	7.11	**0.0053**
hierBH	**149**	**836**	16	18899	**15.13**	0.0846
hierTest_BF	61	924	**3**	**18912**	6.19	**0.0159**
hierBH	**154**	**831**	19	18896	**15.63**	0.1004
hierTest_BF	25	531	**1**	**19343**	4.5	**0.0052**
hierBH	**68**	**488**	18	19326	**12.23**	0.0931
UI test	29	527	13	19331	5.22	0.0672

Table 5.11: Performance indicators for methods `hierTest_BF`, `hierBH` and `UI_test` (only for common edge detection). 5 conditions, partially shared band structure, $p = 200$ and $n_k = 100$ in all conditions. A single horizontal line separates the conditions; under the double line, same performance indicators for common edge detection.

Method	TP	FN	FP	TN	TPR (%)	FPR (%)
hierTest_BF	187	763	**10**	**18940**	19.68	**0.0528**
hierBH	**514**	**436**	76	18874	**54.11**	0.4011
hierTest_BF	190	760	**10**	**18940**	20.00	**0.0528**
hierBH	**517**	**433**	77	18873	**54.42**	0.4063
hierTest_BF	120	830	9	18941	12.63	0.0475
hierBH	**437**	**513**	33	18917	**46**	0.1741
UI test	257	693	**4**	**18946**	27.05	**0.0211**

Table 5.12: Performance indicators for methods `hierTest_BF`, `hierBH` and `UI_test` (only for common edge detection). 2 conditions, same block structure, $p = 200$ and $n_k = 100$ in both conditions. A single horizontal line separates the conditions; under the double line, same performance indicators for common edge detection.

Method	TP	FN	FP	TN	TPR (%)	FPR (%)
hierTest_BF	199	751	**9**	**18941**	20.95	**0.0475**
hierBH	**514**	**436**	68	18882	**54.11**	0.3588
hierTest_BF	194	756	**10**	**18940**	20.42	**0.0528**
hierBH	**513**	**437**	74	18876	**54.00**	0.3905
hierTest_BF	197	753	**10**	**18940**	20.74	**0.0528**
hierBH	**514**	**436**	73	18877	**54.11**	0.3852
hierTest_BF	196	754	**10**	**18940**	20.63	**0.0528**
hierBH	**503**	**447**	75	18875	**52.95**	0.3958
hierTest_BF	198	752	**10**	**18940**	20.84	**0.0528**
hierBH	**520**	**430**	82	18868	**54.74**	0.4327
hierTest_BF	212	738	**10**	**18940**	22.32	**0.0528**
hierBH	**519**	**431**	83	18867	**54.63**	0.4380
hierTest_BF	198	752	**10**	**18940**	20.84	**0.0528**
hierBH	**508**	**442**	73	18877	**53.47**	0.3852
hierTest_BF	209	741	**10**	**18940**	22.00	**0.0528**
hierBH	**509**	**441**	68	18882	**53.58**	0.3588
hierTest_BF	186	764	**9**	**18941**	19.58	**0.0475**
hierBH	**513**	**437**	67	18883	**54.00**	0.3536
hierTest_BF	197	753	**10**	**18940**	20.74	**0.0528**
hierBH	**511**	**439**	78	18872	**53.79**	0.4116
hierTest_BF	32	918	**0**	**18950**	3.37	**0**
hierBH	286	664	10	18940	30.11	0.0528
UI test	**709**	**241**	187	18763	**74.63**	0.9868

Table 5.13: Performance indicators for methods `hierTest_BF`, `hierBH` and `UI_test` (only for common edge detection). 10 conditions, same block structure, $p = 200$ and $n_k = 100$ in all conditions. A single horizontal line separates the conditions; under the double line, same performance indicators for common edge detection.

From them, we can see that the number of conditions does not really change the performances of the two methods. This is related to the prior we are using in our model: as discussed in Section 3.3, if the borrowing of information is enabled only through hyperparameter δ, the estimate provided by the joint model is almost the same as that provided by K separate models. This is particularly true when underlying structure and sample sizes are identical across conditions, as in the case we are commenting.

Regarding the common edges, we can observe form the tables illustrated so far that the performances deteriorate as we increase the heterogeneity across conditions. As an example, we compare Figures 5.13 and 5.14, displaying common edges detection with methods `UI test`, `hierTest_BF` and `hierBH`: the corresponding numerical data can be found in Tables 5.6 and 5.8. When the underlying network structure is identical (Figure 5.13), all methods recover it correctly, `UI test` in particular; if we introduce some heterogeneity (Figure 5.14), however, we notice that lots of edges are wrongly detected as present. They concentrate, quite intuitively, in those blocks that are shared across some of the conditions. This effect becomes less evident when we increase the number of conditions, as we can see in Figure 5.15, where we display the same setting just described with $K = 10$.
We finally observe that we are measuring some of the many performance indicators available. They give already a general overview about our methods: still, TPR and FPR are not enough by themselves to highlight all their properties. Let us focus on FPR, for instance: it is always very low, which is a good sign. However, recalling its formula, $FPR = FP/(FP + TN)$, we see that at the denominator there is the number of True Negatives, which is extremely high, given that we are dealing with very sparse matrices in our simulations. Even a considerable number of False Positives can be somehow concealed in the ratio with True Negatives. If, instead, we computed the Precision index ($prec = TP/(TP + FP)$), we would observe much worse results: indeed, the number of False Positives is often in the same order of magnitude as the True Positives. This holds true especially for `hierBH`, while `hierTest_BF` usually attains fair results in terms of precision, being it intrinsically more conservative.
We show an extreme example in Table 5.9: the last block of results, under the double horizontal lines, refers to the detection of common edges: if we look only at TPR and FPR percentages, we do not spot anything too wrong, unless a general lack of power. However, looking also at the other columns, we notice that the number of True Positives is way lower than that of False Positives. This means that, in a real case scenario, where we do not know whether the edges that have been detected are actually present or not, we have no confidence about the selected edges at all.

Summing up, we can draw some conclusions. The performances are not completely satisfactory: in particular, we observed a general lack of power. For these simulations we used the model which sets the target matrix to the identity. As already discussed, this approach is not optimal and can be improved by allowing the sharing of information also by means of hyperparameter M. It is likely that we will achieve better results by changing the model accordingly.
In the simulations we presented, we notice a general trade-off: `hierBH` usually has almost double power than `hierTest_BF`, but it pays a small prize in terms of False Positives.
Thus, we think that the choice of the method to use should be driven by the application: if more interested in detecting as many edges as possible, regardless of some False Positives, in view of a subsequent confirmatory analysis, we suggest `hierBH`. Method `hierTest_BF` should be privileged if the main concern is about precision.

5.2 Application to real data

In this last section we are going to apply our method to a real dataset, in order to show an example of analysis that can be performed. An in-depth interpretation of the results is beyond our scope, as it requires a deep knowledge in molecular biology. Still, we can make some general observations and see it the method outputs plausible results.
The data are divided in 19 conditions, representing 19 cancer types; each of them contains the protein expression relative to 217 proteins.
Moreover, in the paper by Şenbabaoğlu et al. [2016] a previous version of the same data collection (containing 187 proteins for 11 cancer types and with slightly smaller sample size) is analysed. Thus, we can compare our findings with theirs, to see if they are reasonable. Before presenting in more details the dataset, we will give a brief overview about what protein expression is. We will refer to Hunter [1993], who wrote a short compendium on molecular biology for computer scientists, and to the paper by Mishra et al. [2015].

5.2.1 Protein expression

All the characteristics an organism inherits are contained in a single messenger molecule: deoxyribonucleic acid, or DNA. The information is represented in a simple, linear, four-element code. The complexity comes into play in the translation of this code into all the inherited characteristics. The particular genetic encoding for an organism is called its genotype; the resulting physi-

cal characteristics are called its phenotype. The difference between genotype and phenotype is very important, because small changes at genotype level can have large consequences on the phenotype.
All of the genetic material in an organism is called its genome. Genetic material is discrete and hence has a particular size, which varies from about 5000 elements in very simple organisms to more than 10^{11} in some higher plants; people's genome is made of about 3×10^9 elements. The size of the genome is not directly related to the complexity of the organism.
All living things are made of cells: in multicellular organisms they specialize to accomplish specific tasks. Tissues are groups of cells specialized for a particular function. In people there are fourteen major tissue types. Despite these variations, all cells in a multicellular organism have exactly the same genetic code. What differs is their gene expression, that is, whether or not the product a gene codes for is produced, and how much is produced.
Inside each cell, genetic messages are translated into the main type of biological molecule, the proteins, which accomplish most of its functions. The "central dogma of molecular Biology" (Crick [1970]), which says that DNA is transcribed in RNA and subsequently translated into proteins, is a very simplistic way to describe the flow of information. The genetic regulatory process is very elaborate and intricate: genes code for products that turn on and off other genes, which in turn regulate other genes, and so on.
One of the key research areas aims at studying how the control of gene expression is managed, how cells "know" what to differentiate into and what makes them go wrong (e.g. the cancer cells). To understand this Biologists take measurements at different molecular levels of the cell (DNA, RNA, Protein, Methylation, microRNA,...), protein expression among the others. Protein expression quantifies the amount of protein that is produced.
The analysis we are going to present falls into the category of Pan-cancer analyses, whose general aim is to examine similarities and differences among the genomic and cellular alterations found across diverse cancer types. In their paper, Mishra et al. [2015] claim that it is particularly relevant to perform Pan-cancer analysis at protein level, since it is more immediately related to patient phenotype than genomic or transcriptomic data.

5.2.2 Analysis of Pan-cancer data

The dataset *DataListPanCan19* we analysed comes from the data portal TCPA (The Cancer Proteome Atlas), illustrated in Li et al. [2013] and it is directly available in `beamDiff` R package.
The pan-cancer dataset contains normalized protein expression data relative to $p = 217$ proteins.

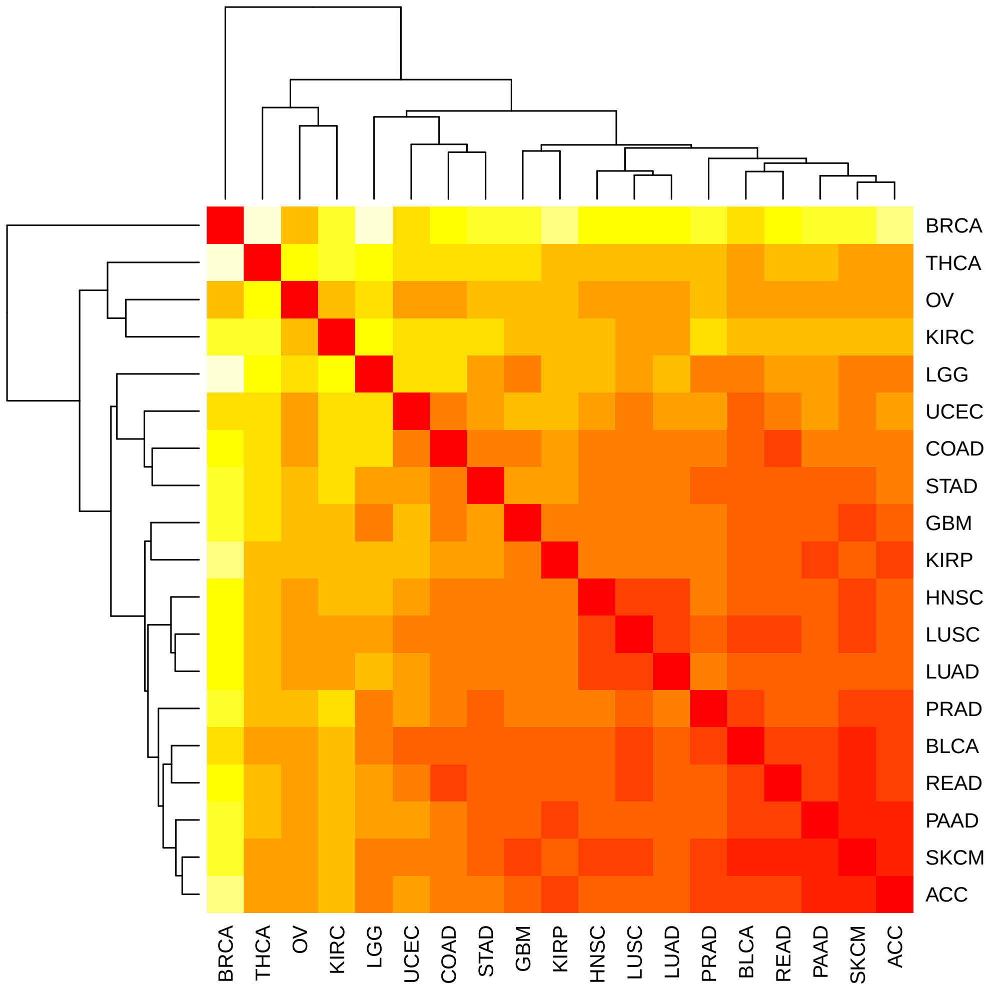


Figure 5.16: Heatmap for pairwise symmetric KL-divergence on *DataList-PanCan19* dataset. The colour scale goes from yellow (0) to red (1).

Acronym	Cancer type	Sample size
ACC	Adenoid Cystic Carcinoma	46
BLCA	Bladder Urothelial Carcinoma	127
BRCA	Breast invasive carcinoma	815
COAD	Colon adenocarcinoma	327
GBM	Glioblastoma multiforme	205
HNSC	Head and Neck squamous cell carcinoma	203
KIRC	Kidney renal clear cell carcinoma	445
KIRP	Kidney renal papillary cell carcinoma	208
LGG	Brain Lower Grade Glioma	257
LUAD	Lung adenocarcinoma	234
LUSC	Lung squamous cell carcinoma	192
OV	Ovarian serous cystadenocarcinoma	411
PAAD	Pancreatic adenocarcinoma	105
PRAD	Prostate adenocarcinoma	164
READ	Rectum adenocarcinoma	129
SKCM	Skin Cutaneous Melanoma	36
STAD	Stomach adenocarcinoma	299
THCA	Thyroid carcinoma	372
UCEC	Uterine Corpus Endometrial Carcinoma	404

Table 5.14: Cancer types present in *DataListPanCan19* dataset. The sample size of each condition is also reported.

The $K = 19$ cancer types are reported in Table 5.14, together with the sample size n_k. We perform the estimation with function `beamDiff`. Once obtained the estimates for marginal and partial correlation matrices, we can already compute one of the measures from information theory, discussed in Section 3.4. In Figure 5.16 the symmetrized version of KL divergence is displayed on a heatmap, with a dendrogram representing the hierarchical clustering (Complete linkage and Euclidean distance). From it, we can have a first insight about group-relatedness.
We can identify, for instance, a group of cancer types at the bottom right corner, composed by ACC, SKCM and PAAD: these groups, however, have relatively small sample sizes, therefore the resulting relatedness might be due to their similarity with the empty graph, rather than among them. More interestingly, we observe in the middle a cluster with LUAD, LUSC and HNSC: this cluster appears also in the analysis by Şenbabaoğlu et al. [2016], where it considered plausible, since it gathers two lung cancers and two squamous cell carcinomas.

Cancer type	sample size	hierBH	hierTest_BF	methods coherency (%)
ACC	46	283	16	100
BLCA	127	423	47	100
BRCA	815	1227	108	100
COAD	327	718	59	100
GBM	205	594	52	100
HNSC	203	515	47	100
KIRC	445	935	76	100
KIRP	208	650	59	100
LGG	257	738	68	100
LUAD	234	583	67	100
LUSC	192	551	54	100
OV	411	766	78	100
PAAD	105	487	53	100
PRAD	164	540	53	100
READ	129	434	45	100
SKCM	36	273	8	100
STAD	299	672	79	100
THCA	372	972	70	100
UCEC	404	693	84	100

Table 5.15: Number of edges selected by methods `hierBH` (third column) and `hierTest_BF` (fourth column) out of the 23436 present in the complete graph. In the last column the proportion of edges detected by `hierTest_BF` present also in `hierBH` selection is reported.

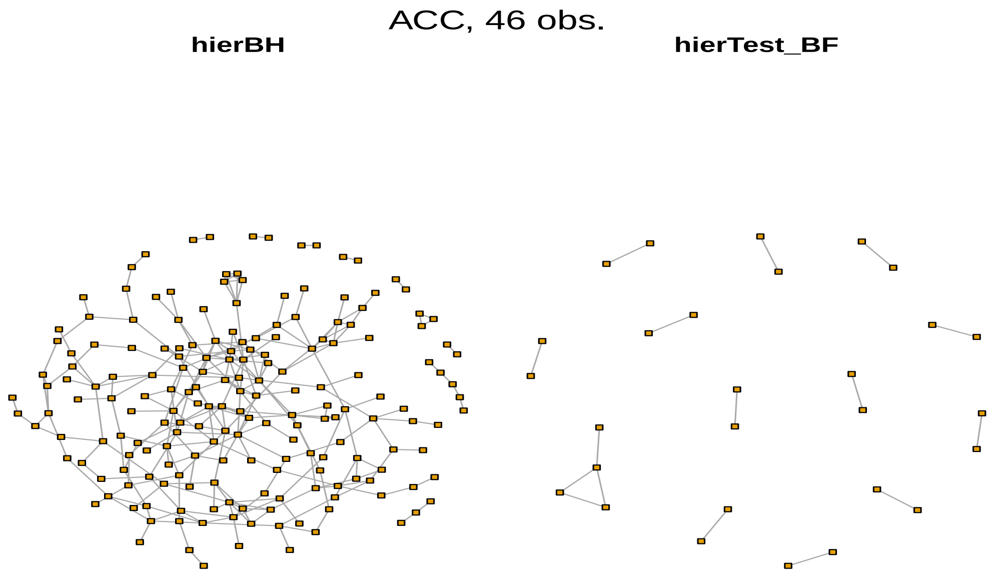


Figure 5.17: Network structure inferred by methods `hierBH` and `hierTest_BF` in cancer type ACC of *DataListPanCan19* dataset.

In order to retrieve the network structures, we apply both `hierBH` and `hierTest_BF`, with threshold at 10%. We expect the latter to output sparser networks, but we can still check whether the edges it selects are a proper subset of those selected by `hierBH`. Figures 5.17, 5.18 and 5.19 confirm our hypothesis: no matter the sample size, `hierTest_BF` outputs a much sparser structure that `hierBH`. From the same figures, we also notice that the sparsity of the graphs seems to depend on the number of observations. In particular, in those with small sample size (cancer types ACC and SKCM) the inferred networks are really sparse: due to the weakness of signal, very few edges are detected as present. On the contrary, the network estimated for BRCA, characterized by a high number of observations, contains many more edges than the others. A possible explanation might be that the sparsity assessed in BRCA is actually the closest to the true one and the problem lies in the other groups, where we do not have power enough to retrieve the entire structure. In Table 5.15 we report, for each condition, the number of edges selected by the two methods; the last column shows the proportion of edges detected by `hierTest_BF` which are detected by `hierBH`, as well. In all groups, we observe

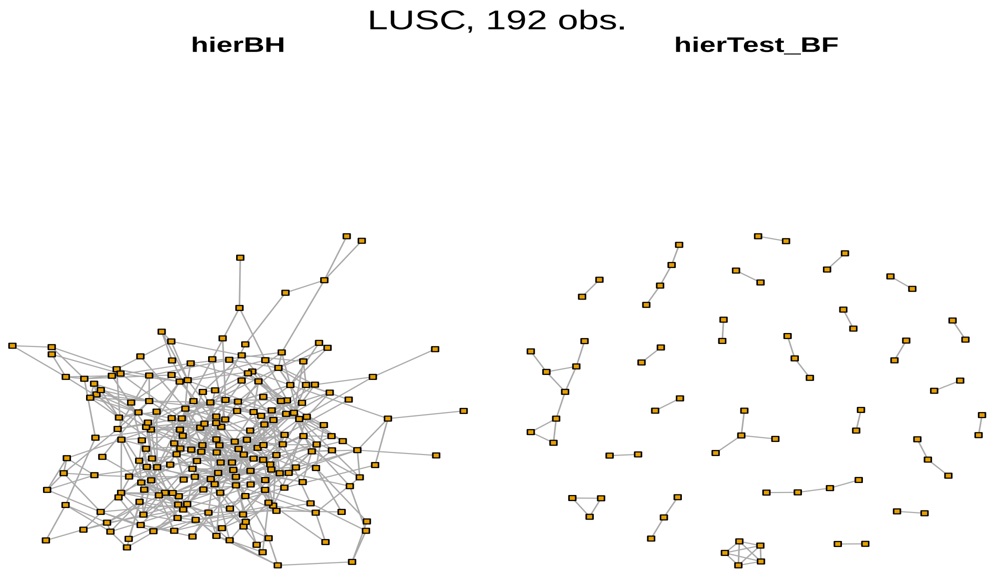


Figure 5.18: Network structure inferred by methods `hierBH` and `hierTest_BF` in cancer type LUSC of *DataListPanCan19* dataset.

perfect consensus between the two. Thus, we believe that the inference by `hierTest_BF`, although too sparse to be meaningful, can provide additional information to that by `hierBH`: indeed, recalling that `hierTest_BF` estimate is characterized by very low False Positive Rate, the edges selected by both are more likely to be True Positives. Lastly, we can verify the relatedness we read on the KL divergence heatmap, by computing the proportion of shared edges between cancer types that were clustered together and between groups that were far apart. The results we will report refer to `hierBH` estimate.
Let S_i be the set of edges selected in group i, the proportion of shared edges between group i and group j, $\tau_{i,j}$, is defined as the ratio between the number of edges present in both networks and the minimum between the cardinality of S_i and S_j:

$$\tau_{i,j} = \frac{card\left(\{S_i \cap S_j\}\right)}{min\left(card(S_i), card(S_j)\right)}.$$

We compute this index for the pairs LUAD-LUSC, LUAD-HNSC and LUAD-LGG; the results are the following: $\tau_{\text{LUAD,LUSC}} = 0.403$, $\tau_{\text{LUAD,HNSC}} = 0.371$ and $\tau_{\text{LUAD,LGG}} = 0.228$. As expected, it is higher in the first two cases, where

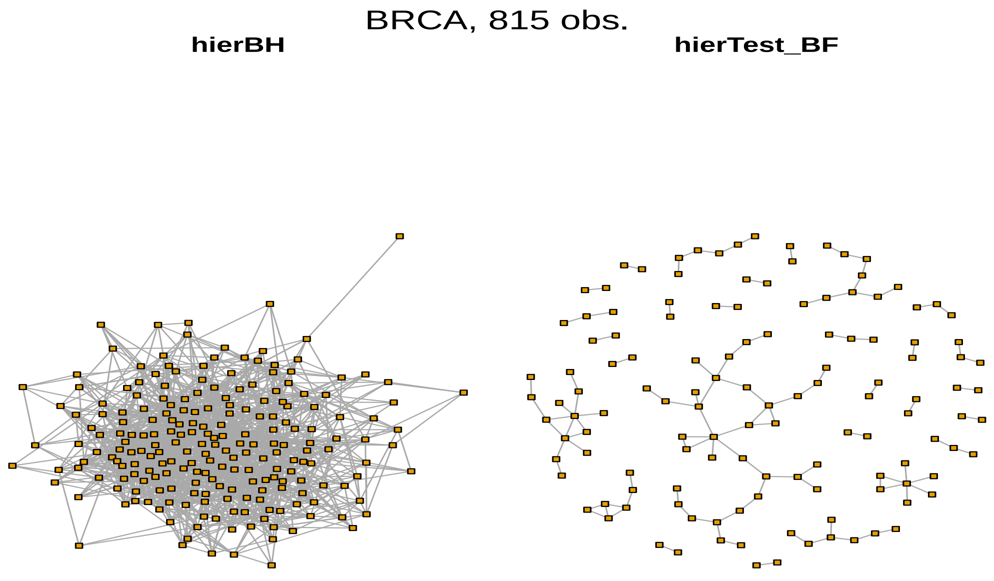


Figure 5.19: Network structure inferred by methods `hierBH` and `hierTest_BF` in cancer type BRCA of *DataListPanCan19* dataset.

we considered two groups in its same cluster.
To conclude, we remind that the estimation step has been performed using `beamDiff`. As we showed in Section 5.1, it does not borrow information enough to gain substantial power on conditions characterized by small sample size. The analysis we have just presented is a further confirmation of this weakness. Most likely, by using at estimation stage `beamDiff_boost` or `beamDiff_new`, we would be able to gain power, especially in conditions with few observations, and to achieve more precise estimates.

Conclusions and further developments

The aim of my thesis work was the theoretical and computational development of a method for the joint inference of multiple Gaussian graphical models. In Chapter 1, after introducing the theoretical framework of multivariate Gaussian models and association graphs, we presented the work by Leday and Richardson [2018], which has been the starting point of my project.
In Chapter 2 we described the optimizations performed on R package `beam`, aimed at improving memory allocation and execution time. We achieved an improvement of 40% on the size of the output object and we halved the execution time.
In Chapter 3 we presented our model for the joint inference of multiple Gaussian graphical models. As the model by Leday and Richardson [2018], it is divided in two steps: estimation (closed-form computation of posterior expectation of marginal and precision matrices) and selection (the network structure is retrieved by testing the presence of each edge).
At the estimation stage, we noticed that setting the target matrix to the identity, as Leday and Richardson [2018] did, was not the best choice. Indeed, we observed on simulated data that our joint estimation did not improve with respect to the individual group-by-group estimation. Aware of that, we suggested two methods to estimate the target matrix with an empirical Bayes approach. The first one, based on the work by Bilgrau et al. [2015] and implemented in function `beamDiff_boost`, is a EM algorithm for the iterative optimization of the joint marginal likelihood of the model in both its variables, M and δ. In the second, implemented in function `beamDiff_beam`, the target matrix $T = M/(\delta - p - 1)$ is set equal to the posterior expectation of the marginal correlation matrix, obtained by running `beam` on all data merged together. In this way, the target matrix will embed some information about the common structure shared by all conditions. Unfortunately, we were not able to develop the entire model based on these new approaches

to the estimation. Indeed, the theory to derive the tail probabilities in the general setting of hyper-parameter M being non-diagonal has yet to be developed.
At selection stage we had to cope with double multiplicity: multiple edges in multiple conditions. We discussed different methods to deal with it, both in a confirmatory and exploratory framework.
In the same chapter, we also introduced some indicators, generally used in information theory, useful to quantify the similarity between groups. Thanks to the conjugacy of our model, they can be computed analytically and in a computationally efficient way. We showed that KL divergence is particularly well-suited as measure of distance between conditions, since it is the same for marginal and partial correlation.
Chapter 4 is devoted to the computational development of the R package `beamDiff`, which heavily relies on R package `beam`, and to data simulation. In particular, we discussed many issues related to the simulation of sparse positive definite matrices in a high-dimensional setting. We concluded illustrating two functions, `simBlockMatrix` and `simBandShuffle`, we implemented to simulate the general framework of multiple conditions, which partially share a common network structure. With respect to other methods we learnt from related papers, ours allow the control of many parameters, such as the sparsity of the matrix, the condition number or the range of magnitude of the non-zero entries.
In Chapter 5 we presented some results from the extensive simulations we carried out both at estimation and selection steps. Regarding the estimation, we have shown that, most of the time, `beamDiff_boost` and `beamDiff_new` provide much better estimates with respect to `beamDiff`.
Actually, `beamDiff_boost` proved to have a severe drawback in settings with few conditions and/or small sample sizes: the estimated prior matrix M can be ill-conditioned. This ill-conditioning will add numeric noise to the estimation, making it completely unreliable. For this reason, we would opt for `beamDiff_new`, whose estimate is much more well-conditioned.
Regarding the selection, we noticed a general lack of power affecting all methods, `hierTest_BF` in particular. However, this might be due also to the estimates obtained at the first step of our method. We believe that the best method, among those proposed, is `hierBH`, either for the good performances and for the fact that in the paper by Peterson et al. [2016] where it has been introduced it was applied to a setting very similar to ours.
In the last section of the chapter we introduced and analysed a dataset containing protein expression for 19 cancer types.
During the project, we also tried to compare our method with some competitors, first of all Joint Graphical Lasso, proposed by Danaher et al. [2014].

The method has been implemented in the R package `JGL` (Danaher [2013]). We decided not to include this comparison here, because of many issues; the most important is that every method is tailored to perform well on a specific statistical question. It is therefore very difficult to make a comparison which is fair for all competitors. Moreover, most of the algorithms we took into consideration were computationally very demanding; `JGL` itself took a really long time to tune its hyperparameters.
Summing up, the output of my Master thesis project has been a working extension of the method proposed by Leday and Richardson [2018] to the multi-condition case. We also managed in the objective of maintaining its strength points: indeed, our method is as fast as applying `beam` to each condition separately and we still control the type I error.

Future work

To conclude, a brief overview about some possible future developments. The very fist goal is to extend the theory to derive the tail probabilities to the case of non-diagonal prior matrix. Once it will be available, we could use at estimation stage the function `beamDiff_new`. In parallel, also R package `beamDiff` needs to be finished, by implementing the function `beamDiff.select` for the selection step and some auxiliary methods.
To further extend our method we could return a weighted graph, with weight depending on the tail probability associated to that edge: the smaller the tail, the higher the weight, since we are more confident about its presence. Another development could be to allow the method to borrow information only between groups that are similar. In our current setting, all conditions are shrunk towards a unique common matrix: in order to better deal with heterogeneous data, it would be interesting to allow multiple targets towards which the conditions can be shrunk.
Under a computational point of view, we believe that, especially in the multi-condition framework, a further optimization of memory allocation would allow to scale at even higher-dimensions. A possible solution consists in storing at estimation step a matrix that is already sparse; this is achieved by considering as exact zeros those entries which are "close enough" to it. An accurate theoretical analysis is required to study the properties of such an approximation.

Appendix A

Documentation of R package 'beam'

The method presented in Leday and Richardson [2018] has been implemented in R package `beam`, now available on the CRAN. Its documentation is here reported: the two main functions `beam` and `beam.select` are described in details, together with the methods acting on the output classes `beam-class` and `beam.select-class`.
In Chapter 2 the optimizations carried out on the code are presented: clearly, in the package the optimized version is implemented. Indeed, it can be seen in `beam-class` documentation that there is a unique data.frame slot, containing all estimations for both marginal and partial correlation.
Among the methods it is worth mentioning `plotCor`, acting on objects of `beam-class`: it outputs the heatmap of marginal and/or partial correlation. This tool is useful to have a first graphical insight about the features more highly correlated.
Methods `ugraph` and `bgraph` act, instead, on objects of `beam.select-class`: they return an `igraph` object representing the conditional and marginal, respectively, dependence structure as a graph. `igraph` R package provides several tools for further analyses on the graphs.

Package ‘beam’

March 23, 2018

Type Package

Title Fast Bayesian Inference in Large Gaussian Graphical Models

Version 1.0

Date 2018-03-06

Author Gwenael G.R. Leday [cre, aut],
Ilaria Speranza [aut],
Harry Gray [ctb]

Maintainer Gwenael G.R. Leday <gwenael.leday@mrc-bsu.cam.ac.uk>

Depends R (>= 3.1.0)

Imports stats, methods, grDevices, graphics, Matrix, fdrtool, igraph,
knitr

Description
Fast Bayesian inference of marginal and conditional independence structures between variables from high-dimensional data (Leday and Richardson (2018) <arXiv:1803.08155>).

LazyLoad yes

License GPL (>= 2.0)

LazyData TRUE

NeedsCompilation yes

Repository CRAN

RoxygenNote 5.0.1

Date/Publication 2018-03-23 17:25:57 UTC

R topics documented:

`beam-package` *Fast Bayesian Inference in Large Gaussian Graphical Models*

Description

The package enables the reconstruction of marginal and conditional independence structures between variables using the method of Leday and Richardson (2018). Inference is carried out by multiple testing of hypotheses about pairwise (marginal or conditional) independence using closed-form Bayes factors. Exact tail probabilities are obtained from the null distributions of the Bayes factors to help address the multiplicity problem and control desired error rates for incorrect edge inclusion. The method is computationally very efficient and allows to address problems with hundreds or thousands of variables.

Details

The main function of the package is `beam` which carries out (inverse) covariance estimation and compute (scaled) Bayes factors as well as tail probabilities (p-values). The function returns an (S4) object of class `beam-class` that is associated with the following methods:

- `summary,beam-method`:
provides a summary of inferred (marginal or conditional) associations.

- `marg,beam-method`:
returns a data.frame with marginal correlations, Bayes factors and/or tail probabilities.

- `cond,beam-method`:
returns a data.frame with partial correlations, Bayes factors and/or tail probabilities.

- `mcor,beam-method`:
return marginal correlation matrix (scaled posterior expectation of the covariance matrix).

- `pcor,beam-method`:
return partial correlation matrix (scaled posterior expectation of the inverse covariance matrix).

- `plotML,beam-method`:
plot log-marginal likelihood of the Gaussian conjugate model as a function of shrinkage parameter.

- `plotCor,beam-method`:
plot heatmap of marginal (upper triangle) and/or partial (lower triangle) correlation estimates.

The function `beam.select` takes as input an object of class `beam-class` and carries out edge selection with multiple testing and error control (false discovery rate, family-wise error rate, ...). The function returns an (S4) object of class `beam.select-class` that is associated with the following methods:

- `summary,beam.select-method`:
provides a summary of inferred (marginal or conditional) associations.

- `marg,beam.select-method`:
returns a data.frame with marginal correlations, Bayes factors and tail probabilities for selected edges.

- `cond,beam.select-method`:
returns a data.frame with partial correlations, Bayes factors and tail probabilities for selected edges.

- `bgraph,beam.select-method`:
return an `igraph` object containing the marginal (in)dependence graph.
- `ugraph,beam.select-method`:
return an `igraph` object containing the conditional (in)dependence graph.

Author(s)

Authors: Gwenael G.R. Leday and Ilaria Speranza

Maintainer: Gwenael G.R. Leday <gwenael.leday@mrc-bsu.cam.ac.uk>

References

Leday, G.G.R. and Richardson, S. (2018). Fast Bayesian inference in large Gaussian graphical models. Submitted.

`beam` *Bayesian inference for high-dimensional Gaussian graphical models*

Description

This function carries out covariance and inverse covariance estimation within the Gaussian conjugate model. The scale matrix parameter of the inverse-Wishart is set to the identity, whereas the degree of freedom parameter is estimated by marginal likelihood maximization (empirical Bayes). The function also computes the Bayes factor and tail probability (p-values) to test the marginal or conditional independence between all pairs of variables.

Usage

```
beam(X, type = "conditional", return.only = c("cor", "BF", "prob"), verbose=TRUE)
```

Arguments

`X`	n by p data matrix
`type`	character. Either "marginal", "conditional" or "both". See Details.
`return.only`	character. Either "cor", "BF", "prob". See details.
`verbose`	logical. Whether information on progress should be be printed.

Details

The arguments `type` and `return.only` have essentially been introduced for computational and memory savings. Using argument `type` the user may indicate whether the marginal dependencies ("marginal"), the conditional dependencies ("conditional") or both ("both") are to be inferred. On the other hand, the argument `return.only` is used to indicate whether the correlations ("cor"), Bayes factors ("BF") or tail probabilities ("prob") should be returned. Default is to return all three quantities both for marginal and conditional dependencies.

Value

An object of class beam-class

Author(s)

Gwenael G.R. Leday and Ilaria Speranza

References

Leday, G.G.R. and Richardson, S. (2018). Fast Bayesian inference in large Gaussian graphical models. Submitted.

Examples

```
# Load data
data(TCPAprad)

# beam
fit <- beam(X = TCPAprad, type="both")

# Print summary
summary(fit)

# Extract matrix of marginal correlations
mcor(fit)[1:5, 1:5]

# Extract matrix of partial correlations
pcor(fit)[1:5, 1:5]

# Plot log-marginal likelihood of the Gaussian conjugate model
plotML(fit)

# Plot heatmap of marginal (upper triangle) and/or
# partial (lower triangle) correlation estimates
plotCor(fit)
```

---

beam-class *Class beam*

Description

An S4 class representing the output of the beam function.

Usage

```
  ## S4 method for signature 'beam'
print(x, ...)

  ## S4 method for signature 'beam'
show(object)

  ## S4 method for signature 'beam'
summary(object, ...)

  ## S4 method for signature 'beam'
marg(object)

  ## S4 method for signature 'beam'
cond(object)

  ## S4 method for signature 'beam'
mcor(object)

  ## S4 method for signature 'beam'
pcor(object)

  ## S4 method for signature 'beam'
plotML(object, ...)

  ## S4 method for signature 'beam'
plotCor(object, type = object@type, order = 'original', by = "marginal")

  ## S4 method for signature 'beam'
bgraph(object)

  ## S4 method for signature 'beam'
ugraph(object)
```

Arguments

`x`	An object of class `beam-class`
`object`	An object of class `beam-class`
`type`	character. Type of correlation to be displayed (marginal, conditional or both)
`order`	character. Either 'original' or 'clust'. If 'clust' the rows and columns of the correlation matrix are reordered using the cluster memberships obtained by the Louvain clustering algorithm.
`by`	character. When type ="both" and order = 'clust', specifies whether the clustering has to be performed using the complete weighted marginal or conditional independence graph.
`...`	further arguments passed to or from other methods.

Slots

`table` dat.frame. A data.frame containing marginal and/or partial correlation estimates, Bayes factors and tail probabilities for each edge.

`deltaOpt` numeric. Empirical Bayes estimate of hyperpaprameter delta.

`alphaOpt` numeric. Empirical Bayes estimate of hyperpaprameter alpha.

`dimX` numeric. Dimension of the input data matrix X.

`type` character. Input argument.)

`varlabs` character. Column labels of X.

`gridAlpha` matrix. A matrix containing the log-marginal likelihood of the Gaussian conjugate model as a function of a grid of values of alpha and delta.

`valOpt` numeric. Maximum value of the log-marginal likelihood of the Gaussian conjugate model.

`return.only` character. Input argument.

`time` numeric. Running time (in seconds).

Author(s)

Gwenael G.R. Leday and Ilaria Speranza

`beam.select` *Edge selection with multiple testing and error control*

Description

Bayesian inference of graphical structures in high-dimensional settings

Usage

```
beam.select(object, thres = 0.1, method = "BH", p0 = NULL,
return.only = c(object@return.only, "adj"))
```

Arguments

`object`	An object of class `beam-class`.
`thres`	numeric. Threshold to be applied on adjusted tail probabilities.
`method`	character. Method to use for multiple comparison adjustment of tail probabilities.
`p0`	numeric. Prior probability on the number of null hypotheses.
`return.only`	character. Quantities to be returned.

Details

The argument `method` allows to adjust the tail probabilities obtained from the null distributions of the Bayes factors for multiple comparisons. Possible choices are: "holm", "bonferroni", "BH", "BY", "HC", "blfdr" and "BFDR". Apart from "HC", "blfdr" and "BFDR", these are passed onto the R function `p.adjust` from package **stats** and we refer the user to its documentation for details. The method "HC" provides an optimal decision threshold based on the Higher Criticism score which is computed using the R function `hc.thresh` from package **fdrtool**. Again, we refer to the associated documentation for details. Methods "blfdr" and "BFDR" only work when a prior probability on the number of null hypotheses is specified via argument `p0`.

The argument `return.only` allows to decide which quantities have to be in the output: it could be any subvector of c('cor', 'BF', 'prob', 'adj') (provided that the requested quantities have been computed in the beam object, except for adjusted probabilities). It can also be set to NULL: in this case, only the selected edges will be returned without any additional information. The default value for this argument are the columns present in the beam object plus the adjusted probabilities.

Value

An object of class `beam.select-class`

Author(s)

Gwenael G.R. Leday and Ilaria Speranza

References

Leday, G.G.R. and Richardson, S. (2018). Fast Bayesian inference in large Gaussian graphical models. Submitted.

`beam.select-class` *Class beam.select*

Description

An S4 class representing the output of the `beam.select` function.

Usage

```
  ## S4 method for signature 'beam.select'
print(x, ...)

  ## S4 method for signature 'beam.select'
show(object)

  ## S4 method for signature 'beam.select'
summary(object, ...)

  ## S4 method for signature 'beam.select'
```

```
marg(object)

  ## S4 method for signature 'beam.select'
cond(object)

  ## S4 method for signature 'beam.select'
mcor(object)

  ## S4 method for signature 'beam.select'
pcor(object)

  ## S4 method for signature 'beam.select'
plotML(object, ...)

  ## S4 method for signature 'beam.select'
plotAdj(object, type=object@type, order = "original")

  ## S4 method for signature 'beam.select'
bgraph(object)

  ## S4 method for signature 'beam.select'
ugraph(object)
```

Arguments

`x`	An object of class `beam.select-class`
`object`	An object of class `beam.select-class`
`type`	character. Type of correlation to be displayed (marginal, conditional or both)
`order`	character. Either 'original' or 'clust'. If 'clust' the rows and columns of the adjacency matrix are reordered using the cluster memberships obtained by the Louvain clustering algorithm.
`...`	further arguments passed to or from other methods.

Slots

`marginal` data.frame. A data.frame containing the marginal correlation estimates, Bayes factors and tail probabilities for the selected edges only.

`conditional` data.frame. A data.frame containing the partial correlation estimates, Bayes factors and tail probabilities for the selected edges only.

`dimX` numeric. Dimension of the imput data matrix X.

`type` character. Input type (marginal, conditional or both)

`varlabs` character. Column labels of X.

`alphaOpt` numeric. Empirical Bayes estimates of hyperpaprameter alpha.

`gridAlpha` matrix. A matrix containing the log-marginal likelihood of the Gaussian conjugate model as a function of a grid of values of alpha and delta.

`valOpt` numeric. Maximum value of the log-marginal likelihood of the Gaussian conjugate model

`method` character. Input method.

`thres` numeric. Input threshold

Author(s)

Gwenael G.R. Leday and Ilaria Speranza

`TCPAprad` *Protein expression data.*

Description

Level 3 normalized proteomic data (v3.0) from The Cancer Proteome Alas (http://tcpaportal.org/tcpa). The data comprise the measurements of 189 antibodies obtained from 164 tumor tissue samples (prostate adenocarcinoma) using reverse phase protein arrays (RPPA).

Usage

```
TCPAprad
```

Format

A 164 by 189 matrix

Source

The Cancer Proteome Alas (http://tcpaportal.org/tcpa)

References

Li J, Lu Y, Akbani R, Ju Z, Roebuck PL, Liu W, Yang J-Y, Broom BM, Verhaak RGW, Kane DW, Wakefield C, Weinstein JN, Mills GB, Liang H. (2013). TCPA: A Resource for Cancer Functional Proteomics Data. *Nature Methods* 10(11), 1046-1047.

Examples

```
data(TCPAprad)
dim(TCPAprad)
TCPAprad[1:5, 1:5]
```

Appendix B

Code

B.1 R package 'beamDiff'

beamDiff-class

`beamDiff-class` is the class returned in output by `beamDiff` function. Many similarities can be found with `beam-class` in `beam` R package.

```
setClass("beamDiff",
         representation(tableList = "list",
                        invCovStdev = "matrix",
                        deltaOpt = "numeric",
                        alphasOpt = "numeric",
                        nk = "numeric",
                        logDetVect = "numeric",
                        p = "numeric",
                        type = "character",
                        varlabs = "character",
                        gridDelta = "matrix",
                        valOpt = "numeric",
                        return.only = "character",
                        numGroups = "numeric",
                        time = "numeric")
)
```

beamDiff

`beamDiff` is the function performing joint inference of marginal and partial correlation matrices.
Bayes factors and tail probabilities are also returned. We refer to 4.1 for further details.

```
##################### HELPER FUNCTIONS #####################
.scaleData <- function(X,n){
  X <- scale(X, center = TRUE, scale = FALSE)
  X <- scale(X, center = TRUE, scale = sqrt(colSums(X^2)/n))
  return(X)
}

.computeEigValVec <- function(X, S, n, p){
  if(n>=p){
    resEigen <- eigen(S, only.values=FALSE) # both eigvals
        and eigvect
  }else{
    resEigen <- eigen(tcrossprod(X), only.values=FALSE)
  }
  return(resEigen)
}

.computeEigvals <- function(eigObj, n, p){
  if(n>=p){
    eigs <- eigObj$val
  }else{
    eigs <- c(eigObj$val, rep(0,p-length(eigObj$val)))
  }
}

.lpvarGamma <- function(z, p){# functions written by Harry
    Gray
  position = 1:p
  if (length(z) == 1){
    ans <- ((p * (p-1))/4) * log(pi) + (sum(lgamma(z + (1-
        position)/2)))
  }else{
    ans <- rep(0, length(z))
    for (i in 1:length(z)){
      ans[i] <- ((p * (p-1))/4) * log(pi) + (sum(lgamma(z[i]
          + (1-position)/2)))
    }
  }
  return(ans)
}

.alphaToLambda <- function(alpha, myn, myp){
  (alpha*myn+(1-alpha)*myp+(1-alpha))/(1-alpha)
}

.lambdaToAlpha <- function(lambda, myn, myp){
  (lambda-myp-1)/(myn+lambda-myp-1)
}
```

```
.logDeterminant <- function(eigvals, p, delta){
  eigvals = eigvals + delta - p - 1
  logEigvals = log(eigvals)
  return(sum(logEigvals))
}

.sumLogML <- function(delta, p, nk, eigvalList){
  logML <- function(mydelta, myp, myn, myeigs){
    part1 <- -0.5*myn*myp*log(pi) + .lpvarGamma((mydelta+myn)
       *0.5, p=myp) - .lpvarGamma(mydelta*0.5, p=myp)
    part2 <- 0.5*mydelta*myp*log(mydelta-myp-1) - 0.5*(
       mydelta+myn)*sum(log((mydelta-myp-1)+myeigs))
    part1 + part2
  }

  logML_group <- Map(logML, delta, p, nk, eigvalList)
  Reduce(sum, logML_group)
}

.upperTriIdxs <- function(p){
  z <- sequence(p)
  cbind(
    row = unlist(lapply(1:(p-1), function(x) 1:x), use.names
       = FALSE),
    col = rep(z[-1], times = tail(z, -1)-1))
}

.approxBeta <- function(u, shp1, shp2, h=.00001){
  x <- seq(from = 0, to = 1, by = h)
  approx(x, pbeta(x, shape1=shp1, shape2=shp2, lower.tail =
     FALSE), yleft = 0, yright = 1, xout = u)$y
}

#######
# This is the core function which will be applied to each
   group, returning the table with the quantities requested
   in output

.Estimation <- function(n, p, X, S, deltaOpt, resEigen,
   return.only, type, verbose, approx.tail, k){

  cat('### Estimating group', k,'###\n')

  # Initialize data.frame with results (so that R already
     knows how much space we are going to use)
  nrows = p*(p-1)/2
  ncols <- length(return.only)
  if(type == 'both'){
    # results <- as.data.frame(matrix(NA_real_, nrow = nrows,
        ncol = 2*ncols))
    results <- matrix(NA_real_, nrow = nrows, ncol = 2*ncols)
  } else{
    # results <- as.data.frame(matrix(NA_real_, nrow = nrows,
        ncol = ncols))
    results <- matrix(NA_real_, nrow = nrows, ncol = ncols)
  }

  # temporary colnames
  colnames(results) <- rep('tempName', ncol(results))

  # Initialize Hdiag list (it will remain empty if only
     marginal evaluation is requested)
  HinvStdev <- list()

  current_free_column <- 1

  # Store in a vector only the upper triangular part of S
  matidxs <- .upperTriIdxs(p)
  Svec <- S[matidxs]

  ### MARGINAL DEPENDENCIES  ###

  # Diagonal prior specific case (M <- (deltaOpt-p-1)*diag(p)
     )
  Hvec <- Svec # All the off diag elements of M are null. On
     the diagonal we'll have n + (deltaopt-p-1)

  if( type=="both" | type=="marginal" ){

    if(verbose){
      cat("Marginal dependencies:\n")
    }

    if(any( c("cor", "BF") %in% return.only)){

      if(verbose){
        cat("  > compute marginal correlation estimates...")
      }

      # Marginal correlation estimates
      rhij <- Hvec/(deltaOpt+n-p-1)

      if("cor" %in% return.only){

        # Update results table with marginal correlation
        colnames(results)[current_free_column] <- 'm_cor'  #
            change col_name
      # results$m_cor <- rhij   # fill column
      results[,'m_cor'] <- rhij
      current_free_column = current_free_column + 1  #
         increment index of free column
    }

    if(verbose){
      cat("DONE \n")
    }

    if("BF" %in% return.only){

      if(verbose){
        cat("  > Bayes factors...")
      }

      # part 1: log Gamma functions
      part1 <- .lpvarGamma((deltaOpt+n-p+2)/2, p=2) - .
         lpvarGamma((deltaOpt-p+2)/2, p=2)
      part1 <- part1 + 2*lgamma((deltaOpt-p+3)/2) - 2*
         lgamma((deltaOpt+n-p+3)/2)

      # part 2: log ratio prior/posterior marginal
         correlations
      part2 <- - ((deltaOpt+n-p+2)/2)*log(1-rhij^2)

      # log-BFs
      logBFs <- part1 + part2

      # Update results table with marginal log bayes
         factors
      colnames(results)[current_free_column] <- 'm_logBF'
         # change col_name
      #results$m_logBF <- logBFs   # fill column
      results[,'m_logBF'] <- logBFs
      current_free_column = current_free_column + 1  #
         increment index of free column

      # Relieve memory
      rm(logBFs, part1, part2)

      if(verbose){
        cat("DONE \n")
      }

    }

    # Relieve memory
    rm(rhij)
    }

    if("prob" %in% return.only){

      if(verbose){
        cat("   > compute tail probabilities...")
      }

      # Sample correlations
      rsij <- Svec/n

      # Update table
      colnames(results)[current_free_column] <- 'm_tail_prob'
           # change col_name

      if(approx.tail == FALSE){ # "exact" pbeta function
        # results$m_tail_prob <- pbeta(rsij^2, 1/2, (n-1)/2,
           lower.tail=FALSE)   # fill the free column
        results[,'m_tail_prob'] <- pbeta(rsij^2, 1/2, (n-1)/
           2, lower.tail=FALSE)
      }else{ # approx pbeta function
        # results$m_tail_prob <- .approxBeta(rsij^2, 1/2, (n
           -1)/2)   # fill the free column
        results[,'m_tail_prob'] <- .approxBeta(rsij^2, 1/2, (
           n-1)/2)
      }
      current_free_column = current_free_column + 1  #
         increment index of free column

      # Relieve memory
      rm(rsij)

    }

    if(verbose){
      cat("DONE \n")
    }

  }

  # Relieve memory
  rm(Svec, Hvec)


  ### CONDITIONAL DEPENDENCIES ###

  # -> Compute partial correlations, (scaled) log-Bayes
     factors and tail probabilities
```

```
if( type=="both" | type=="conditional" ){

  if(verbose){
    cat("Conditional dependencies:\n")
    cat("  > compute partial correlation estimates...")
  }

  # Compute scaled posterior expectation of the inverse
     covariance matrix (partial correlation estimates)
  if(n>=p){
    H <- (deltaOpt-p-1)*diag(p) + S
    Hinv <- solve(H)
    rm(H) # relieve memory
  }else{
    prodtp <- crossprod(X, resEigen$vectors)
    Hinv <- tcrossprod(prodtp,diag(1/(1+(1/(deltaOpt-p-1))*
       resEigen$val)))
    Hinv <- tcrossprod( Hinv, prodtp)/((deltaOpt-p-1)^2)
    Hinv <- (1/(deltaOpt-p-1))*diag(p) - Hinv
    #Hinv <- (1/(deltaOpt-p-1))*diag(p) - (1/(deltaOpt-p-1)
       ^2)*t(X)%*%solve(diag(n)+(1/(deltaOpt-p-1))*
       tcrossprod(X))%*%X

    # Relieve memory
    rm(prodtp)

  }

  # Relieve memory
  rm(S)

  # Partial correlation estimates
  HinvStdev <- sqrt(diag(Hinv))
  rkij <- - cov2cor(Hinv)
  rkij <- rkij[matidxs]  # keep only the upper triangular
     part, discard the rest

  #cat(Hinv)

  if("cor" %in% return.only){
    colnames(results)[current_free_column] <- 'p_cor'  #
       change col_name
    # results$p_cor <- rkij  # fill the free column
    results[,'p_cor'] <- rkij
    current_free_column = current_free_column + 1  #
       increment index of free column
  }
```

```
    if(verbose){
      cat("DONE \n")
    }

    # Compute scaled log-Bayes factors
    if("BF" %in% return.only){

      if(verbose){
        cat("  > compute Bayes factors...")
      }

      # part 1: log Gamma functions
      part1 <- lgamma((deltaOpt+n)/2) - lgamma(deltaOpt/2)
      part1 <- part1 + lgamma((deltaOpt+n-1)/2) - lgamma((
          deltaOpt-1)/2)
      part1 <- part1 + 2*lgamma((deltaOpt+1)/2) - 2*lgamma((
          deltaOpt+n+1)/2)

      # part 2: log ratio prior/posterior marginal
          correlations
      part2 <- - ((deltaOpt+n)/2)*log(1-rkij^2)

      # Update table with log-BFs
      colnames(results)[current_free_column] <- 'p_logBF'  #
          change col_name
      # results$p_logBF <- part1 + part2
      results[,'p_logBF'] <- part1 + part2
      current_free_column = current_free_column + 1  #
          increment index of free column

      # Relieve memory
      rm(part1, part2)

      if(verbose){
        cat("DONE \n")
      }

    }

    if("prob" %in% return.only){

      if(verbose){
        cat("  > compute tail probabilities...")
      }

      gii2gjj2 <- (deltaOpt-p-1)^2
      diagHinv <- diag(Hinv)

      kii2kjj2 <- tcrossprod(1/diagHinv, 1/diagHinv)
      kii2kjj2 <- kii2kjj2[matidxs]
      kii2kjj2 <- kii2kjj2/((1-rkij^2)^2)
      Gii2Gjj2 <- tcrossprod(diagHinv,diagHinv)
      Gii2Gjj2 <- Gii2Gjj2[matidxs]
      tpdiag1 <- diagHinv[matidxs[,1]]
      tpdiag2 <- diagHinv[matidxs[,2]]
      Gii2plusGjj2 <- tpdiag1 + tpdiag2

      # Relieve memory
      rm(tpdiag1, tpdiag2, diagHinv, matidxs)

      # Compute rfij
      # rfij <- (sqrt(kii2kjj2)*tableC[,3])/sqrt(kii2kjj2 +
         gii2gjj2 - (deltaOpt-p-1)* ( Gii2plusGjj2/(Gii2Gjj2*
         (1-tableC[,3]^2)) ))
      num <- sqrt(kii2kjj2)
      num <- num*rkij
      denom <- 1-rkij^2
      denom <- denom*Gii2Gjj2
      denom <- - (deltaOpt-p-1)*(Gii2plusGjj2/denom)
      denom <- denom + kii2kjj2
      denom <- denom + gii2gjj2
      rfij <- num/sqrt(denom)

      # Relieve memory
      rm(Gii2plusGjj2, Gii2Gjj2, kii2kjj2, gii2gjj2)

      # Update table
      rfij2 <- rfij^2

      colnames(results)[current_free_column] <- 'p_tail_prob'
           # change col_name
      if(approx.tail == FALSE){
        # results$p_tail_prob <- pbeta(rfij2, 1/2, (n-1)/2,
           lower.tail=FALSE)
        results[,'p_tail_prob'] <- pbeta(rfij2, 1/2, (n-1)/2,
            lower.tail=FALSE)
      }else{
        # results$p_tail_prob <- .approxBeta(rfij2, 1/2, (n
           -1)/2)
        results[,'p_tail_prob'] <- .approxBeta(rfij2, 1/2, (n
           -1)/2)
      }

      # Relieve memory
      rm(rfij, rfij2)

      if(verbose){
        cat("DONE \n")
      }

    }

    # Relieve memory
    rm(Hinv, rkij)

  }
  return(list(res = results, HinvStdev = HinvStdev))
}

#################### MAIN FUNCTION ####################

beamDiff <- function(..., type = "both", return.only = c("cor
   ", "BF", "prob"), approx.tail = FALSE, verbose=TRUE){

 time0 <- proc.time()

 ##########################################
 #              PREPROCESSING             #
 ##########################################

 if(verbose){
    cat("Preprocessing... ")
 }

 # check arguments
 matList <- list(...)
 if(length(matList) == 1 & is.list(matList[[1]])){ # this
    allows the user to pass as input both a list of matrices
    or the matrices separated by a comma
   matList = matList[[1]]
 }

 numGroups <- length(matList)
 if(numGroups == 0){
   stop("No data provided.")
 }
 if(any(sapply(matList, is.matrix)) == FALSE){
   stop("One or more input data are not in matrix form.")
 }
 if(any(sapply(matList, function(X) any(is.na(X))))){
   stop("One or more matrices contain missing values.")
 }

 pk = sapply(matList, ncol)
 p = unique(pk)
 if(length(p) > 1){
   stop("All the data matrices must have the same number of
      features.")
}

X1 <- matList[[1]]
if(!is.null(colnames(X1))){
  varlabs <- colnames(X1)
}else{
  varlabs <- character()
}
rm(X1)

if(is.character(type)){
  if(length(type)==1){
    if(!type%in%c("both", "marginal", "conditional")){
      stop("type is not recognized")
    }
  }else{
    stop("type must be a character of length equal to 1")
  }
}else{
  stop("type must be a character")
}
if(is.character(return.only)){
  if(length(return.only)%in%c(1:3)){
      ind.return.only <- return.only%in%c("cor", "BF", "prob"
         )
    if(any(!ind.return.only)){
      stop("return.only contains characters that are not
         recognized")
    }
  }else{
    stop("return.only must contain at least 1 element and 3
       elements at most")
  }
}else{
  stop("return.only must be a character")
}

# Dimension data
nk <- sapply(matList, nrow)

if(p >= 2000){
  approx.tail = TRUE
}

# Standardize data
matList <- Map(.scaleData, matList, nk)

# Cross-product
```

```
SList <- lapply(matList, crossprod)

# For each matrix compute eigvals and eigvect, according to
   the more efficient method

# for each group eigvals&eigvects
eigList <- Map(.computeEigValVec, matList, SList, nk, p)
eigvalList <- Map(.computeEigvals, eigList, nk, p) # extract
    eigvals and fill with 0 the remaining elements if n < p

if(verbose){
  cat("DONE \n")
}

###########################################
#            OPTIMAL SHRINKAGE            #
###########################################

if(verbose){
  cat("Optimal shrinkage... ")
}

# Range of values
eps <- 10^(-6)
lowerVal <- min(.alphaToLambda(alpha=eps, myn=nk, myp=p))
upperVal <- max(.alphaToLambda(alpha=1-eps, myn=nk, myp=p))
initVal <- mean(.alphaToLambda(alpha=0.5, myn=nk, myp=p))
# Maximize log-marginal likelihood
resOpt <- optim(par=initVal, fn=.sumLogML, p = p, nk=nk,
                eigvalList= eigvalList,
                method='Brent', lower=lowerVal, upper=
                   upperVal, control = list(fnscale=-1))

# Optimum
deltaOpt <- resOpt$par
alphasOpt <- .lambdaToAlpha(lambda=deltaOpt, myn=nk, myp=p)
   # Vector with optimal alphas (group-specific, depending
   on nk)

myalphas <- seq(from=quantile(alphasOpt, probs=0.2), to=
   quantile(alphasOpt, probs=0.8), length=100)
mydeltas <- as.vector(sapply(myalphas, .alphaToLambda, myn=
   nk, myp=p)) # delta values corresponding to the optimal
   alpha values
deltaSeq <- seq(from=min(mydeltas), to=max(mydeltas), length
   =100)
allSumMLs <- sapply(deltaSeq, .sumLogML, p = p, nk = nk,
   eigvalList = eigvalList)
```

```
gridDelta <- cbind(deltaSeq, allSumMLs)
colnames(gridDelta) <- c("delta", "sum-logML")
if(verbose){
  cat("DONE \n")
}

## compute logdeterminant of posterior scaling matrix of
   covariance
logDetVect <- sapply(eigvalList, .logDeterminant, p = p,
   delta = deltaOpt)

#############################
#        ESTIMATION         #
#############################
results <- Map(.Estimation, nk, p, matList, SList, deltaOpt,
    eigList, rep(list(return.only), numGroups), type,
   verbose, approx.tail,1:numGroups)
tableList <- lapply(results, function(X) X$res) # extract
   the matrices with results
invCovStdev <- sapply(results, function(X) X$HinvStdev) #
   extract the vectors with diag elements of H

time1 <- proc.time() - time0

#########################
#        OUTPUT         #
#########################
# List
out <- new("beamDiff",
           "tableList" = tableList,
           "invCovStdev" = invCovStdev,
           "deltaOpt" = deltaOpt,
           "alphasOpt"= alphasOpt,
           "logDetVect" = logDetVect,
           "nk" = nk,
           "p" = p,
           "type"= type,
           "varlabs" = varlabs,
           "gridDelta" = gridDelta,
           "valOpt" = resOpt$value,
           "return.only" = return.only,
           "numGroups" = numGroups,
           "time" = time1[3])
return(out)
}
```

crossDistance

`crossDistance` function returns a list with marginal cross entropy, conditional cross entropy and KL divergence. More details on these quantities can be found in Section 3.4.

```
setMethod(
  f = "crossDistance",
  signature = "beamDiff",
  definition = function(object, symmetric = FALSE){

    ##### helper functions #####
    lpvarGamma <- function(z, p = object@p){
      return(p*(p-1)/4*log(pi) + sum(lgamma(z + (1 - 1:p)/2))
          )
    }

    pvarDiGamma <- function(z,p = object@p){
      return(sum(digamma(z + (1 - 1:p)/2)))
    }

    pcor2pcov <- function(Cormat, sd){ #ok
      scalingMat <- sd %*% t(sd)
      Cormat <- - Cormat
      diag(Cormat) <- 1

      Covmat <- scalingMat * Cormat # element-wise product
      return(Covmat)
    }

    mcor2mcov <- function(MargMat, scalingFact){ #ok
      return(scalingFact * MargMat)
    }

    tr <- function(X){ # trace of a matrix # ok
      sum(diag(X))
    }

    margEntropy <- function(delta, logdet, p = object@p){
      return(lpvarGamma(delta/2) + (p+1)/2*logdet - (delta+p
          +1)/2*pvarDiGamma(delta/2) - p*(p+1)/2*log(2) +
          delta*p/2)
    }

    condEntropy <- function(delta, logdet, p = object@p){
      return(lpvarGamma(delta/2) - (p+1)/2*logdet - (delta-p
          -1)/2*pvarDiGamma(delta/2) + p*(p+1)/2*log(2) +
          delta*p/2)
    }
```

```
ComputeKLdiv <- function(delta1, delta2, logdet1, logdet2
    , invM1, M2, p = object@p){ # ok
  trace12 <- tr(invM1 %*% M2)
  KL12 <- delta2/2*(logdet1 - logdet2) + lpvarGamma(
      delta2/2) - lpvarGamma(delta1/2) + (delta1 - delta2)
      /2 * pvarDiGamma(delta1/2) +
    delta1/2 * trace12 - delta1 * p/2
  return(KL12)
}

############################
numGr <- object@numGroups
nk <- object@nk
logdetVect <- object@logDetVect
deltaOpt <- object@deltaOpt
p <- object@p
invCovStdev <- object@invCovStdev
objType <- object@type

KLdiv <- matrix(0, nrow = numGr, ncol = numGr) #
   initialize KL divergence matrix

if(objType == 'both'){ # in object both marginal and
   partial correlation estimations are stored. No need to
    compute any inverse
  corList <- lapply(1:numGr, mcor, object = object)
  MList <- Map(mcor2mcov, MargMat = corList, scalingFact
      = nk + deltaOpt - p - 1)

  for(i in 1:numGr){
    inv_M_i <- pcor2pcov(pcor(object, k = i), invCovStdev
        [,i]) # the inverse of M_i is (M+S)^{-1}, i.e. the
         elements of the inverse partial correlation
        scaling matrix
    KLdiv[i,-i] <- unlist(Map(ComputeKLdiv, # fill the i-
        th row (except the diag elem which is 0) of the KL
        -div matrix
                              delta1 = deltaOpt + nk[i],
                              delta2 = deltaOpt + nk[-i],
                              logdet1 = logdetVect[i],
                              logdet2 = logdetVect[-i],
                              invM1 = rep(list(inv_M_i),
                                 numGr-1),
                              M2 = MList[-i]
    ))
  }
}
if(objType == 'marginal'){ # only marginal correlation is
```

```
        stored. No estimation of scaling matrix of precision
       matrix. Inversion needed
      corList <- lapply(1:numGr, mcor, object = object)
      MList <- Map(mcor2mcov, MargMat = corList, scalingFact
         = nk + deltaOpt - p - 1)

      for(i in 1:numGr){
        inv_M_i <- solve(MList[[i]]) # estimation of the
           inverse partial correlation scaling matrix
        KLdiv[i,-i] <- unlist(Map(ComputeKLdiv, # fill the i-
           th row (except the diag elem which is 0) of the KL
           -div matrix
                                  delta1 = deltaOpt + nk[i],
                                  delta2 = deltaOpt + nk[-i],
                                  logdet1 = logdetVect[i],
                                  logdet2 = logdetVect[-i],
                                  invM1 = rep(list(inv_M_i),
                                     numGr-1),
                                  M2 = MList[-i]
        ))
      }
    }
    if(objType == 'conditional'){ # only conditional
       correlation is stored. No estimation of scaling matrix
        of covariance matrix. Inversion needed
      pcorList <- lapply(1:numGr, pcor, object = object)
      MinvList <- Map(pcor2pcov, pcorList, split(invCovStdev,
          col(invCovStdev))) # estimations of M_{i}^{-1}. #
         split() converts the matrix in list of its columns

      for(j in 1:numGr){ # now we fill KL matrix by columns
  # estimation of the inverse partial correlation scaling
      matrix
        M_j <- solve(MinvList[[j]])
   # fill the i-th row (except the diag elem which is 0) of
       the KL-div matrix
        KLdiv[-j,j]<-unlist(Map(ComputeKLdiv,
                                delta1 = deltaOpt + nk[-j],
                                delta2 = deltaOpt + nk[j],
                                logdet1 = logdetVect[-j],
                                logdet2 = logdetVect[j],
                                invM1 = MinvList[-j],
                                M2 = rep(list(M_j), numGr-1)
        ))
      }
    }

    margEntropyList <- Map(margEntropy, delta = nk + deltaOpt
       , logdet = logdetVect) # marginal entropy
    margCrossEntropy <- KLdiv + unlist(margEntropyList) #
       compute the cross-entropy MargCrossEntr[i,j] = KL[i,j]
        + MargEntr[i]


    condEntropyList <- Map(condEntropy, delta = nk + deltaOpt
       , logdet = logdetVect) # conditional entropy
    condCrossEntropy <- KLdiv + unlist(condEntropyList) #
       CondCrossEntr[i,j] = KL[i,j] + CondEntr[i]

    result <- list(KLdivergence = KLdiv, marginalCrossEntropy
        = margCrossEntropy, partialCrossEntropy =
       condCrossEntropy)

    if(symmetric){
      result <- lapply(result, function(X) X+t(X))   # return
          the sum of K[i,j] and K[j,i] for each (i,j)
    }

    return(result)
  }
)
```

B.2 Hyperparameters elicitation

beamDiff_boost

This function jointly estimates marginal and partial correlation matrices, using the EM algorithm by Bilgrau et al. [2015] and described in Section 3.3 to set the hyperparameters.

```
##################### HELPER FUNCTIONS #####################
# function written by Harry Gray
.lpvarGamma <- function(z, p){
 position = 1:p
 if (length(z) == 1){
  ans <- ((p * (p-1))/4) * log(pi) + (sum(lgamma(z + (1-
      position)/2)))
 }else{
  ans <- rep(0, length(z))
  for (i in 1:length(z)){
   ans[i] <- ((p * (p-1))/4) * log(pi) + (sum(lgamma(z[i] +
       (1-position)/2)))
  }
 }
 return(ans)
}

.upperTriIdxs <- function(p){
 z <- sequence(p)
 cbind(row = unlist(lapply(1:(p-1), function(x) 1:x), use.
    names = FALSE), col = rep(z[-1], times = tail(z, -1)-1))
}

.alphaToLambda <- function(alpha, myn, myp){
 (alpha*myn+(1-alpha)*myp+(1-alpha))/(1-alpha)
}

.lambdaToAlpha <- function(lambda, myn, myp){
 (lambda-myp-1)/(myn+lambda-myp-1)
}

.getLogDets <- function(mySlist, myM){
 myfun <- function(ii, theS, theM){
  if(ii==0){
   return(determinant(myM)$modulus)
  }else{
   return(determinant(theM + theS[[ii]])$modulus)
  }
 }
 sapply(0:length(mySlist), myfun, theS=mySlist, theM=myM,
 simplify=TRUE)
}

.sumLogML <- function(mydelta, myn, myp, myM, mySlist){
 part1 <- -0.5*myp*log(pi)*sum(myn)
 part2 <- sum(.lpvarGamma((mydelta+myn)*0.5, p=myp)) - length
    (myn)*.lpvarGamma(mydelta*0.5, p=myp)
 part3 <- 0.5*length(myn)*mydelta*determinant(myM, logarithm
    = TRUE)$modulus
 myfun <- function(ii, thedelta, then, theS, theM){
  0.5*(then[ii]+thedelta)*determinant(theM + theS[[ii]],
  logarithm = TRUE)$modulus
 }
 part4 <- sum(sapply(1:length(myn), myfun, thedelta=mydelta,
    then=myn, theS=mySlist, theM=myM, simplify=TRUE))
 part1 + part2 + part3 - part4
}

# same as sumLogML but optimized when M does not change
# (LogDets are computed once for all)
.sumLogML2 <- function(mydelta,myn,myp,myM,mySlist,myLogDets)
   {
 part1 <- -0.5*myp*log(pi)*sum(myn)
 part2 <- sum(.lpvarGamma((mydelta+myn)*0.5, p=myp)) -
 length(myn)*.lpvarGamma(mydelta*0.5, p=myp)
 part3 <- 0.5*length(myn)*mydelta*myLogDets[1]
 myfun <- function(ii, thedelta, then, theS, theM, theLogDets
    ){
  0.5*(then[ii]+thedelta)*theLogDets[ii+1]
 }
 part4 <- sum(sapply(1:length(myn), myfun, thedelta=mydelta,
    then=myn, theS=mySlist, theM=myM, theLogDets=myLogDets,
    simplify=TRUE))
 part1 + part2 + part3 - part4
}

# function written by Gwenael Leday
.joint_ML_optimization <- function(Slist, n, p, K, maxiter,
   eps_converge, plotML){
 # Update M for fixed delta
 updateM <- function(oldM, mydelta, mySlist, myn){
  myfun <- function(ii, thedelta, then, theS, theM){
   (then[ii]+thedelta)*solve(theM + theS[[ii]])
  }
  tp <- sapply(1:length(myn), myfun, thedelta=mydelta, then=
     myn, theS=mySlist, theM=oldM,  simplify=FALSE)
  solve(Reduce('+', tp))*(length(myn)*mydelta)
 }
 # Update delta for fixed M
 updateDelta <- function(myn, myp, mySlist, myM, myLogDets){
  eps <- 10^(-3)
  lowerVal <- min(.alphaToLambda(alpha=eps, myn=myn, myp=myp)
     )
  upperVal <- max(.alphaToLambda(alpha=1-eps, myn=myn, myp=
     myp))
  initVal <- mean(.alphaToLambda(alpha=0.5, myn=myn, myp=myp)
     )
  resOpt <- optim(par=initVal, fn=.sumLogML2, myp = myp,
  myn=myn, myM= myM, mySlist=mySlist,
  myLogDets=myLogDets,
  method='Brent', lower=lowerVal, upper=upperVal,
  control = list(fnscale=-1))
  resOpt$par
 }

 # Initialisation
 M_old <- M_new <- diag(p)
 allDeltas <- rep(NA, maxiter+1)
 allDeltas[1] <- sum(n)
 allLogML <- matrix(NA, maxiter, K)
 totalLogML <- rep(NA, maxiter+1)
 totalLogML[1] <- .sumLogML(mydelta=allDeltas[1], myn=n, myp=
   p, myM=M_old, mySlist=Slist)

# Algorithm - empirical Bayes
bool <- TRUE
ct <- 1
cat('iter = ')
while(bool){
 # Update M
 M_new<-updateM(M_old, mydelta = allDeltas[ct], mySlist =
    Slist, myn = n)

 # Update delta
 logDets <- .getLogDets(mySlist=Slist, myM=M_new)
 allDeltas[ct+1] <- updateDelta(myn=n, myp=p, mySlist=Slist,
     myM=M_new, myLogDets=logDets)

 # Total log-marginal likelihood
 totalLogML[ct+1] <- .sumLogML(mydelta=allDeltas[ct+1], myn=
    n,
 myp=p, myM=M_new, mySlist=Slist)

 # update M_old
 M_old <- M_new

 # Monitor convergence
 if(ct>2){
  diffLogML <- totalLogML[ct+1]-totalLogML[ct]
  relDiffLogML <- diffLogML/abs(totalLogML[ct])
  if(relDiffLogML<eps_converge){
   bool <- FALSE
  }
 }
 if(ct==maxiter){
  warning('No convergence. relDiffLogML =', relDiffLogML,'\n
     ')
  bool <- FALSE
 }
 if(bool){
  ct <- ct + 1
 }
}

if(plotML){
 plot(totalLogML[1:(ct+1)][-1], xlab="iteration", ylab="sum
    log-ML")
}

cat('\nkappa(M) =', kappa(M_new),'\n')
```

```
 # Estimate of M and delta upon convergence
 return(list("M" = M_new, "delta" = allDeltas[ct+1], "value"
     = totalLogML[ct+1]))
}

# This is the core function which will be applied to each
   group, returning the table with the quantities requested
   in output
.Estimation <- function(n, p, X, S, deltaOpt, MOpt, group_idx
   ){
 cat('GROUP', group_idx,'(',n,'obs )','\n')
 # Initialize data.frame with results (so that R already
    knows how much space we are going to use)
 nrows = p*(p-1)/2 ; ncols <- 2 # mcor and pcor
 results <- matrix(NA_real_, nrow = nrows, ncol = ncols) ;
 colnames(results) <- c('m_cor','p_cor')

 # Upper triangular indexes
 matidxs <- .upperTriIdxs(p)

 # compute posterior covariance matrix
 H <- MOpt + S
 cat('kappa(S) =', kappa(S),'\tkappa(H) =',kappa(H),'\n')

 # Marginal correlation estimates
 rhij <- cov2cor(H)[matidxs] ; results[,'m_cor'] <- rhij

 # Compute scaled posterior exp. of the inv. cov. matrix
 Hinv <- solve(H)

 # Partial correlation estimates
 rkij <- - cov2cor(Hinv) ; rkij <- rkij[matidxs]  ;
 results[,'p_cor'] <- rkij
 # store diags to retrieve the original (non-scaled) matrices
 return(list(pcor = results, diagH = diag(H),
 diagHinv = diag(Hinv)))
}


#################### MAIN FUNCTION ####################
# Joint optimization of M and delta
beamDiff_boost <- function(matList, plotML_flag = FALSE,
   maxiter = 100, eps = .001){
 numGroups <- length(matList)
 pk = sapply(matList, ncol) ; p = unique(pk)

 # Dimension data
 nk <- sapply(matList, nrow)
 # Cross-product
 SList <- lapply(matList, crossprod)

 # OPTIMAL SHRINKAGE
 resOpt <- .joint_ML_optimization(SList, nk, p, numGroups,
    maxiter, eps, plotML = plotML_flag)

 # Optimum
 deltaOpt <- resOpt$delta
 MOpt <- resOpt$M

 alphasOpt <- .lambdaToAlpha(lambda=deltaOpt, myn=nk, myp=p)

 myalphas <- seq(from=quantile(alphasOpt, probs=0.2), to=
    quantile(alphasOpt, probs=0.8), length=100)
 mydeltas <- as.vector(sapply(myalphas, .alphaToLambda, myn=
    nk, myp=p))
 deltaSeq <- seq(from=min(mydeltas),to=max(mydeltas),length
    =100)

 logDets <- .getLogDets(SList, MOpt)
 allSumMLs <- sapply(deltaSeq, .sumLogML2, myn = nk, myp = p,
 mySlist = SList, myLogDets = logDets)

 gridDelta <- cbind(deltaSeq, allSumMLs)
 colnames(gridDelta) <- c("delta", "sum-logML")

 results <- Map(.Estimation, nk, p, matList, SList, deltaOpt,
 rep(list(MOpt), numGroups), as.list(1:numGroups))
 tableList <- lapply(results, function(X) X$pcor)
 diagH_list <- lapply(results, function(X) X$diagH)
 diagHinv_list <- lapply(results, function(X) X$diagHinv)

 # List ith results
 out <- list("tableList" = tableList,
    "diagH_list" = diagH_list,
    "diagHinv_list" = diagHinv_list,
    "deltaOpt" = deltaOpt,
    "MOpt" = MOpt,
    "alphasOpt"= alphasOpt,
    "nk" = nk,
    "p" = p,
    "gridDelta" = gridDelta,
    "valOpt" = resOpt$value,
    "numGroups" = numGroups)
 return(out)
}
```

beamDiff_new

This function jointly estimates marginal and partial correlation matrices. beam is preliminary run on all data merged together and its estimate of the covariance structure in used as prior hyperparameter M. More details on the algorithm can be found in Section 3.3

```
require(beam)

##################### HELPER FUNCTIONS #####################
.scaleData <- function(X,n){
  X <- scale(X, center = TRUE, scale = FALSE)
  X <- scale(X, center = TRUE, scale = sqrt(colSums(X^2)/n))
  return(X)
}

.lpvarGamma <- function(z, p){ # function written by Harry
   Gray
  position = 1:p
  if (length(z) == 1){
    ans <- ((p * (p-1))/4) * log(pi) + (sum(lgamma(z + (1-
       position)/2)))
  }else{
    ans <- rep(0, length(z))
    for (i in 1:length(z)){
      ans[i] <- ((p * (p-1))/4) * log(pi) + (sum(lgamma(z[i]
         + (1-position)/2)))
    }
  }
  return(ans)
}

.alphaToLambda <- function(alpha, myn, myp){
  (alpha*myn+(1-alpha)*myp+(1-alpha))/(1-alpha)
}

.lambdaToAlpha <- function(lambda, myn, myp){
  (lambda-myp-1)/(myn+lambda-myp-1)
}

.sumLogML <- function(delta, p, nk, logdet_psi_hat, psi_hat,
   Slist){
  logML <- function(mydelta, myp, myn, logdet_psi_hat, psi_
     hat, S){
    part1 <- -0.5*myn*myp*log(pi) + .lpvarGamma((mydelta+myn)
       *0.5, p=myp) - .lpvarGamma(mydelta*0.5, p=myp)
    part2 <- 0.5*mydelta*myp*log(mydelta-myp-1) + 0.5*mydelta
       *logdet_psi_hat - 0.5*(mydelta+myn)*determinant((
       mydelta-p-1)*psi_hat + S, logarithm = T)$modulus
    part1 + part2
  }
  K <- length(nk)
  logML_group <- Map(logML, delta, p, nk, logdet_psi_hat, rep
     (list(psi_hat), K), Slist)
  Reduce(sum, logML_group)
}

.upperTriIdxs <- function(p){
  z <- sequence(p)
  cbind(
    row = unlist(lapply(1:(p-1), function(x) 1:x), use.names
       = FALSE),
    col = rep(z[-1], times = tail(z, -1)-1))
}

.Estimation <- function(n, p, X, S, deltaOpt, MOpt, group_idx
   ){
  cat('GROUP', group_idx,'(',n,'obs )','\n')
  # Initialize data.frame with results (so that R already
     knows how much space we are going to use)
  nrows = p*(p-1)/2 ; ncols <- 2 # mcor and pcor
  results <- matrix(NA_real_, nrow = nrows, ncol = ncols) ;
     colnames(results) <- c('m_cor','p_cor')

  # Upper triangular indexes
  matidxs <- .upperTriIdxs(p)

  # compute posterior covariance matrix
  H <- MOpt + S
  cat('kappa(S) =', kappa(S),'\tkappa(H) =',kappa(H),'\n')

  # Marginal correlation estimates
  rhij <- cov2cor(H)[matidxs] ; results[,'m_cor'] <- rhij

  # Compute scaled posterior expectation of the inverse
     covariance matrix (partial correlation estimates)
  Hinv <- solve(H)

  # Partial correlation estimates
  rkij <- - cov2cor(Hinv) ; rkij <- rkij[matidxs]  ; results
     [,'p_cor'] <- rkij

  return(results)
}

##################### MAIN FUNCTION #####################
beamDiff_new <- function(matList){
  numGroups <- length(matList)
```

```
pk = sapply(matList, ncol)
p = unique(pk)

# Dimension data
nk <- sapply(matList, nrow)

# Standardize data
matList <- Map(.scaleData, matList, nk)

# Cross-product
SList <- lapply(matList, crossprod)

allData <- do.call(rbind, matList) # put all data together
    and run beam on them
beam_fit <- beam(allData, type = 'marginal', return.only =
    'cor', verbose = F)
psi_hat <- beam::mcor(beam_fit) # get the estimation for
    the marginal correlation matrix

# now we have to compute logML and find optimal delta: pre-
    compute all quantities independent of delta
logdet_psi_hat <- determinant(psi_hat, logarithm = T)$
    modulus

# Range of values
eps <- 10^(-6)
lowerVal <- min(.alphaToLambda(alpha=eps, myn=nk, myp=p))
upperVal <- max(.alphaToLambda(alpha=1-eps, myn=nk, myp=p))
initVal <- mean(.alphaToLambda(alpha=0.5, myn=nk, myp=p))
# Maximize log-marginal likelihood
resOpt <- optim(par=initVal, fn=.sumLogML, p = p, nk=nk,
    logdet_psi_hat=logdet_psi_hat, psi_hat=psi_hat, Slist =
    SList, method='Brent', lower=lowerVal, upper=upperVal,
    control = list(fnscale=-1))

# Optimum
deltaOpt <- resOpt$par
alphasOpt <- .lambdaToAlpha(lambda=deltaOpt, myn=nk, myp=p)
     # Vector with optimal alphas (group-specific, depending
     on nk)

myalphas <- seq(from=quantile(alphasOpt, probs=0.2), to=
    quantile(alphasOpt, probs=0.8), length=100)
mydeltas <- as.vector(sapply(myalphas, .alphaToLambda, myn=
    nk, myp=p)) # delta values corresponding to the optimal
    alpha values
deltaSeq <- seq(from=min(mydeltas), to=max(mydeltas),
    length=100)
allSumMLs <- sapply(deltaSeq, .sumLogML, p = p, nk=nk,
      logdet_psi_hat=logdet_psi_hat, psi_hat=psi_hat, Slist =
      SList)

  gridDelta <- cbind(deltaSeq, allSumMLs)
  colnames(gridDelta) <- c("delta", "sum-logML")

  MOpt <- (deltaOpt-p-1)*psi_hat

  tableList <- Map(.Estimation, nk, p, matList, SList,
      deltaOpt, rep(list(MOpt), numGroups), as.list(1:
      numGroups))

  # List ith results
  out <- list("tableList" = tableList,
              "deltaOpt" = deltaOpt,
              "MOpt" = MOpt,
              "alphasOpt"= alphasOpt,
              "nk" = nk,
              "p" = p,
              "gridDelta" = gridDelta,
              "valOpt" = resOpt$value,
              "numGroups" = numGroups)
  return(out)
}
```

B.3 Data simulation

In this section we present two functions we implemented to simulate a set of precision matrices. More details about the two algorithms can be found in Section 4.2.

simBlockMatrix

```
simBlockMatrix <- function(sparsity, nBlocks, val_lb = 0, val
    _ub = 1, p, max_iter = 1000, maxCond = 500, minCorThres =
    0, minMedThres = 0, shrink = 0){
  totEdges <- p*(p-1)/2 # size of upper triang part
  blockSize <- p/nBlocks # rows and cols of each block
  totEdgesBlock <- blockSize*(blockSize-1)/2 # tot number of
      values in each block
  numEdgesBlock <- round(sparsity*nBlocks*totEdgesBlock) #
      number of non-zero values in each block
  n_iter <- 1
  bestMinEigFound <- -999 # set starting values to assess
      during the loop the "goodness" of the sampled matrices
```

```
bestMinValuefound <- 0
bestMedValueFound <- 0
while(n_iter < max_iter){
  n_iter <- n_iter + 1

  if(n_iter %% 1000 ==  0){ # every 1000 iterations display
      the iteration number
    cat('#',n_iter,'#\n')
  }

  non_null_idxs <- sample.int(totEdgesBlock, size =
     numEdgesBlock) # sample the positions of the non-zero
     values
  # then create the block matrix with non-zero entries (
     sampled by a signed uniform) in the simulated
     positions
  OmegaBlock <- .idxs2mat(non_null_idxs, value = ifelse(
     rbinom(numEdgesBlock, 1, .5), +1, -1) * runif(
     numEdgesBlock, val_lb, val_ub), p = blockSize)

  eigvals <- eigen(OmegaBlock)$values # get the eigenvalues
      of the matrix
  minEig <- min(eigvals)
  if(minEig > bestMinEigFound){ # update the best result
     obtained so far, if it is the case
    bestMinEigFound <- minEig
  }
  if(minEig > 0){ # then the matrix is positive definite
     already
    condNum <- max(eigvals)/min(eigvals) # compute
       condition number
    if(condNum > maxCond){
      cat('Ill-conditioned matrix:', condNum,'\n')
    }else{ # both eigenvalues and condition number are ok
      cat('##### Found it! #####\n')
      Omega <- matrix(0, p, p) # build the matrix,
         replicating OmegaBlock along the diagonal
      for(i in 1:(nBlocks-1)){
        Omega[((i-1)*blockSize+1) : (i*blockSize), ((i-1)*
           blockSize+1) : (i*blockSize)] <- OmegaBlock
      }
      return(Omega)
    }
  }else{ # in this case, the matrix is not positive
     definite
    OmegaBlock <- OmegaBlock + (shrink - minEig)*diag(
       blockSize) # add along the diagonal the min Eigval +
        shrink (which will therefore become the min eigval)
      OmegaBlock <- cov2cor(OmegaBlock) # now along the diag
         we have values > 1 --> rescale the matrix to have 1
         along the diag

      minCor <- min(abs(OmegaBlock[OmegaBlock != 0])) # check
          which is the minimum non-zero value
      medCor <- median(abs(OmegaBlock[OmegaBlock != 0])) #
         check the median of the non-zero values
      if(minCor > bestMinValuefound){ # update the best
         result obtained so far, if it is the case
        bestMinValuefound <- minCor
      }
      if(medCor > bestMedValueFound){
        bestMedValueFound <- medCor
      }
      if(medCor > minMedThres){ # if the value is acceptable
        condNum <- kappa(OmegaBlock) # compute condition
           number
        if(condNum > maxCond){
          cat('Ill-conditioned matrix:', condNum,'\n')
        }else{
          cat('##### Found it! #####\n')
          Omega <- matrix(0, p, p) # build the matrix,
             replicating OmegaBlock along the diagonal
          for(i in 1:nBlocks){
            Omega[((i-1)*blockSize+1) : (i*blockSize), ((i-1)
               *blockSize+1) : (i*blockSize)] <- OmegaBlock
          }
          return(Omega)
        }
      }
    }
  }
  # if we get here, it means we did not find any good
     candidate: return the information about best results,
     which are helpful to tune the next attempt
  cat('### MATRIX NOT FOUND ( best eigval:', bestMinEigFound,
      '; best minCor:', bestMinValuefound,'; best medCor:',
     bestMedValueFound,') ###\n')
  return(NULL)
}
```

simBandShuffle

```
simBandShuffle <- function(p, width, shuffle.from.list,
   shuffle.to.list, K, lb = 0, ub = 1){
  # Function needed
```

```
.generateBand <- function(d, width){
  if(width>(d-1)){
    width <- d - 1
  }
  adj <- toeplitz(c(0, rep(1, width), rep(0, p - width -1))
     )
  return(adj)
}

.simOmega <- function(p, width){
  # Generate sparse precision matrix with band structure
  myinds <- .generateBand(d=p, width=width)
  M <- myinds
  ind <- upper.tri(M, diag=FALSE) & M == 1
  #M[ind] <- runif(sum(ind), min=-1, max=1) # sample from
     uniform [-1,1]
  M[ind] <- ifelse(rbinom(sum(ind), 1, .5), +1, -1) * runif
     (sum(ind), lb, ub) # sample from unif [-ub,-lb] U [lb,
     ub]
  M[lower.tri(M)] = t(M)[lower.tri(M)]
  minEig <- min(eigen(M,symmetric = TRUE, only.values =
     TRUE)$values) # get minimum eigenvalue
  M <- M + diag(p)*(abs(minEig)+0.1) # make the matrix pos
     def

  return(M)
}

.shuffle <- function(M, shuffle.from, shuffle.to){      #
   Randomly permute rows and columns
  if(shuffle.from != 0){
    idx <- sample(shuffle.from:shuffle.to)
    M[shuffle.from:shuffle.to, shuffle.from:shuffle.to] <-
       M[idx, idx]
  }
  return(M)
}

###

Omega1 <- .simOmega(p,width) # generate first matrix

if(K > 1){ # create OmegaList by replicating Omega1, then
   shuffle
  OmegaList <- rep(list(Omega1),K)
  if(length(shuffle.from.list) == 1){
    if(shuffle.from.list != 0){ # means no shuffle
      shuffle.from.list <- rep(list(shuffle.from.list),K-1)
    }
    }
    if(length(shuffle.to.list) == 1){ shuffle.to.list <- rep(
       list(shuffle.to.list),K-1) }
    # first matrix is never shuffled, if there is more than
       one group
    OmegaList[2:K] <- Map(.shuffle, OmegaList[2:K], shuffle.
       from.list, shuffle.to.list)
    return(OmegaList)

  }else{
    Omega1 <- .shuffle(Omega1, shuffle.from.list, shuffle.to.
       list)
    return(Omega1)
  }
}
```

B.4 Network structure inference

The following functions implement three approaches we described in Section 3.5 aimed at retrieving the network structure. All of them take as input the `beamDiff-class` object returned by `beamDiff` function and the level of the test.

UItest

```
require(fdrtool)

.computeP0fromHC <- function(condTailProb, type = 'max'){
  if(type == 'max'){
    pvals <- apply(condTailProb,1,max)
  }else{  # type = 'all'
    pvals <- unlist(condTailProb)
  }

  hcThres <- hc.thresh(pvals, plot = FALSE)
  rej_proportion <- length(which(pvals < hcThres))/length(
     pvals)

  p0 <- 1 - rej_proportion

  return(p0)
}

UItest <- function(object, presence = c(1:length(
   object@tableList)), method = "BFDR", thres = .1){
```

```
  logBFmat <- sapply(object@tableList, function(X) X[,'p_
      logBF']) # matrix with log(Bayes factors) for each
      condition
  tailsMat <- sapply(object@tableList, function(X) X[,'p_tail
      _prob']) # matrix with tail probabilities for each
      condition

  totEdges <- nrow(logBFmat)
  K <- ncol(logBFmat)

  res <- rep(0, totEdges) # initialize vector for results

  p0 <- .computeP0fromHC(tailsMat, type = 'max')

  logBFmat[, -c(presence)] <- - logBFmat[, -c(presence)] #
      change sign to all logBFs which are not tested for
      common presence

  overallLogBF <- rowSums(logBFmat) # sum of the logBFs (sign
       already adjusted)

  blfdrsvec <- p0/(exp(overallLogBF)*(1-p0)+p0)

  if(method=="blfdr"){
    res[which(blfdrsvec < thres)] <- 1
  }else{ # method == "BFDR"
    allthr <- sort(blfdrsvec, index.return=TRUE)
    BFDRsvec <- rep(0,length(allthr$x))
    BFDRsvec[allthr$ix] <- cumsum(allthr$x)/(1:length(allthr$
        x))

    res[which(BFDRsvec < thres)] <- 1
  }
  return(res)
}
```

hierTest_BF

```
require(cherry)

hierTest_BF <- function(object, alpha = .05){
  dfList <- object@tableList
  p <- object@p  # get number of covariates
  K <- object@numGroups
  totEdges <- p*(p-1)/2
  tailProb <- sapply(dfList, function(X) X[,'p_tail_prob']) #
      matrix with K cols corresponding to K tails
  rownames(tailProb) <- 1:totEdges

  alpha_BF_corrected <- alpha/totEdges # Bonferroni
     correction

  presMatrix <- matrix(0, totEdges, K) # initialize a matrix
     with K columns which will tell you where the edge is
     present
  rownames(presMatrix) <- 1:totEdges # use rownames to keep
     track of the original indexing

  for(i in 1:totEdges){
    pvals <- tailProb[i,] # consider the tails corresponding
       to i-th edge for its presence in the K conditions
    names(pvals) <- 1:K

    if(min(pvals) <= alpha_BF_corrected){ # perform hier
       testing only if the min pval in the row is below the
       threshold
      res <- curveFisher(pvals, alpha = alpha_BF_corrected,
         plot = FALSE) # hierarchical testing with cherry
         package

      selIdx <- 1
      while(res[selIdx] == selIdx && selIdx <= K){selIdx <-
         selIdx + 1} # increment the index until the num of
         rej hp is equal to the num of groups considered

      if(selIdx > 1){ # if so, the edge has been allocated to
          at least one group (as expected)
        presGroups <- as.numeric(names(res)[1:(selIdx-1)]) #
           get the names (or indicators) of the groups
        presMatrix[i,presGroups] <- 1
      }
    }
  }
  return(presMatrix)
}
```

hierBH

```
hierBH <- function(object, alpha1, alpha2){
  # alpha1: level of first test where we detect interesting
     edges
  # alpha2: level of second where we allocate edges to groups
  tailProbs <- sapply(object@tableList, function(X) X[,'p_
     tail_prob'])

  K <- ncol(tailProbs) # number of groups
  totEdges <- nrow(tailProbs) # number of edges

  res <- matrix(0, totEdges, K)

  # step0: simes pvalues
  simes_pvals <- apply(tailProbs, 1, function(x) min(K/rank(x
     )*x)) # for each row compute Simes pvalues and then take
      the minimum

  # step1: BH at FDR target level alpha1
  BH_simes_pvals <- p.adjust(simes_pvals, method = 'BH') #
     adjust values with BH
  selected_edges <- which(BH_simes_pvals < alpha1) #
     threshold at alpha1
  selEdges <- length(selected_edges) # number of selected
     edges

  # step2: group allocation on selected edges
  for(i in selected_edges){ # go through the edges that have
     been selected
    family_tailProbs <- tailProbs[i,]
    BH_family_tailProbs <- p.adjust(family_tailProbs, method
       = 'BH') # adjust values with BH
    selected_groups <- which(BH_family_tailProbs < alpha2*
       selEdges/totEdges) # threshold at alpha2 corrected to
       account for previous selection
    res[i, selected_groups] <- 1 # put 1 in the result matrix
  }

  return(res)
}
```

Acknowledgments

The Master thesis project described herein has been conducted during a six-months interniship, from September 2017 to March 2018, at the MRC Biostatistics Unit, University of Cambridge.
First of all, I would like to express my sincere gratitude to professor Francesca Ieva, my internal advisor in Politecnico di Milano, who informed me about this incredible opportunity at the University of Cambridge and supported me throughout the whole thesis project.
I would also like to thank all the BSU members, in particular the Phd students, who welcomed me since the very first day, and my supervisors, dr. Gwenaël Leday and dr. Leonardo Bottolo: I am extremely grateful for their constant support and all the things they taught me. I feel very lucky for having the opportunity to work with them, in such a prestigious research group.
I take this occasion to acknowledge my family, Giacomo and all the friends that made these five years really special.

Bibliography

Yoav Benjamini and Marina Bogomolov. Selective inference on multiple families of hypotheses. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1):297–318, 2014.

Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the royal statistical society. Series B (Methodological)*, pages 289–300, 1995.

Anders Ellern Bilgrau, Rasmus Froberg Brøndum, Poul Svante Eriksen, Karen Dybkær, and Martin Bøgsted. Estimating a common covariance matrix for network meta-analysis of gene expression datasets in diffuse large b-cell lymphoma. *arXiv preprint arXiv:1503.07990*, 2015.

Winston Chang and Javier Luraschi. *profvis: Interactive Visualizations for Profiling R Code*, 2017. URL `https://CRAN.R-project.org/package=profvis`. R package version 0.3.4.

Julien Chiquet, Yves Grandvalet, and Christophe Ambroise. Inferring multiple graphical structures. *Statistics and Computing*, 21(4):537–553, 2011.

Francis Crick. Central dogma of molecular biology. *Nature*, 227(5258):561, 1970.

Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL `http://igraph.org`.

Patrick Danaher. *JGL: Performs the Joint Graphical Lasso for sparse inverse covariance estimation on multiple classes*, 2013. URL `https://CRAN.R-project.org/package=JGL`. R package version 2.3.

Patrick Danaher, Pei Wang, and Daniela M Witten. The joint graphical lasso for inverse covariance estimation across multiple classes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(2):373–397, 2014.

David Edwards. *Introduction to graphical modelling.* Springer Science & Business Media, 2012.

Ronald Aylmer Fisher. *Statistical methods for research workers.* Genesis Publishing Pvt Ltd, 1925.

Ryan Gill, Somnath Datta, and Susmita Datta. A statistical framework for differential network analysis from microarray data. *BMC bioinformatics*, 11(1):95, 2010.

J. J. Goeman, A. Solari, and R. J. Meijer. *Cherry: Multiple testing methods for exploratory research*, 2015. R package version 0.6-11.

Jelle J Goeman, Aldo Solari, et al. Multiple testing for exploratory research. *Statistical Science*, 26(4):584–597, 2011.

Jian Guo, Elizaveta Levina, George Michailidis, and Ji Zhu. Joint estimation of multiple graphical models. *Biometrika*, 98(1):1–15, 2011.

Alexis Hannart and Philippe Naveau. Estimating high dimensional covariance matrices: A new look at the gaussian conjugate framework. *Journal of Multivariate Analysis*, 131:149–162, 2014.

Satoshi Hara and Takashi Washio. Learning a common substructure of multiple graphical gaussian models. *Neural Networks*, 38:23–38, 2013.

Lawrence Hunter. Molecular biology for computer scientists. *Artificial intelligence and molecular biology*, pages 1–46, 1993.

Michael I Jordan et al. Graphical models. *Statistical Science*, 19(1):140–155, 2004.

Bernd Klaus and Korbinian Strimmer. Signal identification for rare and weak features: higher criticism or false discovery rates? *Biostatistics*, 14 (1):129–143, 2012.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques.* MIT press, 2009.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

G. G. R. Leday and S. Richardson. Fast Bayesian inference in large Gaussian graphical models. *ArXiv e-prints*, March 2018.

Gwenael G.R. Leday and Ilaria Speranza. *beam: Fast Bayesian Inference in Large Gaussian Graphical Models*, 2018. URL `https://CRAN.R-project.org/package=beam`. R package version 1.0.

Jun Li, Yiling Lu, Rehan Akbani, Zhenlin Ju, Paul L Roebuck, Wenbin Liu, Ji-Yeon Yang, Bradley M Broom, Roeland GW Verhaak, David W Kane, et al. Tcpa: a resource for cancer functional proteomics data. *Nature methods*, 10(11):1046, 2013.

Sameer Mishra, Chanchala D Kaddi, and May D Wang. Pan-cancer analysis for studying cancer stage using protein expression data. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 8189–8192. IEEE, 2015.

Carl N Morris. Parametric empirical bayes inference: theory and applications. *Journal of the American Statistical Association*, 78(381):47–55, 1983.

David Pearson. A note on the multivariate normal distribution (mvn) and partitioned or block matrices. 2001.

Christine Peterson, Francesco C Stingo, and Marina Vannucci. Bayesian inference of multiple gaussian graphical models. *Journal of the American Statistical Association*, 110(509):159–174, 2015.

Christine B Peterson, Marina Bogomolov, Yoav Benjamini, and Chiara Sabatti. Many phenotypes without many false discoveries: error controlling strategies for multitrait association studies. *Genetic epidemiology*, 40 (1):45–56, 2016.

R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2013. URL `http://www.R-project.org/`.

RStudio Team. *RStudio: Integrated Development Environment for R.* RStudio, Inc., Boston, MA, 2016. URL `http://www.rstudio.com/`.

Yasin Şenbabaoğlu, Selçuk Onur Sümer, Francisco Sánchez-Vega, Debra Bemis, Giovanni Ciriello, Nikolaus Schultz, and Chris Sander. A multi-method approach for proteomic network inference in 11 human cancers. *PLoS computational biology*, 12(2):e1004765, 2016.

R John Simes. An improved bonferroni procedure for multiple tests of significance. *Biometrika*, 73(3):751–754, 1986.

Bruno M Tesson, Rainer Breitling, and Ritsert C Jansen. Diffcoex: a simple and sensitive method to find differentially coexpressed gene modules. *BMC bioinformatics*, 11(1):497, 2010.

John W Tukey. T13 n: The higher criticism. *Course notes, Stat*, 411, 1976.

John W Tukey. We need both exploratory and confirmatory. *The American Statistician*, 34(1):23–25, 1980.

Beatriz Valcárcel, Peter Würtz, Nafisa-Katrin Seich al Basatena, Taru Tukiainen, Antti J Kangas, Pasi Soininen, Marjo-Riitta Järvelin, Mika Ala-Korpela, Timothy M Ebbels, and Maria de Iorio. A differential network approach to exploring differences between biological states: an application to prediabetes. *PLoS One*, 6(9):e24702, 2011.

Katrijn Van Deun, H Hoijtink, Lieven Thorrez, Leentje Van Lommel, Frans Schuit, and Iven Van Mechelen. Testing the hypothesis of tissue selectivity: the intersection–union test and a bayesian approach. *Bioinformatics*, 25 (19):2588–2594, 2009.

Hadley Wickham. *Advanced r.* CRC Press, 2014.