




## rags2ridges: A One-Stop- $\ell_2$ -Shop for Graphical Modeling of High-Dimensional Precision Matrices

Carel F. W. Peeters   
Wageningen University  
& Research

Anders Ellern Bilgrau   
Aalborg University

Wessel N. van Wieringen   
Amsterdam UMC  
Vrije Universiteit Amsterdam

---

### Abstract

A graphical model is an undirected network representing the conditional independence properties between random variables. Graphical modeling has become part and parcel of systems or network approaches to multivariate data, in particular when the variable dimension exceeds the observation dimension. **rags2ridges** is an R package for graphical modeling of high-dimensional precision matrices through ridge ( $\ell_2$ ) penalties. It provides a modular framework for the extraction, visualization, and analysis of Gaussian graphical models from high-dimensional data. Moreover, it can handle the incorporation of prior information as well as multiple heterogeneous data classes. As such, it provides a one-stop- $\ell_2$ -shop for graphical modeling of high-dimensional precision matrices. The functionality of the package is illustrated with an example dataset pertaining to blood-based metabolite measurements in persons suffering from Alzheimer's disease.

*Keywords:* graphical modeling, high-dimensional data, networks, regularization, R.

---

## 1. Introduction

Network approaches have become ubiquitous in modern applied data science. In an abstract sense, networks are simply a collection of nodes (or vertices) that are pairwise connected by a constellation of links (or edges). Hence, a network is simply a graph. What the nodes and edges signify depends on the domain of application. In social networks for example, nodes often represent social actors such as people while the edges represent social interaction of some sort, such as friendship. In physical networks, nodes represent physical entities such as

mainframes while the edges represent physical connection such as through fiber optic cables. In biochemical networks, nodes represent biological entities such as protein molecules, while the edges represent some form of biological interaction, such as signal transduction. The scientific study of such networks is known as network science (for an overview, see, e.g., Newman 2010). Irrespective of the domain, network science provides a unifying framework for studying complex systems.

Our interest will be with networks in which the nodes represent random variables whose joint probability distribution is defined by the constellation of connections between nodes. Such networks are known as *graphical models*. More specifically, we will consider graphs  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  consisting of a finite set  $\mathcal{V}$  of  $p$  vertices, corresponding to random variables  $Y_1, \dots, Y_p$  with joint probability distribution  $P \sim \mathcal{N}_p(\mathbf{0}, \mathbf{\Sigma})$ , and set of edges  $\mathcal{E}$ , such that for all pairs  $\{Y_j, Y_{j'}\}$  with  $j \neq j'$ :

$$\left(\mathbf{\Sigma}^{-1}\right)_{jj'} = (\mathbf{\Omega})_{jj'} = 0 \iff Y_j \perp\!\!\!\perp Y_{j'} | \{Y_{j''} : j'' \neq j, j'\} \iff (j, j') \notin \mathcal{E}.$$

That is, a zero entry in the inverse covariance matrix (generally known as the concentration or precision matrix  $\mathbf{\Omega}$ ) mutually implies that the corresponding random variables are independent given the remaining variables which mutually implies that the corresponding variables are *not* connected by an undirected edge ( $(j, j') \notin \mathcal{E}$ ). Hence, the undirected graph  $\mathcal{G}$  is a conditional independence graph that describes the support of the (off-diagonal elements of the) precision matrix. Such networks are generally known as *Gaussian graphical models* (GGMs) and are the continuous counterpart to the Ising model from statistical physics (Ising 1925).

The entries of the precision matrix are proportional to partial correlations. A partial correlation is a measure of the conditional linear association between a random variable pair. Partial correlations form the basis of network reconstruction as the conditioning weeds out spurious associations. As such, GGMs are generally preferred to the simpler correlation networks, as available in R (R Core Team 2021) through the **WGCNA** package (Langfelder and Horvath 2008, 2012), in which two variables are connected when their marginal correlation coefficient is nonzero. The ability to distinguish between direct and indirect associations is especially important when the number of variables ( $p$ ) becomes large relative to the number of observations ( $n$ ). In such high-dimensional data the presence of spurious associations obfuscates a mechanistic understanding of the data.

The collection of high-dimensional data has become routine with the advent of high-throughput technology and data-stream mining algorithms. A GGM is often the model of choice when the desire is to get a handle on the system of interrelations among the variables but when there is not enough information to infer a causal structure (as indicated by a directed graph). This has led to a flurry of packages for extracting GGMs from high-dimensional data in R. The packages **glasso** (Friedman, Hastie, and Tibshirani 2008, 2019), **huge** (Liu, Lafferty, and Wasserman 2009; Jiang *et al.* 2021), and **clime** (Cai, Liu, and Luo 2011, 2012) estimate sparse precision matrices by either penalizing the log-likelihood of the data with an  $\ell_1$ -penalty on the precision matrix or by exploiting the relationship between  $\ell_1$ -penalized regression coefficients and precision matrix entries. These lasso-penalized approaches are popular as  $\ell_1$ -penalties automatically induce sparsity. Another approach would be to use ridge or  $\ell_2$ -penalization followed by post-hoc edge selection. Such an approach underlies the packages **GeneNet** (Schäfer and Strimmer 2005; Schaefer, Opgen-Rhein, and Strimmer 2021), **GGMridge** (Ha and Sun 2014; Ha 2016), and **beam** (Leday and Richardson 2019; Leday and Speranza 2020). When

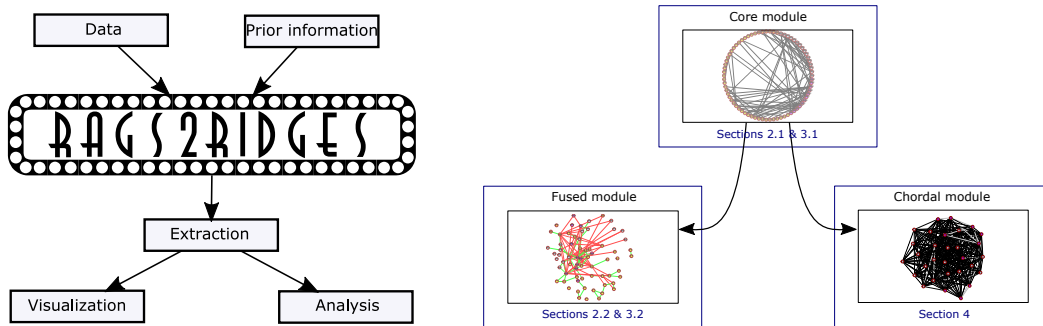


Figure 1: Left-panel: **rags2ridges** is a one-stop-shop as it (a) can combine data and prior information in inferring a GGM and (b) provides amenities for visualization and analysis of the corresponding network. Moreover, it (c) can do so for a single network as well as jointly for multiple networks. Right-panel: **rags2ridges** has a modular setup. The core module carries workhorse functions that support and feed into extension modules.

the data consist of multiple classes one may consider jointly estimating multiple (sparse) precision matrices. The packages **JGL** (Danaher, Wang, and Witten 2014; Danaher 2018), **RidgeFusion** (Price 2014; Price, Geyer, and Rothman 2015), and **pGMGM** (Gao, Zhu, Shen, and Pan 2016; Gao and Zhu 2016) were created to this end. All mentioned packages focus on network *extraction* by estimating precision matrices and their support. However, network evaluation also involves *visualization* and *analysis*. Visualization pertains to the effective graphical communication of networks. Analysis pertains to descriptions and inference on properties of retained networks. Hence, for visualization or analysis, users of the packages above have to defer to other packages such as **igraph** (Csárdi and Nepusz 2006; Csárdi 2022).

This paper introduces the R package **rags2ridges** (Peeters, Bilgrau, and Van Wieringen 2022), available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=rags2ridges> as well as its developer page at <https://github.com/CFWP/rags2ridges> and dedicated website at <https://cfwp.github.io/rags2ridges>. It provides functionality for the extraction as well as visualization and analysis of GGMs. The package uses  $\ell_2$ -penalized maximum likelihood estimation of the precision matrix coupled with post-hoc support determination. Such an approach has distinct advantages over a lasso-approach: (i) It gives probabilistic control over edge selection; (ii) it allows for the incorporation of prior information; (iii) it can handle more extreme  $p/n$  ratios (Van Wieringen and Peeters 2016); (iv) it is more successful in retrieving the network topology when the true topology is not very sparse (Van Wieringen and Peeters 2016). Especially this latter point is of interest as there is accumulating evidence that many networks are dense (Boyle, Li, and Pritchard 2017). What sets package **rags2ridges** apart from packages **GeneNet**, **GGMridge**, and **beam** is the usage of an algebraic  $\ell_2$ -penalty, resulting in an estimator that has lower risk (in terms of the expected quadratic or squared Frobenius loss) than the estimators in the mentioned packages (Van Wieringen and Peeters 2016). Moreover, in contrast to all packages mentioned above, **rags2ridges** can handle the estimation of a single precision matrix, as well as the joint estimation of multiple precision matrices, thus covering both the single and multiple group setting. These conveniences combined make **rags2ridges** a one-stop-shop for network evaluation using GGMs (left-hand panel of Figure 1). We note that there are also Python (Van Rossum *et al.*

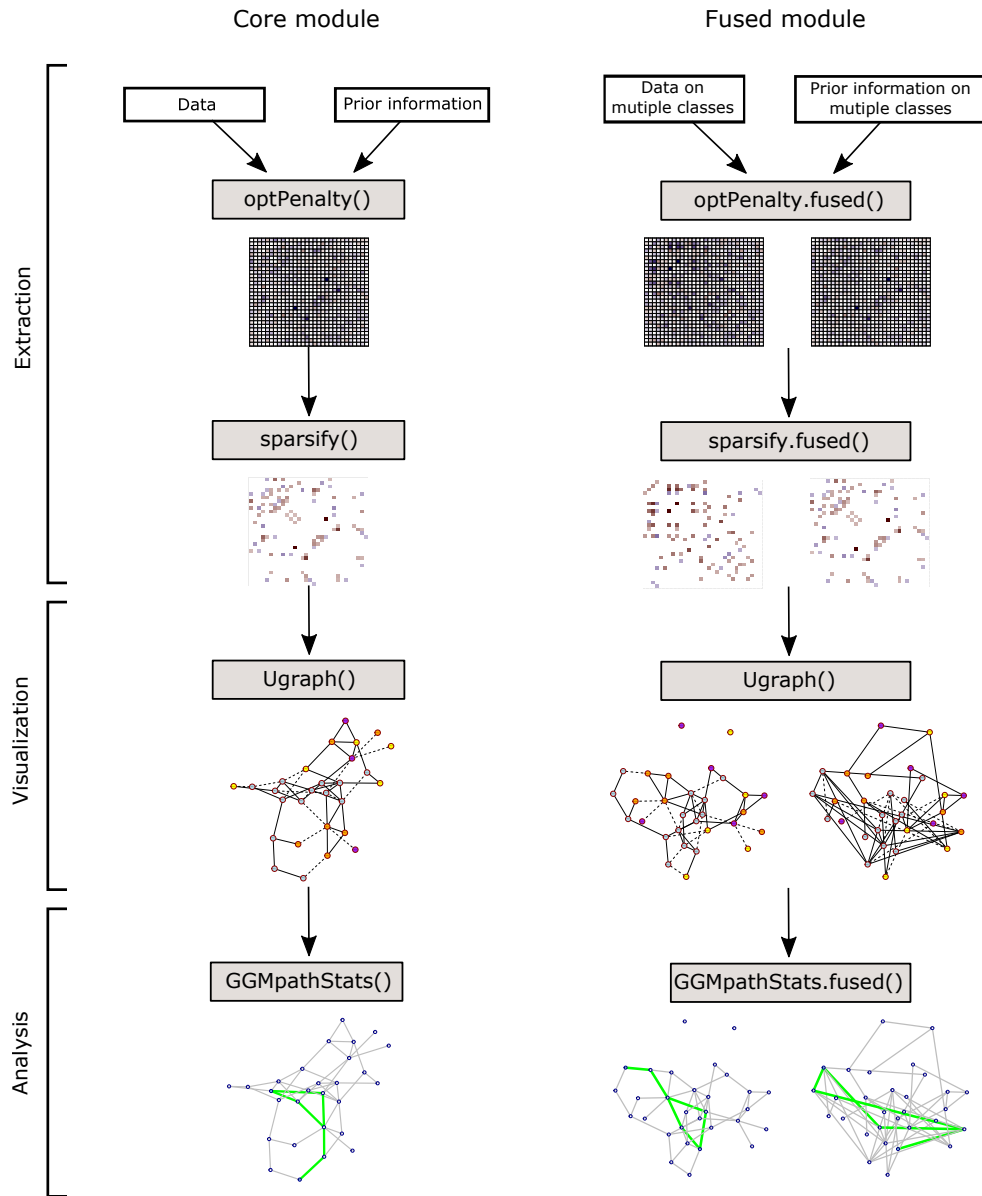


Figure 2: The **rags2ridges** package has a simple workflow. Left-panel: A typical workflow for the core module. The `optPenalty()` function family takes data and, when available, prior information (on the precision matrix of the data) to produce a regularized precision matrix. The support of this matrix may be determined with the `sparsify()` function. It outputs a sparsified regularized precision matrix that may be viewed as a weighted adjacency matrix representing a conditional independence graph. This graph may then be visualized with the `Ugraph()` function. The resulting network can be analyzed with a plethora of functions, such as `GGMpathStats()`. This function searches for the strongest (mediating and moderating) paths between nodes of interest. Right-panel: The fused module provides an analogous workflow for the joint extraction, visualization, and analysis of multiple networks. Table 1 provides an overview of the main functions in the package.

2011) libraries that provide an encompassing approach to (G)GMs, such as **skggm** (Laska and Narayan 2018) and **pgmpy** (Ankan and Panda 2015, 2021).

While **rags2ridges** was originally conceived as supporting the (exploratory) extraction and analysis of biochemical networks from omics-type data, it is certainly not restricted to that domain of application. The package is useful whenever one wants to infer undirected graphs in which the nodes represent random variables. As such the package has, next to transcriptomics (Van Wieringen and Peeters 2015), metabolomics (De Leeuw *et al.* 2017), and proteomics (Hayashi *et al.* 2019), also been successfully used in studies pertaining to, for example, immune activation (Blokhuys *et al.* 2019), ophthalmology (Elsman, Peeters, Van Nispen, and Van Rens 2020), and primate brain shape evolution (Clavel, Aristide, and Morlon 2019). The package is also useful when (non-sparse) regularized precision or covariance matrices are needed to support downstream analyses such as phylogenetic regressions (Clavel and Morlon 2020), regularized linear discriminant analysis (Friedman 1989), or regularized factor analysis (Mes *et al.* 2020).

The remainder of the paper reflects the modular setup of **rags2ridges** (right-hand panel of Figure 1). The core module carries the workhorse functions and revolves around the extraction, visualization, and analysis of single networks. Section 2.1 handles its technicalities while Section 3.1 gives an illustration. The core module feeds into the fused module, which revolves around the joint extraction, (differential) visualization, and analysis of multiple networks. Section 2.2 subsequently handles technicalities of the fused module while Section 3.2 gives an illustration. The core and fused modules are the most practically useful modules. Their workflows are sketched in Figure 2. The chordal module, as well as possible modular extensions, are discussed in Section 4.

## 2. Technical details

In this section we review the main technical details underlying the core (Section 2.1) and fused (Section 2.2) modules. In doing so, the main functions of the package are introduced. Moreover, it is shown how core functions have analogues in the fused module.

### 2.1. Core module

#### *Penalized ML precision estimation*

Let  $\mathbf{y}_i$ ,  $i = 1, \dots, n$ , be a  $p$ -dimensional random variate drawn from  $\mathcal{N}_p(\mathbf{0}, \boldsymbol{\Sigma})$ . Let  $\mathbf{S}$  denote the sample covariance estimate. The maximum likelihood (ML) estimator  $\hat{\boldsymbol{\Omega}}$  of the precision matrix  $\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1}$  then maximizes the following log-likelihood:

$$\mathcal{L}(\boldsymbol{\Omega}; \mathbf{S}) \propto \ln |\boldsymbol{\Omega}| - \text{tr}(\mathbf{S}\boldsymbol{\Omega}). \quad (1)$$

The maximum is achieved for  $\hat{\boldsymbol{\Omega}} = \mathbf{S}^{-1}$  whenever  $n > p$ . It is however well-known that  $\mathbf{S}$  is a poor estimate of  $\boldsymbol{\Sigma}$  when  $p \gtrsim n$ . When  $p$  approaches  $n$ ,  $\mathbf{S}$  will tend to become ill-conditioned, implying large propagation of numerical error when taking its inverse. When  $p > n$ ,  $\mathbf{S}$  is singular, leaving  $\hat{\boldsymbol{\Omega}}$  undefined. We approach these problems from a penalized ML estimation perspective.

Amend the log-likelihood (1) with the  $\ell_2$ -penalty

$$-\frac{\lambda}{2} \|\boldsymbol{\Omega} - \mathbf{T}\|_2^2,$$

where  $\mathbf{T}$  denotes a positive semi-definite symmetric target matrix and where  $\lambda \in (0, \infty)$  is a strictly positive penalty parameter. The resulting penalized log-likelihood can be maximized analytically, giving an algebraic ridge precision estimator (Van Wieringen and Peeters 2016):

$$\hat{\boldsymbol{\Omega}}(\lambda) = \left\{ \left[ \lambda \mathbf{I}_p + \frac{1}{4} (\mathbf{S} - \lambda \mathbf{T})^2 \right]^{1/2} + \frac{1}{2} (\mathbf{S} - \lambda \mathbf{T}) \right\}^{-1}. \quad (2)$$

The estimator can be conceived as a Stein-type estimator, calibrating in the inversion the low-bias but high-variance  $\mathbf{S}$  with the higher-bias but lower-variance  $\mathbf{T}$ . The penalty parameter  $\lambda$  then determines the amount of shrinkage. Through  $\mathbf{T}$  one can incorporate prior information without having to formally specify prior distributions. This may range from non-informative diagonal or null-matrices to highly informative weighted adjacency matrices representing existing knowledge on the network of interest. The function `default.target()` allows one to easily specify a plethora of non-informative target matrices.

The estimator in (2) has some appreciable properties (Van Wieringen and Peeters 2016): (i) It is always positive-definite; (ii) it has (when  $n > p$ )  $\mathbf{S}^{-1}$  as its right-hand limit when  $\lambda \rightarrow 0$ ; (iii) its left-hand limit when  $\lambda \rightarrow \infty$  is  $\mathbf{T}$ ; (iv) it is asymptotically unbiased; and (v) it is consistent. Moreover, under suitable penalty-choices and scalar target matrices, it dominates  $\mathbf{S}$  in terms of lower mean squared error (Van Wieringen 2017). In addition, the estimator lends itself to a natural process of iterative updating, in which the estimator from a previous round serves as  $\mathbf{T}$  in a subsequent round (Van Wieringen, Stam, Peeters, and Van de Wiel 2020). Also, it achieves lower risk than the archetypal ridge precision estimators  $[(1 - \lambda')\mathbf{S} + \lambda'\mathbf{T}]^{-1}$  (see, e.g., Ledoit and Wolf 2004; Schäfer and Strimmer 2005; Leday and Richardson 2019) and  $[\mathbf{S} + \lambda''\mathbf{I}_p]^{-1}$  (see, e.g., Ha and Sun 2014), both of which are also available in **rags2ridges**. As such, the estimator (2) provides an adequate basis for graphical modeling. It is available through the workhorse function `ridgeP()` when using the default argument `type = "Alt"`.

### *Choosing the penalty parameter*

An important issue is choosing a value for the penalty parameter. One wants to choose  $\lambda$  such that the precision estimate is well-conditioned while not unnecessarily suppressing relevant data signal. As the  $\ell_2$ -penalty does not automatically induce sparsity in the estimate, it is natural to choose a method that seeks loss efficiency rather than model selection consistency. **rags2ridges** employs cross-validation (CV). The  $K$ -fold CV score for the estimate  $\hat{\boldsymbol{\Omega}}(\lambda)$  based on the fixed penalty  $\lambda$  can be given as:

$$\varphi_K(\lambda) = \sum_{k=1}^K n_k \left\{ -\ln |\hat{\boldsymbol{\Omega}}_{-k}(\lambda)| + \text{tr}[\mathbf{S}_k \hat{\boldsymbol{\Omega}}_{-k}(\lambda)] \right\},$$

where  $n_k$  is the size of subset  $k$ , for  $k = 1, \dots, K$  disjoint subsets. Further,  $\mathbf{S}_k$  denotes the sample covariance matrix estimate based on subset  $k$ , while  $\hat{\boldsymbol{\Omega}}_{-k}(\lambda)$  denotes the estimated regularized precision matrix on all samples not in  $k$ . The optimal penalty value  $\lambda^*$  is then chosen such that:

$$\lambda^* = \arg \min_{\lambda} \varphi_K(\lambda),$$

which relates to the maximization of predictive accuracy. The `optPenalty()` family of functions gives access to penalty value determination by evaluation of the  $K$ -fold cross-validated negative log-likelihood score. It contains brute-force, approximation, and automated-search options. The automated-search option applies a root-finding algorithm to automatically search for the optimal value along the domain of  $\lambda$  and is our usual option of choice.

After obtaining  $\hat{\Omega}(\lambda^*)$ , we might like to assess its conditioning. We use the spectral condition number, i.e., the ratio of the maximum to minimum eigenvalue, for this assessment. A matrix with a high condition number is said to be ill-conditioned. Numerically, ill-conditioning will mean that small changes in the penalty parameter lead to dramatic changes in the condition number. When plotting the condition number against the (domain of the) penalty parameter, there is a point of relative stabilization (when working in the  $p > n$  situation) that can be characterized by a leveling-off of the acceleration along the curve (Peeters, Van de Wiel, and Van Wieringen 2020). Such condition number plots are the output of the `CNplot()` function. The condition number plot thus tracks the domain of the penalty parameter for which the regularized precision matrix is ill-conditioned and visualizes whether  $\hat{\Omega}(\lambda^*)$  finds itself in the stable domain of well-conditionedness.

### *Support determination*

If we have a well-conditioned estimate of the precision matrix we want to extract a network. In graphical modeling this implies determining the support of the estimated (regularized) precision matrix. Support determination is best performed on a scaled version of the precision matrix: the partial correlation matrix  $\hat{\mathbf{P}}(\lambda^*)$ . The support of the partial correlation matrix equals the support of the corresponding precision matrix, but the former simplifies evaluation by having all entries on the same scale. The `sparsify()` function determines the support of a partial correlation/precision matrix by post-hoc thresholding and sparsifies it accordingly.

The function offers multiple options for thresholding. The most simple is available through the argument `threshold = "top"`. It allows one to retain the  $x$  strongest edges in terms of their absolute partial correlation values by additionally specifying `top = x`. Another simple option is accessed through `threshold = "absValue"`, which retains all edges whose corresponding absolute partial correlation  $|\hat{\mathbf{P}}(\lambda^*)_{jj'}| \geq \text{absValueCut}$ . The `sparsify()` function also offers probabilistic control over edge selection through a multiple testing procedure. Specifying the argument `threshold = "localFDR"` leads to support determination by a local false discovery rate (lFDR) procedure proposed by Schäfer and Strimmer (2005). It assumes that the nonredundant partial correlation coefficients follow a mixture distribution:

$$f \left\{ [\hat{\mathbf{P}}(\lambda^*)]_{jj'} \right\} = \eta_0 f_0 \left\{ [\hat{\mathbf{P}}(\lambda^*)]_{jj'}; \kappa \right\} + (1 - \eta_0) f_{\mathcal{E}} \left\{ [\hat{\mathbf{P}}(\lambda^*)]_{jj'} \right\}, \quad (3)$$

with mixture weight  $\eta_0 \in [0, 1]$ . In (3),  $f_0\{\cdot\}$  denotes the distribution of a null-edge, which is a scaled beta-density with  $\kappa$  degrees of freedom. The distribution of a present edge is given by  $f_{\mathcal{E}}\{\cdot\}$ . In the  $p > n$  situation one has to estimate  $\kappa$ ,  $\eta_0$  and  $f_{\mathcal{E}}\{\cdot\}$ , the details of which can be found in Efron (2010) and Schäfer and Strimmer (2005). With these estimates the lFDR can be calculated, representing the empirical posterior probability that  $Y_j$  and  $Y_{j'}$  are *not* connected given their observed partial correlation:

$$P \left( (j, j') \notin \mathcal{E} \mid [\hat{\mathbf{P}}(\lambda^*)]_{jj'} \right) = \frac{\hat{\eta}_0 f_0 \left\{ [\hat{\mathbf{P}}(\lambda^*)]_{jj'}; \hat{\kappa} \right\}}{\hat{\eta}_0 f_0 \left\{ [\hat{\mathbf{P}}(\lambda^*)]_{jj'}; \hat{\kappa} \right\} + (1 - \hat{\eta}_0) \hat{f}_{\mathcal{E}} \left\{ [\hat{\mathbf{P}}(\lambda^*)]_{jj'} \right\}}.$$

The procedure then will retain those edges whose probability of being present equals or exceeds the probability specified in the `FDRcut` argument, i.e., it will retain those edges for which  $1 - P\left((j, j') \notin \mathcal{E}[[\hat{\mathbf{P}}(\lambda^*)]_{jj'}]\right) \geq \text{FDRcut}$ .

### *Visualization*

The `Ugraph()` function is the main function for the network visualization of the sparsified precision matrix  $\hat{\mathbf{\Omega}}(\lambda^*)_0$  or the sparsified partial correlation matrix  $\hat{\mathbf{P}}(\lambda^*)_0$ . Visualization is very important for a first understanding of a network. The very same network can be visualized in uncountably many ways, and some ways might opaque rather than enlighten its structure and properties. Hence, the `Ugraph()` function tries to offer flexibility as well as easy access to variety in graph drawing. Through `Ugraph()` one can control, among others, node size, node color, edge color, edge thickness, and node-pruning (trimming nodes that are unconnected). Control over these elements is essential when the network has many nodes and/or many connections. Moreover, `Ugraph()` allows for the differentiation between positive and negative edge weights. Also, the function gives a plethora of options for (force-directed) layout algorithms. Most layout functions supported by `igraph` are supported (the function is partly a wrapper around certain `igraph` functions). These layouts can be invoked by a character that mimics a call to an `igraph` layout function through the `lay` argument. When using `lay = NULL` one can specify the placement of nodes with the `coords` argument. The row dimension of this matrix should equal the number of (pruned) vertices. The column dimension then should equal 2 (for 2D layouts) or 3 (for 3D layouts). The `coords` argument can also be viewed as a convenience argument as it enables one, e.g., to layout a graph according to the coordinates of a previous call to `Ugraph()`. This enables the visual comparison of multiple graphs.

### *Analysis*

Following visualization we might desire analyzing the retained structure. The simplest analyses describe network properties at the node-level. Dominant in this respect are *centrality measures*, which aim to identify the most important nodes (hubs) within a network. There are many centrality measures that capture different aspects of node importance. Two well-known and oft studied centrality measures are degree centrality and betweenness centrality. Degree centrality simply indicates the number of connections in which a node takes part. It is indicative of the nodes that are central or influential in terms of the number of connections: more connections could imply deeper regulatory influence. Betweenness centrality measures centrality in terms of information flow. Under the assumption that information is passed over short(est) paths a node becomes central when the number of short(est) paths that pass through it is high. The `GGMnetworkStats()` function calculates, for a given sparsified precision or partial correlation matrix, the degree and betweenness centralities as well as various other measures of centrality. It also calculates the number of positive and the number of negative edges for each node. In addition, for each variate the mutual information (with all other variates), the variance, and the partial variance are represented. See, e.g., [Newman \(2010\)](#) for more information on centrality measures and other node-level statistics.

We might also analyze network properties from the perspective of paths. A path is a sequence of edges that connect a sequence of distinct nodes. One might then be interested, for example, in the collection of strongest paths between two endnodes of interest. The `GGMpathStats()` function uses a result by [Jones and West \(2005\)](#) stating that the observed covariance between any pair of nodes can be decomposed as a sum of path contributions for all paths connecting



these nodes in the conditional independence graph. The function returns all paths, their respective lengths, and their respective contributions to the marginal covariance between the endnodes of interest. It allows one to identify, for example, the strongest paths between such nodes, and if these are mediating or moderating paths.

Another analysis perspective is found in groups of nodes. The nodes in a network often cluster in groups: collections of nodes that are more deeply connected to each other than to nodes outside their direct topological environment. These groups are often called *communities*. There is interest in the detection of these communities as they are thought to represent functional modules. Community detection, loosely speaking, then refers to “the search for naturally occurring groups in a network” (Newman 2010, p. 371). The `Communities()` function performs community detection. It uses an edge-betweenness-based method of community detection, commonly known as the Girvan-Newman algorithm (Newman and Girvan 2004). The community structure in the graph can also be visualized when using the argument `graph = TRUE`. The arguments for this visualization are analogous to those in the `Ugraph()` function.

## 2.2. Fused module

Often, data consist of observations that are subject to class membership. Class membership may have different connotations. It may refer to certain sub-classes within a single data set such as disease subtypes. It may also designate different data sets or studies. Likewise, “the class indicator may also refer to a conjunction of both subclass and study membership to form a two-way design of factors of interest (e.g., breast cancer subtypes present in a batch of study-specific data sets)” (Bilgrau, Peeters, Eriksen, Bøgsted, and Van Wieringen 2020, p. 3). In such settings one may desire the joint estimation of multiple regularized precision matrices from (aggregated) high-dimensional data consisting of distinct classes. The fused module in `rag2ridges` contains functions for doing so.

Suppose  $\mathbf{y}_{ig}$  is a realization of a  $p$ -dimensional random variate for  $i = 1, \dots, n_g$  independent observations nested within  $g = 1, \dots, G$  non-overlapping classes, drawn from  $Y_g \sim \mathcal{N}_p(\mathbf{0}, \Sigma_g)$ . Our desire is to obtain an estimate of the precision matrix  $\Omega_g$  for each class  $g$ . Let  $\{\Omega_g\}$  and  $\{\mathbf{S}_g\}$  respectively denote the sets  $\{\Omega_g\}_{g=1}^G$  and  $\{\mathbf{S}_g\}_{g=1}^G$ , and consider the log-likelihood for the joint  $n_\bullet = \sum_{g=1}^G n_g$  data observations:

$$\mathcal{L}(\{\Omega_g\}; \{\mathbf{S}_g\}) \propto \sum_g n_g \left\{ \ln |\Omega_g| - \text{tr}(\mathbf{S}_g \Omega_g) \right\}. \quad (4)$$

We again consider a high-dimensional setting in which  $p > n_\bullet$ . In such a situation the class-specific ML estimates  $\mathbf{S}_g$  are singular and their inverses do not exist. Moreover, if they could be obtained then the precision estimates would be treated as if they exist in independent class-specific vacuums. The pooled covariance matrix  $n_\bullet^{-1} \sum_{g=1}^G n_g \mathbf{S}_g$ , often used as an estimate of the common covariance matrix across classes, will also be singular with an undefined inverse. In addition, if this inverse would exist, it ignores class-specific information. These problems can be approached by generalizing the  $\ell_2$ -penalized ML framework from Section 2.1 to *targeted fused ridge estimation*.

Amend, to this end, the log-likelihood (4) with the following fused  $\ell_2$ -penalty (Bilgrau *et al.* 2020):

$$-\sum_g \frac{\lambda_{gg}}{2} \|\Omega_g - \mathbf{T}_g\|_2^2 + \sum_{g_1, g_2} \frac{\lambda_{g_1 g_2}}{4} \|(\Omega_{g_1} - \mathbf{T}_{g_1}) - (\Omega_{g_2} - \mathbf{T}_{g_2})\|_2^2, \quad (5)$$

Core function	Fused analogue	Supports
<code>ridgeP()</code>	<code>ridgeP.fused()</code>	Extraction
<code>default.target()</code>	<code>default.target.fused()</code>	Extraction
<code>optPenalty()</code>	<code>optPenalty.fused()</code>	Extraction
<code>sparsify()</code>	<code>sparsify.fused()</code>	Extraction
<code>Ugraph()</code>	<code>DiffGraph()</code>	Visualization
<code>GGMnetworkStats()</code>	<code>GGMnetworkStats.fused()</code>	Analysis
<code>GGMpathStats()</code>	<code>GGMpathStats.fused()</code>	Analysis
<code>Communities()</code>	–	Analysis

Table 1: Overview of the main functions in the **rags2ridges** package.

where  $\mathbf{T}_g$  denote class-specific positive semi-definite symmetric target matrices,  $\lambda_{gg} \in (0, \infty)$  denote class-specific ridge penalty parameters, and where  $\lambda_{g_1g_2} = \lambda_{g_2g_1} \in [0, \infty)$  indicate pair-specific *fusion* penalty parameters. All penalties can be collected in the penalty matrix  $\mathbf{\Lambda}$ , carrying the  $\lambda_{gg}$  along the diagonal and the  $\lambda_{g_1g_2}$  off-diagonal. The penalty  $\lambda_{gg}$  controls the rate of shrinkage of  $\mathbf{\Omega}_g$  towards  $\mathbf{T}_g$  while  $\lambda_{g_1g_2}$  determines the retainment of entry-wise similarities between  $(\mathbf{\Omega}_{g_1} - \mathbf{T}_{g_1})$  and  $(\mathbf{\Omega}_{g_2} - \mathbf{T}_{g_2})$ . The addition of the fused  $\ell_2$ -penalty “allows for the joint estimation of multiple precision matrices from high-dimensional data classes that chiefly share the same structure but that may differentiate in locations of interest” (Bilgrau *et al.* 2020, p. 5). It also improves sensitivity and specificity of topological discoveries by borrowing power across classes (Bilgrau *et al.* 2020). The formulation of the penalty in (5) is very general and can be specialized to many special cases, including the single-network setting of Section 2.1.

The targeted fused penalized log-likelihood has a closed-form maximizing argument for any given class  $g$  (Bilgrau *et al.* 2020):

$$\hat{\mathbf{\Omega}}_g(\mathbf{\Lambda}, \{\mathbf{\Omega}_{g'}\}_{g' \neq g}) = \left\{ \left[ \bar{\lambda}_g \mathbf{I}_p + \frac{1}{4} (\bar{\mathbf{S}}_g - \bar{\lambda}_g \mathbf{T}_g)^2 \right]^{1/2} + \frac{1}{2} (\bar{\mathbf{S}}_g - \bar{\lambda}_g \mathbf{T}_g) \right\}^{-1}, \quad (6)$$

which can be viewed as a generalization of (2), with

$$\bar{\mathbf{S}}_g = \mathbf{S}_g - \sum_{g' \neq g} \frac{\lambda_{g'g}}{n_g} (\mathbf{\Omega}_{g'} - \mathbf{T}_{g'}), \quad \text{and} \quad \bar{\lambda}_g = \frac{\sum_{g'} \lambda_{g'g}}{n_g}.$$

The estimator (6) has appreciable properties analogous to the properties stated for (2). It forms the iterative core of finding the general maximizing argument  $\{\hat{\mathbf{\Omega}}_g\}$  by block coordinate ascent (Bilgrau *et al.* 2020).

The block coordinate ascent procedure can itself be subjected to cross-validation to select values for the entries in  $\mathbf{\Lambda}$ . In fact, the full functionality of the core module can be extended to deal with the joint estimation and evaluation of multiple (sparsified) precision matrices. The functions of this fused module, suffixed with `.fused`, mirror the functions of the core module. Table 1 gives an overview of core functions and their fused analogues as well as their usage in the network evaluation cycle: extraction, visualization, analysis.

### 2.3. Some computational details

The worst-case asymptotic time complexity of `ridgeP()` and `ridgeP.fused()` is  $\mathcal{O}(p^3)$  due to the necessity of the matrix square root. Computational efficiency is then ensured through the employment of various strategies. Importantly, note that only a single spectral decomposition is required in order to obtain the complete regularization path under a *scalar matrix* target choice (Van Wieringen and Peeters 2016; Peeters *et al.* 2020). The computational benefit stems from the property of rotation equivariance: Under such targets the ridge estimator only operates on the eigenvalues of the sample covariance matrix. This benefit transfers to the fused setting (Bilgrau *et al.* 2020). For more general targets we cannot exploit equivariance such that alternative strategies are employed for efficiency (Bilgrau *et al.* 2020). First, for certain special cases, closed-form solutions exist (see Section 2 of the SM to Bilgrau *et al.* 2020) to the optimization of (4) amended with (5) and these are used when appropriate. Second, when not appropriate they are used to provide warm starts for the general optimization problem. Third, we exploit the fact that our precision matrices can be obtained without inversion (see Proposition 5 of Bilgrau *et al.* 2020). Hence, we can avoid the expensive operations involved in matrix inversion. Fourth, most functions have C++ at their core (Eddelbuettel and François 2011), ensuring additional computational swiftness (see, e.g., Section 5.6 of Bilgrau *et al.* 2020). Fifth, efficient root-finding algorithms are employed to find optimal penalty-values. Combined, these efforts make analyses with large  $p$  feasible. Comparative timing benchmarks are carried out in Bilgrau *et al.* (2020). Our workhorse functions compare favorably to our competitors in terms of speed and scalability. The same can be said for loss, stability, and (sensitivity and specificity of) topology-retrieval (Van Wieringen and Peeters 2015, 2016; Bilgrau *et al.* 2020).

## 3. Illustrations

Section 3.1 illustrates the extraction, visualization, and analysis of *single* networks through the core module. Section 3.2 illustrates the joint extraction, visualization, and analysis of *multiple* networks through the fused module. Throughout, we will be using metabolomics data on patients with Alzheimer’s disease (AD). AD is a chronic neurodegenerative disease and the main cause of dementia. Metabolomics refers to the collective quantification of metabolites: small-molecules that result from metabolic processes. Metabolomics on peripheral fluids is of interest in terms of non-invasive measurement of disease-specific biochemical changes.

The data, internally packaged for educational purposes, are a doubly anonymized subset of the data reported in De Leeuw *et al.* (2017). The data can be loaded with a simple call.

```
R> data("ADdata", package = "rags2ridges")
```

`ADdata` consists of 3 objects related to blood-based metabolomics data on various AD subtypes. `ADmetabolites` is a matrix containing metabolic expressions of  $p = 230$  metabolites (rows) on  $n = 127$  samples (columns). The `sampleInfo` object is a ‘`data.frame`’ containing clinical information on the samples. This information pertains to diagnosis: AD Class 1 (AD patients without a known genetic predisposition for AD) or AD Class 2 (AD patients with a known genetic predisposition for AD). The `variableInfo` object is also a ‘`data.frame`’, containing information on the metabolic features. This information pertains to the compound family a metabolite belongs to: amines, organic acids, lipids, or oxidative stress compounds.

These objects are easily probed by calling `head(ADmetabolites)`, `head(sampleInfo)`, and `table(variableInfo)`, providing basic insights into metabolite expression, and clinical and feature information.

### 3.1. Core module

#### *Extraction*

We first need an estimate of the precision matrix. In this section we will be interested in data for AD Class 2 (i.e, AD patients with a known genetic predisposition for AD). The precision matrix estimate will have to be a regularized estimate as the number of features ( $p = 230$ ) is larger than the number of observations ( $n = 87$  for AD Class 2). We will use the ridge estimate of the precision matrix given in (2).

We start by scaling the data, as the features have different scales and as the variability of the features may differ substantially.

```
R> ADclass2 <- ADmetabolites[, sampleInfo$ApoEClass == "Class 2"]
R> ADclass2 <- scale(t(ADclass2))
```

One could simply call `ridgeP()` if one would desire a ridge estimate of the precision matrix for a *given* penalty value (say, e.g.,  $\lambda = 0.3$ ).

```
R> P <- ridgeP(covML(ADclass2), lambda = 0.3)
```

The (scaled) data can also be used in the `optPenalty.kCVauto()` function which uses function `ridgeP()` internally. This function determines the optimal value of the penalty parameter by application of the Brent algorithm (Brent 1971) to the *KCV* negative log-likelihood score. The search for the optimal value is automatic in the sense that in order to invoke the root-finding abilities of the Brent method, only a minimum value (`lambdaMin`) and a maximum value (`lambdaMax`) for the penalty parameter need to be specified (from which a starting penalty value is automatically generated). As we have little *a priori* knowledge, we will keep the target matrix non-informative: an identity matrix (one of the default targets in the `default.target()` function). We choose  $K = 10$ , such that the penalty value at which the 10-fold cross-validated negative log-likelihood score is minimized is deemed optimal.

```
R> set.seed(1234)
R> OPT <- optPenalty.kCVauto(ADclass2, fold = 10, lambdaMin = 1e-07,
+   lambdaMax = 20, target = default.target(covML(ADclass2)),
+   type = "DUPV")
```

The output is an object of class ‘list’, with `$optLambda` containing the optimal penalty value and `$optPrec` containing the precision matrix under the optimal value of the penalty parameter. It is often informative to have a look at the optimal penalty value.

```
R> OPT$optLambda
```

```
0.1561957
```

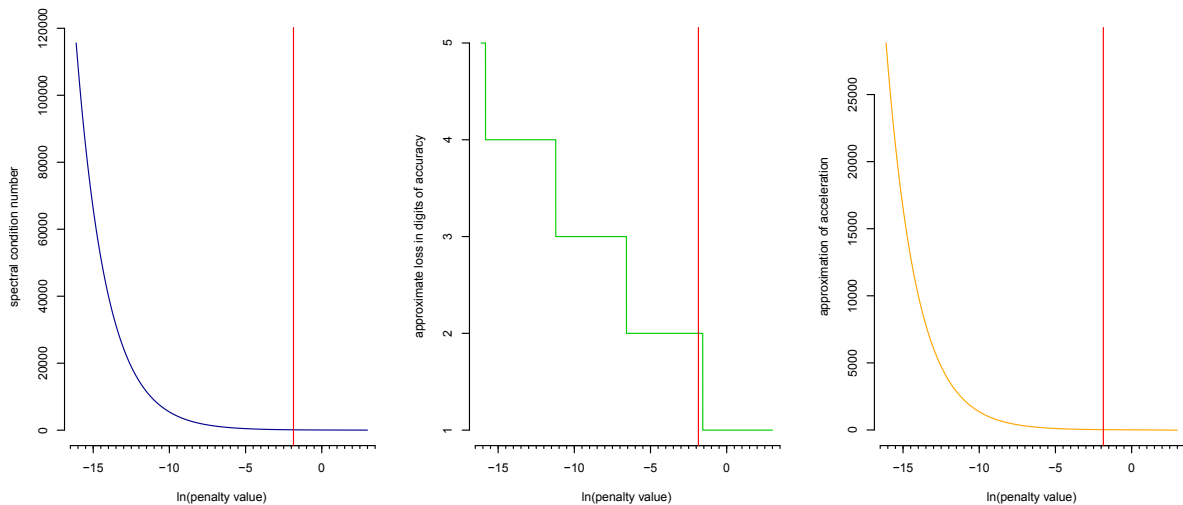


Figure 3: A condition number plot with interpretational aids.

Using the `CNplot()` function, we can assess the conditioning of  $\hat{\Omega}(.1561957)$ . It outputs a graph of the spectral condition number over the domain of the penalty parameter.

```
R> CNplot(covML(ADclass2), lambdaMin = 1e-07, lambdaMax = 20, step = 5000,
+ target = default.target(covML(ADclass2), type = "DUPV"), laids = TRUE,
+ vertical = TRUE, value = OPT$optLambda)
```

When `laids = TRUE`, the basic condition number plot is amended/enhanced with two additional plots (over the same domain of the penalty parameter as the basic plot): (i) the approximate loss in digits of accuracy (for the operation of inversion) and (ii) an approximation to the second-order derivative of the curvature in the basic plot. These interpretational aids can enhance interpretation of the basic condition number plot (Peeters *et al.* 2020). When `vertical = TRUE` a vertical line is added at the constant given in the argument `value`. This option can be used to assess if the previously obtained optimal penalty value has led to a precision estimate that is well-conditioned. The graph is given in Figure 3. The far-left panel gives the basic spectral condition number plot. The red vertical line indicates the value for the penalty that was deemed optimal. This optimal penalty value lies in the domain of well-conditionedness: small perturbations in this value do not affect the spectral condition number greatly. The middle and far-right panels depict the interpretational aids. From the middle plot, for example, we understand that the estimated loss in digits of accuracy in operations based on the optimal precision matrix is 2.

Now that we have a well-conditioned estimate of the precision matrix we want to determine its support. We will use the `lFDR` option in the `sparsify()` function for doing so. Elements are retained if their posterior probability of being present (equalling  $1 - \text{lFDR}$ )  $\geq \text{FDRcut}$ . As metabolic networks are very dense, we set `FDRcut` quite high: `.999`.

```
R> P0 <- sparsify(OPT$optPrec, threshold = "localFDR", FDRcut = 0.999,
+ verbose = FALSE)
```

```
- Retained elements: 310
- Corresponding to 1.18 % of possible edges
```

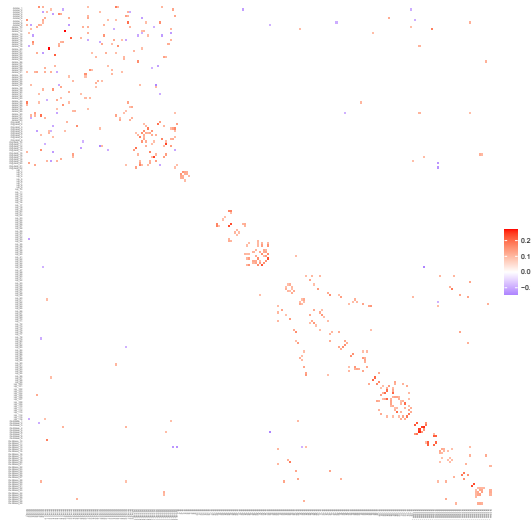


Figure 4: Heatmap of the sparsified partial correlation matrix. The legend represents the color scale for entries in the negative (blue) and positive (red) range.

The function returns an object of class ‘list’: `P0$sparsePrecision` contains the sparsified precision matrix while `P0$sparseParCor` contains its scaled version, the sparsified partial correlation matrix.

Note that the matrix object returned by a call to `ridgeP()` (of class ‘`ridgeP`’), as well as any function that uses `ridgeP()` internally, can be dispatched to `print()`, `summary()`, and `plot()`. A first visualization of the sparsified matrices is then easily obtained through a call to `plot()`.

```
R> plot(P0$sparseParCor, diag = FALSE, textsize = 3)
```

The resulting heatmap, given in Figure 4, is essentially a visualization of the extracted network, as it represents the weighted adjacency matrix of the conditional independence graph at hand. However, more insightful network visualization is possible, a topic to which we turn next.

### *Visualization*

Visualization is very important for a first understanding of the graph. Visualization should always be geared towards maximizing information and clarity with a minimum of (visual) clutter. The code below gives increasingly elaborate calls to `Ugraph()`.

```
R> opar <- par(mfrow = c(1, 3))
R> Ugraph(P0$sparseParCor, type = "fancy", Vsize = 3, Vcex = 0.1)
R> Ugraph(P0$sparseParCor, type = "fancy", Vsize = 3, Vcex = 0.1,
+   prune = TRUE)
R> set.seed(1567)
R> Coords <- Ugraph(P0$sparseParCor, type = "fancy", lay = "layout_with_fr",
+   prune = TRUE, Vsize = 7, Vcex = 0.3)
R> par(opar)
```

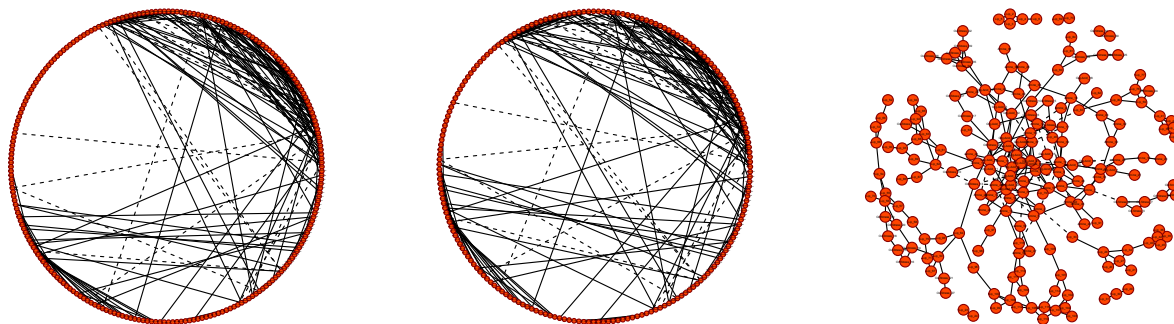


Figure 5: Visual results from various simple calls to `Ugraph()`.

The first call only specifies the type of graph (`type`), the size of the vertices (`Vsize`), and the size of the vertex labels (`Vcex`). When `type = "fancy"` a graph is given in which dashed lines indicate negative edges while solid lines indicate positive edges. The result (left-hand panel of Figure 5) is a graph/network under a default layout that gives a circular placement of the vertices. There are quite some features, such that the network structure is not directly clear. One could try to clarify the structure by pruning the graph. That is, by removing unconnected nodes. This is easily achieved by adding the `prune = TRUE` argument, given in the second call. As there are few unconnected nodes, the pruning does little to alleviate the clutter (middle-panel of Figure 5). In this situation it might be better to use a layout different from the circular placement of vertices. One option is the (force-directed) Fruchterman-Reingold algorithm (Fruchterman and Reingold 1991) which tries to position the nodes such that all edges are approximately of equal length and tries to minimize the number of crossing edges. Such a layout can be obtained by specifying `lay = "layout_with_fr"`, given in the third call. The structure of the network now becomes more clear as the new layout seems to suggest that there are core and periphery substructures (right-hand panel of Figure 5).

Even more information can be conveyed if we would include information on compound families in the node-colorings. That is, we could color each node according to the compound family it belongs to. The `Ugraph()` function also supports visual comparison of networks as it outputs the layout-coordinates of the network that is visualized. Hence, for the next call to `Ugraph()` the retained coordinates of the previous call are used for node-placement.

```
R> PcorP <- pruneMatrix(P0$sparseParCor)
R> Colors <- rownames(PcorP)
R> Colors[grep("Amine", rownames(PcorP))] <- "lightblue"
R> Colors[grep("Org.Acids", rownames(PcorP))] <- "orange"
R> Colors[grep("Lip", rownames(PcorP))] <- "yellow"
R> Colors[grep("Ox.Stress", rownames(PcorP))] <- "purple"
R> Ugraph(PcorP, type = "fancy", lay = NULL, coords = Coords,
+   Vcolor = Colors, Vsize = 7, Vcex = 0.3)
```

The result is given in Figure 6. The colorings convey that the core-structure consists of amine, oxidative stress and organic acid compounds mostly, while the periphery-structure consists of lipid compounds mostly. We note that, in case more flexibility in visualization is needed than provided by `Ugraph()`, the sparsified precision matrix is easily parsed to (for example) `graph.adjacency()`, enabling access to the flexibility of **igraph** graphs.

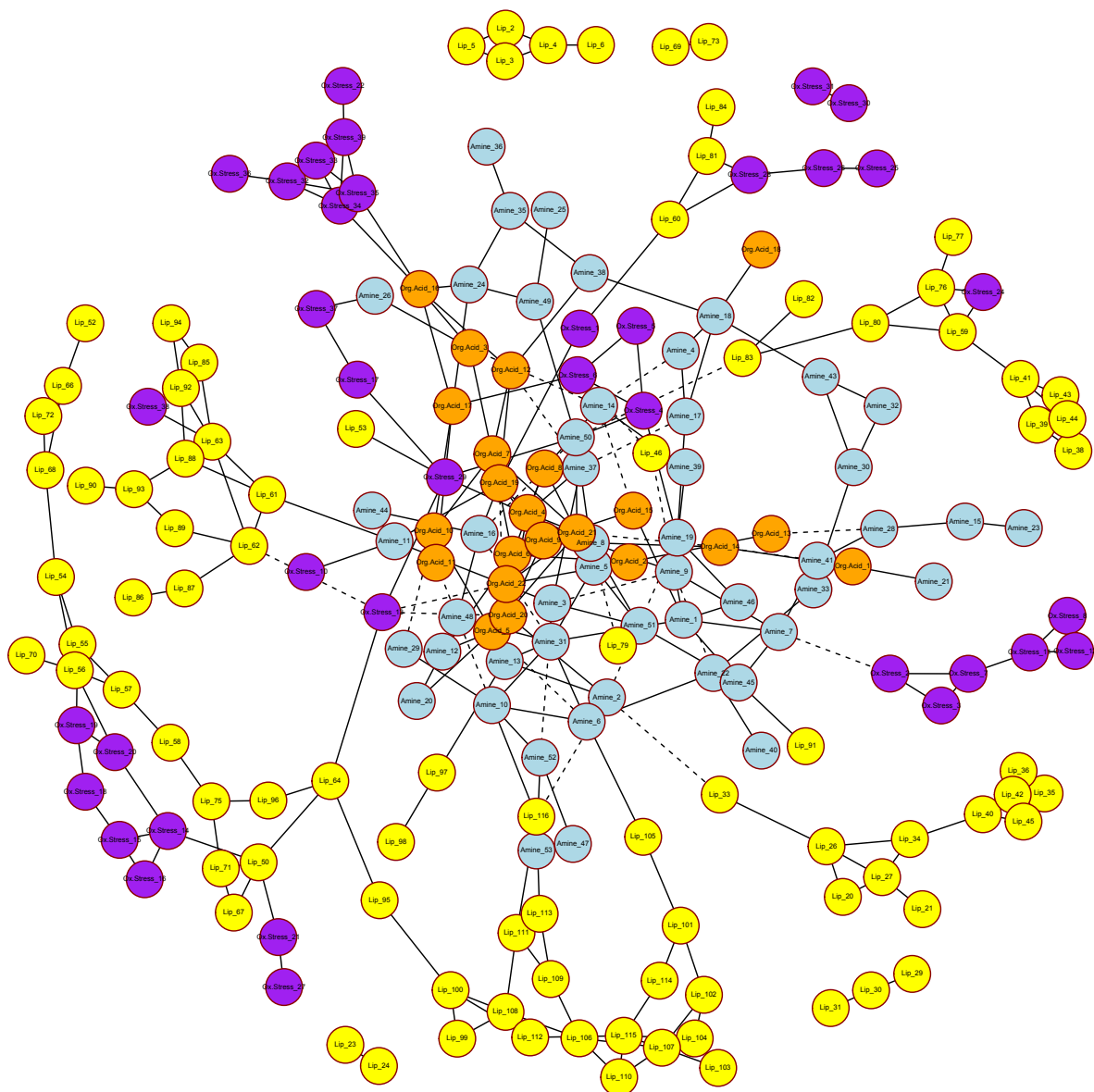


Figure 6: Metabolite network for AD patients with a known genetic predisposition for AD. Dashed edges indicate negative edge weight. Solid edges indicate positive edge weight. Node colorings are according to compound family: blue for amines, orange for organic acids, yellow for lipids, and purple for oxidative stress compounds.

### Analysis

Following visualization we might desire analyzing the retained structure. The `GGMnetworkStats()` function calculates various network statistics, mostly at the node-level.

```
R> NwkSTATS <- GGMnetworkStats(PcorP)
```

It outputs an object of class 'list' with, among others, slots for node degree (`$degree`) and



node betweenness (`$betweenness`). This information can, for example be used to evaluate which nodes sort high degree centrality scores.

```
R> head(sort(NwkSTATS$degree, decreasing = TRUE))

      Amine_9      Amine_31      Amine_50  Org.Acids_5 Org.Acids_19 Org.Acids_21
           7             7             7             7             7             7
```

Or to evaluate which nodes sort high betweenness centrality scores.

```
R> head(sort(NwkSTATS$betweenness, decreasing = TRUE))

Ox.Stress_13      Lip_64      Amine_50      Lip_50      Amine_2 Ox.Stress_29
4943.551      4212.359      3021.298      2908.500      2789.298      2574.450
```

These results convey quantitatively what we grasped qualitatively from Figure 6: Amines, and organic acids dominate the core structure (as reflected in the degree centralities). Lipids #50 and #64 have high betweenness centralities as they act as gatekeepers between the core structure and the largest peripheral lipid structure.

We might also analyze network properties from the perspective of paths. The `GGMpathStats()` function calculates path statistics for specified node pairs. The arguments `node1` and `node2` are numerics designating the nodes of interest. The function both calculates and visualizes the strongest paths. The argument `nrPaths` then indicates how many paths should be visualized in the graphical output. We could, for example, be interested in finding the two strongest paths between Amines #1 and #2.

```
R> Paths <- GGMpathStats(P0$sparsePrecision, node1 = 1, node2 = 2,
+   lay = NULL, coords = Coords, nrPaths = 2, Vsize = 4, Vcex = 0.2,
+   prune = TRUE, scale = 2, legend = FALSE)
```

The graphical output, using the node-coordinates of previous calls to `Ugraph()`, can be found in Figure 7. As in Jones and West (2005), paths whose weights have an opposite sign to the marginal covariance (between endnodes of the path) are referred to as *moderating paths* while paths whose weights have the same sign as the marginal covariance are referred to as *mediating paths*. The edges of mediating paths are represented in green while the edges of moderating paths are represented in red. Strong paths tend to be short paths.

At the group-level we might be interested in detecting communities. This can be done with the `Communities()` function. When `graph = TRUE` (default) the community structure in the graph is also visualized. The arguments for this visualization mirror a call to `Ugraph()`.

```
R> set.seed(24354)
R> Commy <- Communities(PcorP, Vcolor = Colors, Vsize = 7, Vcex = 0.3)
```

The resulting visualized community structure can be found in Figure 8. The same network now appears as was visualized in Figures 6 and 7. But now the network also expresses the community structure. The colored borders demarcate communities. The nodes are also colored according to community membership. We see two large and intertwined amine-organic acid communities. We also see various oxidative stress and lipid communities. A numerical classification of module memberships for each node can be accessed through calling `Commy$membership`.

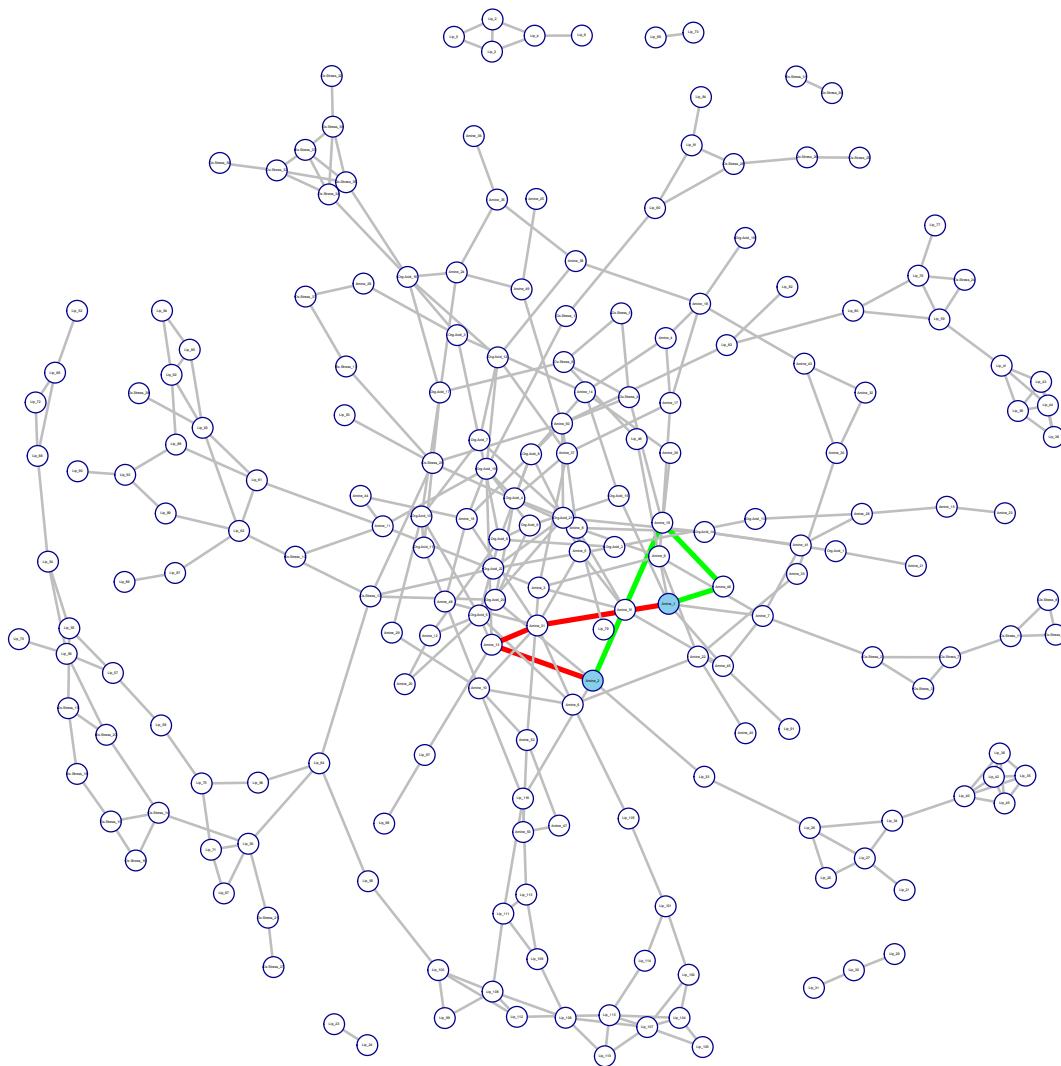


Figure 7: The metabolite network of Figure 6, but now visualized to convey the two strongest paths between Amines #1 and #2. The nodes of interest are colored blue. The edges of mediating paths are green while the edges of moderating paths are red. The strongest mediating path is Amine #1 – Amine #46 – Amine #19 – Amine #2. The strongest moderating path is Amine #1 – Amine #31 – Amine #13 – Amine #2.

### 3.2. Fused module

#### *Extraction*

We could also consider the two classes of AD patients simultaneously. We then take interest in jointly estimating the class-specific precision matrices as a basis for joint (or integrative) network modeling. The data for AD Class 1 patients (those without the genetic predisposition) consist of  $n_1 = 40$  observations on  $p = 230$  metabolic features. First, we construct lists of class-specific target matrices and class-specific data matrices. We scale the class-specific data, as the features have different scales and as the variability of the features may differ sub-

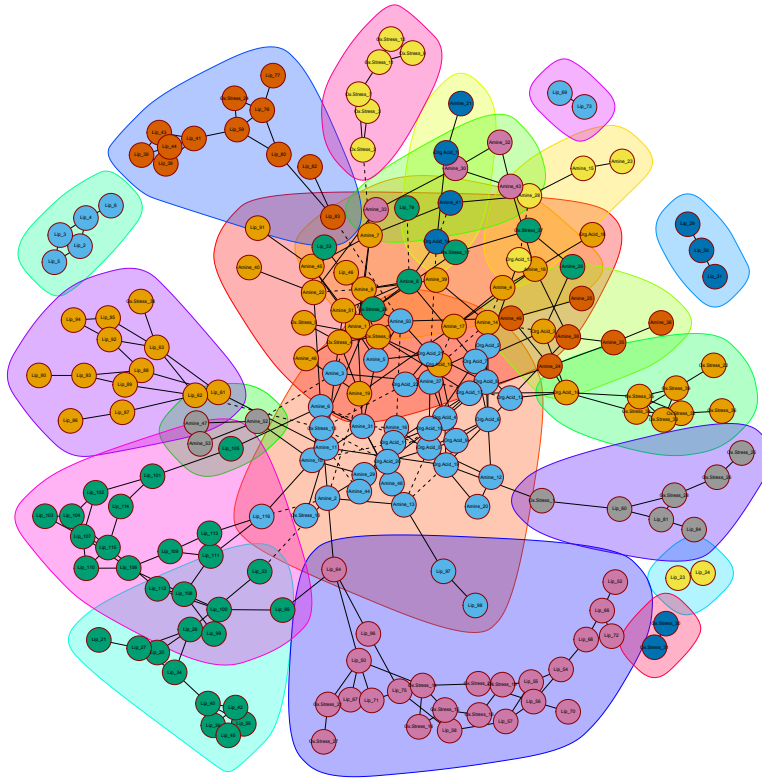


Figure 8: Community structure of the extracted metabolite network.

stantially. Again, as we have little *a priori* knowledge, we will keep the class-specific targets simple: Both classes are assigned an identity matrix as the target matrix.

```
R> ADclass1 <- ADmetabolites[, sampleInfo$ApoEClass == "Class 1"]
R> ADclass2 <- ADmetabolites[, sampleInfo$ApoEClass == "Class 2"]
R> ADclass1 <- scale(t(ADclass1))
R> ADclass2 <- scale(t(ADclass2))
R> rAD1 <- cor(ADclass1)
R> rAD2 <- cor(ADclass2)
R> Rlist <- list(rAD1 = rAD1, rAD2 = rAD2)
R> samps <- c(nrow(ADclass1), nrow(ADclass2))
R> Tlist <- default.target.fused(Slist = Rlist, ns = samps, type = "DUPV")
R> Ylist <- list(AD1data = ADclass1, AD2data = ADclass2)
```

The constructed lists will be directly used in the estimation of class-specific precision matrices. As both classes consist of patients with AD, one would expect the network topologies of the respective groups to share some of their structure, while potentially differing in a number of (topological) locations of interest. The fusion framework of Section 2.2 takes this into account explicitly.

The `optPenalty.fused()` function finds the optimal ridge and fusion penalty parameters via  $K$ -fold cross-validation of the negative fused log-likelihood score. It does so by using multi-dimensional optimization routines with automatically generated starting values. The

penalty-values at which this score is minimized are deemed optimal. The call below uses  $K = 10$  and a penalty structure specifying a ridge penalty for each class as well as a single fusion penalty (as we only have two classes in this simple example).

```
R> set.seed(8910)
R> OPTf <- optPenalty.fused(Ylist = Ylist, Tlist = Tlist,
+   lambda = as.matrix(cbind(c("ridge1", "fusion"), c("fusion", "ridge2"))),
+   cv.method = "kCV", k = 10, verbose = FALSE)
```

The class-specific precision matrices under the optimal penalties are contained in the list `OPTf$Plist`. Again, it is informative to have a look at the penalties.

```
R> OPTf$lambda.unique

      ridge1      fusion      ridge2
1.973739e+01 2.085230e-19 1.693029e+01
```

We see that the penalty values emphasize individual regularization over retainment of entry-wise similarities, indicating quite strong differences in class-specific precision matrices. We could of course use the `CNplot()` function to see that both class-specific precision matrices are well-conditioned.

After obtaining class-specific precision matrices, we again need to determine their support in order to extract class-specific networks. This is done through the `sparsify.fused()` function. Its arguments are analogous to the arguments for the `sparsify()` function. We again use lFDR thresholding with `FDRcut` set to `.999`.

```
R> POs <- sparsify.fused(OPTf$Plist, threshold = "localFDR",
+   FDRcut = 0.999, verbose = FALSE)
```

```
- Retained elements: 94
- Corresponding to 0.36 % of possible edges

- Retained elements: 289
- Corresponding to 1.1 % of possible edges
```

The `sparsify.fused()` function returns, for all classes considered, both the sparsified precision matrix and the sparsified partial correlation matrix. For the AD Class 1 network, 94 edges are retained, while for the AD Class 2 network again around 300 edges are deemed to have a high probability of being present. This could be a sign that the AD Class 1 network has a more erratic or random structure, making it harder to determine good estimates for the unknowns in the mixture distribution (3). Visualization can support a first assessment.

### *Visualization*

When visualizing multiple networks over the same features, there are some additional issues to think of (in addition to layout, coloring, etc.). One such issue is the possible retainment of coordinates over class-specific graphs. The main function for visualization, `Ugraph()`,

also supports visual comparison of class-specific networks. Reusing the layout-coordinates of previous calls enables the visualization of class-specific networks in the same coordinates. At times, this may be insightful as it allows one to visually track differential connections between class-specific networks. In order to do so for the AD Class 1 and AD Class 2 networks, we first use the `Union()` function. This convenience function subsets two square matrices (over the same features) to the union of features that have nonzero row (column) entries (i.e., to features implied in graphical connections). This allows one to prune both networks to those features that are connected in at least one of them. The AD Class 2 network will determine the coordinates for the AD Class 1 network. Again, we use node-colorings to heighten the information-density of the visualization. In addition, we use the `DiffGraph()` function, which visualizes in a single graph the edges that are unique to any two input networks. It uses edge-colorings according to class-specific presence.

```
R> TST <- Union(P0s$AD1data$sparseParCor, P0s$AD2data$sparseParCor)
R> PCclass1 <- TST$M1subset
R> PCclass2 <- TST$M2subset
R> Colors <- rownames(PCclass2)
R> Colors[grep("Amine", rownames(PCclass2))] <- "lightblue"
R> Colors[grep("Org.Acids", rownames(PCclass2))] <- "orange"
R> Colors[grep("Lip", rownames(PCclass2))] <- "yellow"
R> Colors[grep("Ox.Stress", rownames(PCclass2))] <- "purple"
R> set.seed(111213)
R> opar <- par(mfrow = c(1, 3))
R> Coords <- Ugraph(PCclass2, type = "fancy", lay = "layout_with_fr",
+   Vcolor = Colors, prune = FALSE, Vsize = 7, Vcex = 0.3,
+   main = "AD Class 2")
R> Ugraph(PCclass1, type = "fancy", lay = NULL, coords = Coords,
+   Vcolor = Colors, prune = FALSE, Vsize = 7, Vcex = 0.3,
+   main = "AD Class 1")
R> DiffGraph(PCclass1, PCclass2, lay = NULL, coords = Coords,
+   Vcolor = Colors, Vsize = 7, Vcex = 0.3,
+   main = "Differential Network")
R> par(opar)
```

The resulting visualizations can be found in Figure 9. The left-hand panel gives the metabolite network for the AD Class 2 data. It strongly resembles the network for the AD Class 2 data obtained in Section 3.1. However, now we have a slightly smaller number of connections. This is due to the fused estimation: There is some retainment of entry-wise similarities between the AD Class 1 and AD Class 2 networks. The AD Class 1 network (middle-panel) seems more erratic (less structured) and less modular at first glance. The differential network is given in the right-hand panel. Edges unique to AD Class 2 are depicted in green while edges unique to AD Class 1 are depicted in red. We can assess the seeming lack of structure of the AD Class 1 network vis-à-vis the AD Class 2 network with some simple analyses.

### *Analysis*

The analyses at the node, path and community level performed in Section 3.1 have natural extensions to the situation in which we have multiple class-specific networks. We will give

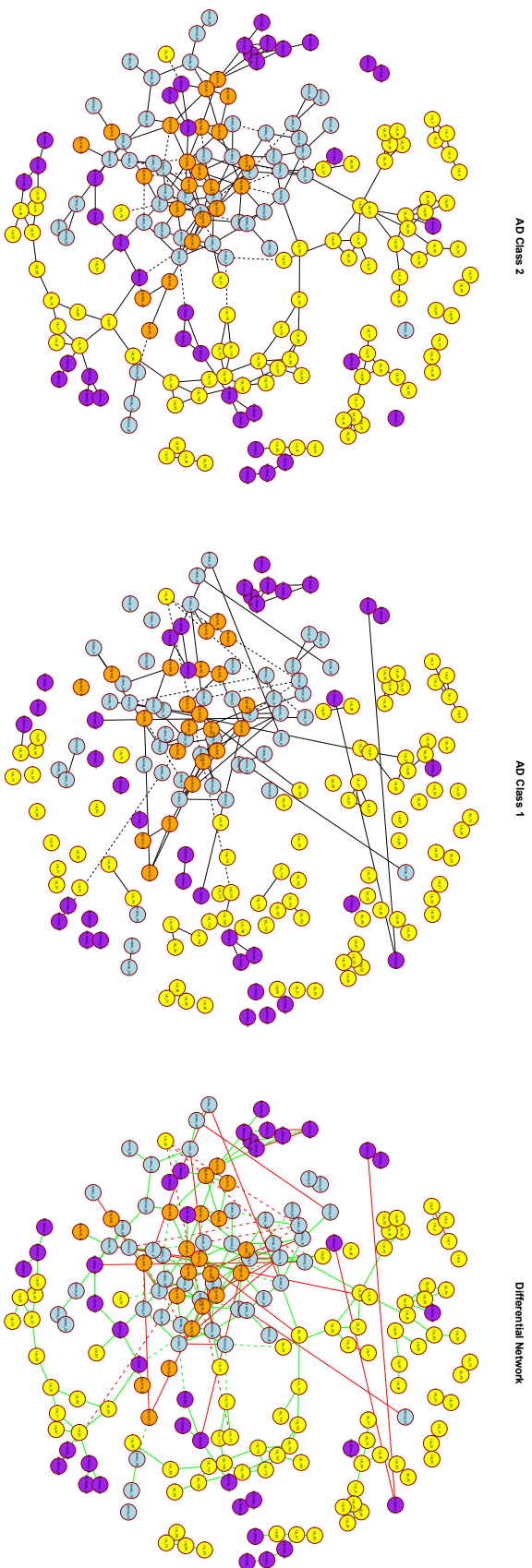


Figure 9: Left-hand panel: Metabolite network for the AD Class 2 data. Middle-panel: Metabolite network for the AD Class 1 data. Right-hand panel: Differential network between AD Classes 1 and 2. Dashed lines indicate negatively weighted edges while solid lines indicate positively weighted edges. Node colorings are according to compound family: blue for amines, orange for organic acids, yellow for lipids, and purple for oxidative stress compounds. Edges unique to AD Class 2 are depicted in green while edges unique to AD Class 1 are depicted in red.

examples for the node and community level. In addition, a simple global (or network-level) analysis is performed.

The `GGMnetworkStats.fused()` function calculates various node-level statistics from a list of sparse precision/partial correlation matrices. It returns a 'data.frame' with the slotnames of the input list prefixed to the column-names. This makes it relatively easy to, for example, compare the top node-degrees for class-specific networks.

```
R> PCOlist      <- list(PCclass1 = PCclass1, PCclass2 = PCclass2)
R> NwkSTATSList <- GGMnetworkStats.fused(PCOlist)
R> DegreesAD1  <- data.frame(rownames(NwkSTATSList),
+   NwkSTATSList$PCclass1.degree)
R> DegreesAD2  <- data.frame(rownames(NwkSTATSList),
+   NwkSTATSList$PCclass2.degree)
R> DegreesAD1o <- DegreesAD1[order(DegreesAD1[, 2], decreasing = TRUE), ]
R> DegreesAD2o <- DegreesAD2[order(DegreesAD2[, 2], decreasing = TRUE), ]
R> head(DegreesAD1o, 7)
```

```
      rownames.NwkSTATSList. NwkSTATSList.PCclass1.degree
3             Amine_3             5
7             Amine_7             4
9             Amine_9             4
13            Amine_13            4
74            Lip_4              4
167           Ox.Stress_6         4
8             Amine_8             3
```

```
R> head(DegreesAD2o, 7)
```

```
      rownames.NwkSTATSList. NwkSTATSList.PCclass2.degree
54           Org.Acid_5             7
65           Org.Acid_16            7
8            Amine_8              6
9            Amine_9              6
55           Org.Acid_6            6
68           Org.Acid_19           6
152          Lip_107              6
```

At first glance the degree distributions seem to differ. We will assess this with a simple test below.

We can apply the `Communities()` function to each class-specific sparsified matrix in order to find (and visualize) the communities for each class-specific network.

```
R> set.seed(141516)
R> opar <- par(mfrow = c(1, 2))
R> CommyC1 <- Communities(PCclass1, Vcolor = Colors, Vsize = 7, Vcex = 0.3,
+   main = "Modules AD Class 1")
```

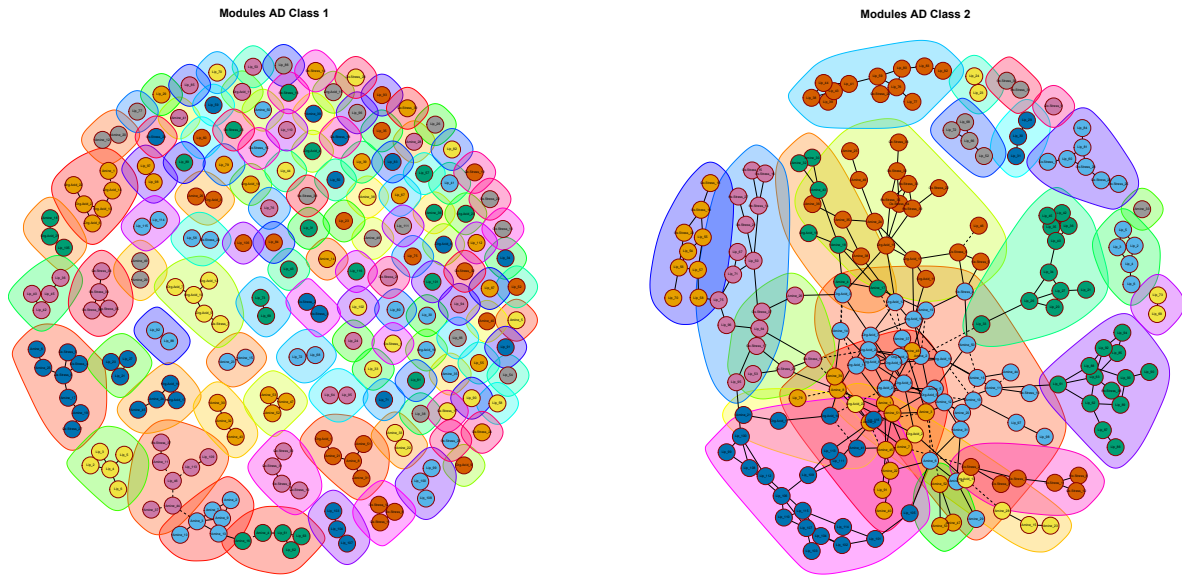


Figure 10: Left-hand panel: Community structure for the AD Class 1 data. Right-hand panel: Community structure for the AD Class 2 data.

```
R> CommyC2 <- Communities(PCclass2, Vcolor = Colors, Vsize = 7, Vcex = 0.3,
+   main = "Modules AD Class 2")
R> par(opar)
```

The visual result is given in Figure 10. The emerging picture is that the network for AD Class 2 is structured and modular while the network for AD Class 1 appears unstructured and loosely arranged.

Juxtaposing multiple networks also allows for tests on global network properties. Simple testing could be done to assess differences in degree distributions or inherent randomness of the networks. We can plot the degree distributions of the respective networks to see that these seem different.

```
R> plot(density(DegreesAD1[, 2]), col = "red", xlim = c(-1, 8),
+   xlab = "Degree", main = "")
R> lines(density(DegreesAD2[, 2]), col = "blue")
R> legenda <- c("AD class 1", "AD class 2")
R> legend(5, 0.5, legend = legenda, lwd = rep(1, 2), lty = rep(1, 2),
+   col = c("red", "blue"), cex = 0.7)
```

For the AD Class 2 network, most nodes have a similar degree (Figure 11). For the AD Class 1 network, the degree distribution seems to follow more of a power-law than a binomial distribution. We can perform a simple dependent two-group Wilcoxon Signed Rank Test to see if the distribution of the difference in degrees is symmetric about 0. We specify a one-sided alternative as we believe that the degree location of AD Class 2 is shifted to the right of the location for AD Class 1.

```
R> wilcox.test(DegreesAD1[, 2], DegreesAD2[, 2], paired = TRUE,
+   alternative = "less")
```



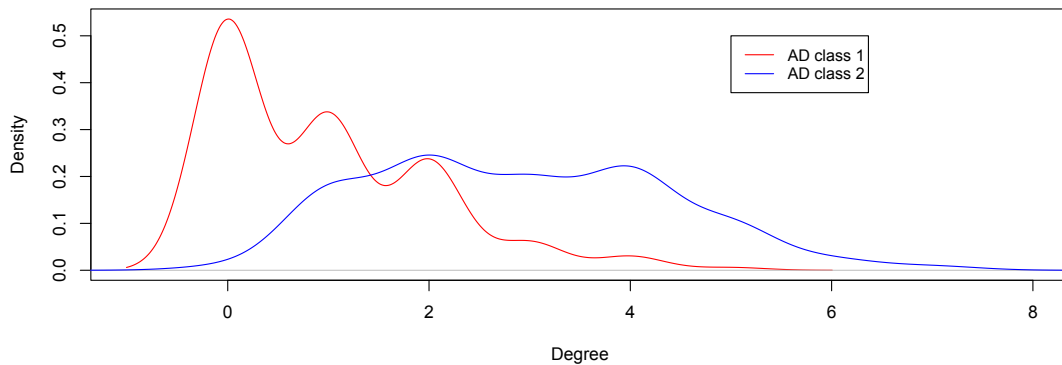


Figure 11: Degree distributions for the AD Class 1 and AD Class 2 networks.

Wilcoxon signed rank test with continuity correction

```
data: DegreesAD1[, 2] and DegreesAD2[, 2]
V = 444, p-value < 2.2e-16
alternative hypothesis: true location shift is less than 0
```

The results imply that this is indeed the case. Hence, the degree distributions imply that the network for AD Class 1 is more loosely connected. AD Class 2 has a known genetic predisposition for AD which may translate in relatively stable aberrations or amplifications across the metabolome. AD Class 1 has no known genetic predisposition for AD. The group is thus likely heterogeneous in disease aetiology, which may be a possible explanation for the lack of cohesiveness at the metabolic level.

### 3.3. Some remarks on the illustrations

At this point we want to make some final remarks regarding the illustrations. First, we have made use of uninformative target matrices. It is also possible to formulate informative target matrices. For example, [Bilgrau \*et al.\* \(2020\)](#) use pilot data in combination with pathway repository information to formulate target matrices. It is also possible to use targets in an updating scheme where the resulting precision matrix under an uninformative target becomes the new target in a subsequent round of analysis ([Van Wieringen \*et al.\* \(2020\)](#)).

Second, be aware that the graphical methods should be largely viewed as exploratory, theory or hypothesis-generating techniques. Any tentative findings they produce are ideally subjected to further research and experimentation.

## 4. Summary and discussion

The **rag2ridges** package was introduced in this paper. It takes an  $\ell_2$ -approach towards graphical modeling of high-dimensional precision matrices. This package supports the full network evaluation cycle, from extraction (through graphical modeling) to visualization and analysis. It has a modular setup and technical details as well as usage of the core and fused modules were explicated above. The core module revolves around the extraction, visualization, and analysis of single networks. The fused module extends core methodology to multiple networks.

Another module is available: the chordal module. This module extends core methodology to the extraction and analysis of graphs that can be triangulated (i.e., chordal graphs).

There is also a dependent sister package, **ragt2ridges** (Van Wieringen 2020). It provides an implementation of methodology of graphical modeling from time-course data. The time-course data are modelled by various variants of vector auto-regressive (VAR) processes. The temporal aspect of the data allows for the reconstruction of both temporal and contemporaneous relations among variates, represented graphically by directed and undirected edges, respectively. From these two types of relations, one may deduce the various types of conditional (in)dependence relationships among the variates. Like **rag2ridges**, its sister learns the VAR-model by means of targeted  $\ell_2$ -regularized estimation procedures. Internally, the reconstruction of the contemporaneous edges exploits the core module of **rag2ridges**. The down-stream functionality of the **ragt2ridges** package is, obviously, geared towards the dynamic consequences of the estimated VAR-models. Loosely, the **rag2ridges** and **ragt2ridges** packages can be thought of as one-stop- $\ell_2$ -shops for the graphical modeling of static and dynamic systems, respectively.

Future modular expansions will most likely consider functionality for robustification and directed graph inferral. A robust module is conceivable in which the core technology is extended by relaxing the assumption of Gaussianity. Another direction would be to conceive the sparsified precision or partial correlation matrix as a moralized representation of a directed graph. One could then reverse-engineer the set of directed graphs that befit, in terms of conditional independence implications, the moralized representation. We hope to offer such extensions in future updates of **rag2ridges**. In addition, we are working towards making the package more R-idiomatic.

## Computational details

The results in this paper were obtained using R 4.0.3 with the **rag2ridges** 2.2.6 package. **rag2ridges** imports from the following non-default packages: **igraph** (Csárdi 2022), **expm** (Goulet, Dutang, Maechler, Shapira, and Stadelmann 2021), **reshape** (Wickham 2007, 2022), **ggplot2** (Wickham 2016; Wickham *et al.* 2020), **Hmisc** (Harrell Jr. 2022), **fdrtool** (Klaus and Strimmer 2021), **snowfall** (Knaus 2015), **sfsmisc** (Maechler 2022), **gRbase** (Højsgaard 2022), **RBGL** (Carey, Long, and Gentleman 2021), **graph** (Gentleman, Whalen, Huber, and Falcon 2021), **Rcpp** (Eddelbuettel *et al.* 2022), and **RSpectra** (Qiu 2019). R and all mentioned packages are available from CRAN at <https://CRAN.R-project.org/>.

## Acknowledgments

This research was partly supported by Grant FP7-269553 (EpiRadBio) through the European Community's Seventh Framework Programme (FP7, 2007–2013). The Authors would like to thank two anonymous Referees whose constructive comments have led to an improvement in presentation.

## References

- Ankan A, Panda A (2015). “**pgmpy**: Probabilistic Graphical Models Using Python.” In *Proceedings of the 14th Python in Science Conference*, pp. 6–11. URL [https://conference.scipy.org/proceedings/scipy2015/pdfs/ankur\\_ankan.pdf](https://conference.scipy.org/proceedings/scipy2015/pdfs/ankur_ankan.pdf).
- Ankan A, Panda A (2021). **pgmpy**: *Probabilistic Graphical Models Using Python*. Python package version 0.1.14, URL <https://pgmpy.org/>.
- Bilgrau AE, Peeters CFW, Eriksen PS, Bøgsted M, Van Wieringen WN (2020). “Targeted Fused Ridge Estimation of Inverse Covariance Matrices from Multiple High-Dimensional Data Classes.” *Journal of Machine Learning Research*, **21**(26), 1–52.
- Blokhuis C, Peeters CFW, Cohen S, Scherpbier HJ, Kuijpers TW, Reiss P, Kootstra NA, Teunissen CE, Pajkrt D (2019). “Systemic and Intrathecal Immune Activation in Association with Cerebral and Cognitive Outcomes in Paediatric HIV.” *Scientific Reports*, **9**, 8004. doi:10.1038/s41598-019-44198-z.
- Boyle EA, Li YI, Pritchard JK (2017). “An Expanded View of Complex Traits: From Polygenic to Omnigenic.” *Cell*, **169**(7), 1177–1186. doi:10.1016/j.cell.2017.05.038.
- Brent RP (1971). “An Algorithm with Guaranteed Convergence for Finding a Zero of a Function.” *The Computer Journal*, **14**(4), 422–425. doi:10.1093/comjnl/14.4.422.
- Cai TT, Liu W, Luo X (2011). “A Constrained  $\ell_1$  Minimization Approach for Sparse Precision Matrix Estimation.” *Journal of the American Statistical Association*, **106**(494), 594–607. doi:10.1198/jasa.2011.tm10155.
- Cai TT, Liu W, Luo X (2012). **clime**: *Constrained L1-Minimization for Inverse (Covariance) Matrix Estimation*. R package version 0.4.1, URL <https://CRAN.R-project.org/package=clime>.
- Carey V, Long L, Gentleman R (2021). **RBGL**: *An Interface to the Boost Graph Library*. R package version 1.70.0, URL <https://bioconductor.org/packages/RBGL/>.
- Clavel J, Aristide L, Morlon H (2019). “A Penalized Likelihood Framework for High-Dimensional Phylogenetic Comparative Methods and an Application to New-World Monkeys Brain Evolution.” *Systematic Biology*, **68**(1), 93–116. doi:10.1093/sysbio/syy045.
- Clavel J, Morlon H (2020). “Reliable Phylogenetic Regressions for Multivariate Comparative Data: Illustration with the MANOVA and Application to the Effect of Diet on Mandible Morphology in Phyllostomid Bats.” *Systematic Biology*, **69**(5), 927–943. doi:10.1093/sysbio/syaa010.
- Csárdi G (2022). **igraph**: *Network Analysis and Visualization*. R package version 1.3.1, URL <https://CRAN.R-project.org/package=igraph>.
- Csárdi G, Nepusz T (2006). “The **igraph** Software Package for Complex Network Research.” *InterJournal, Complex Systems*(1695), 1–9.

- Danaher P (2018). **JGL**: *Performs the Joint Graphical Lasso for Sparse Inverse Covariance Estimation on Multiple Classes*. R package version 2.3.1, URL <https://CRAN.R-project.org/package=JGL>.
- Danaher P, Wang P, Witten DM (2014). “The Joint Graphical Lasso for Inverse Covariance Estimation Across Multiple Classes.” *Journal of the Royal Statistical Society B*, **76**(2), 373–397. doi:10.1111/rssb.12033.
- De Leeuw FA, Peeters CFW, Kester MI, Harms AC, Struys EA, Hankemeier T, Van Vlijmen HWT, Van der Lee SJ, Van Duijn CM, Scheltens P, Demirkan A, Van de Wiel MA, Van der Flier WM, Teunissen CE (2017). “Blood-Based Metabolic Signatures in Alzheimer’s Disease.” *Alzheimer’s & Dementia: Diagnosis, Assessment & Disease Monitoring*, **8**(1), 196–207. doi:10.1016/j.dadm.2017.07.006.
- Eddelbuettel D, François R (2011). “**Rcpp**: Seamless R and C++ Integration.” *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.
- Eddelbuettel D, François R, Allaire JJ, Ushey K, Kou Q, Russell N, Ucar I, Bates D, Chambers J (2022). **Rcpp**: *Seamless R and C++ Integration*. R package version 1.0.8.3, URL <https://CRAN.R-project.org/package=Rcpp>.
- Efron B (2010). *Large-Scale Inference: Empirical Bayes Methods for Estimation, Testing, and Prediction*. Cambridge: Cambridge University Press.
- Elsman EBM, Peeters CFW, Van Nispen RMA, Van Rens GHMB (2020). “Network Analysis of Items From the Participation and Activity Inventory for Children and Youth (PAI-CY) 7–12 Years with Visual Impairment.” *Translational Vision Science & Technology*, **9**, 19. doi:10.1167/tvst.9.6.19.
- Friedman J (1989). “Regularized Discriminant Analysis.” *Journal of the American Statistical Association*, **84**(405), 165–175. doi:10.2307/2289860.
- Friedman J, Hastie T, Tibshirani R (2008). “Sparse Inverse Covariance Estimation with the Graphical Lasso.” *Biostatistics*, **9**(3), 432–441. doi:10.1093/biostatistics/kxm045.
- Friedman J, Hastie T, Tibshirani R (2019). **glasso**: *Graphical Lasso: Estimation of Gaussian Graphical Models*. R package version 1.11, URL <https://CRAN.R-project.org/package=glasso>.
- Fruchterman TMJ, Reingold EM (1991). “Graph Drawing by Force-Directed Placement.” *Software: Practice and Experience*, **21**(11), 1129–1164. doi:10.1002/spe.4380211102.
- Gao C, Zhu Y (2016). **pGMGM**: *Estimating Multiple Gaussian Graphical Models (GGM) in Penalized Gaussian Mixture Models (GMM)*. R package version 1.0, URL <https://CRAN.R-project.org/package=pGMGM>.
- Gao C, Zhu Y, Shen X, Pan W (2016). “Estimation of Multiple Networks in Gaussian Mixture Models.” *Electronic Journal of Statistics*, **10**(1), 1133–1154. doi:10.1214/16-ejs1135.
- Gentleman R, Whalen E, Huber W, Falcon S (2021). **graph**: *A Package to Handle Graph Data Structures*. R package version 1.72.0, URL <https://bioconductor.org/packages/graph/>.

- Goulet V, Dutang C, Maechler M, Shapira M, Stadelmann M (2021). **expm**: *Matrix Exponential, Log, etc.* R package version 0.999-6, URL <https://CRAN.R-project.org/package=expm>.
- Ha MJ (2016). **GGMridge**: *Gaussian Graphical Models Using Ridge Penalty Followed by Thresholding and Reestimation.* R package version 1.1, URL <https://CRAN.R-project.org/package=GGMridge>.
- Ha MJ, Sun W (2014). “Partial Correlation Matrix Estimation Using Ridge Penalty Followed by Thresholding and Re-Estimation.” *Biometrics*, **70**(3), 762–770. doi:10.1111/biom.12186.
- Harrell Jr FE (2022). **Hmisc**: *Harrell Miscellaneous.* R package version 4.7-0, URL <https://CRAN.R-project.org/package=Hmisc>.
- Hayashi N, Yamaguchi S, Rodenburg F, Wong SY, Ujimoto K, Miki T, Iba T (2019). “Multiple Biomarkers of Sepsis Identified by Novel Time-Lapse Proteomics of Patient Serum.” *PLOS One*, **14**, e0222403. doi:10.1371/journal.pone.0222403.
- Højsgaard S (2022). **gRbase**: *A Package for Graphical Modelling in R.* R package version 1.8-7, URL <https://CRAN.R-project.org/package=gRbase>.
- Ising E (1925). “Beitrag zur Theorie des Ferromagnetismus.” *Zeitschrift für Physik*, **31**, 253–258. doi:10.1007/bf02980577.
- Jiang H, Fei X, Liu H, Roeder K, Lafferty J, Wasserman L, Li X, Zhao T (2021). **huge**: *High-Dimensional Undirected Graph Estimation.* R package version 1.3.5, URL <https://CRAN.R-project.org/package=huge>.
- Jones B, West M (2005). “Covariance Decomposition in Undirected Gaussian Graphical Models.” *Biometrika*, **92**(4), 779–786. doi:10.1093/biomet/92.4.779.
- Klaus B, Strimmer K (2021). **fdrtool**: *Estimation of (Local) False Discovery Rates and Higher Criticism.* R package version 1.2.17, URL <https://CRAN.R-project.org/package=fdrtool>.
- Knaus J (2015). **snowfall**: *Easier Cluster Computing (Based on snow).* R package version 1.84-6.1, URL <https://CRAN.R-project.org/package=snowfall>.
- Langfelder P, Horvath S (2008). “**WGCNA**: An R Package for Weighted Correlation Network Analysis.” *BMC Bioinformatics*, **9**, 559. doi:10.1186/1471-2105-9-559.
- Langfelder P, Horvath S (2012). “Fast R Functions for Robust Correlations and Hierarchical Clustering.” *Journal of Statistical Software*, **46**(11), 1–17. doi:10.18637/jss.v046.i11.
- Laska J, Narayan M (2018). **skggm**: *A scikit-Learn Compatible Package for Gaussian and Related Graphical Models.* Python package version 0.2.8, URL <https://github.com/skggm/skggm>.
- Leday GGR, Richardson S (2019). “Fast Bayesian Inference in Large Gaussian Graphical Models.” *Biometrics*, **75**(4), 1288–1298. doi:10.1111/biom.13064.

- Leday GGR, Speranza I (2020). **beam**: *Fast Bayesian Inference in Large Gaussian Graphical Models*. R package version 2.0.2, URL <https://CRAN.R-project.org/package=beam>.
- Ledoit O, Wolf M (2004). “A Well-Conditioned Estimator for Large-Dimensional Covariance Matrices.” *Journal of Multivariate Analysis*, **88**(2), 365–411. doi:10.1016/S0047-259X(03)00096-4.
- Liu H, Lafferty J, Wasserman L (2009). “The Nonparanormal: Semiparametric Estimation of High Dimensional Undirected Graphs.” *Journal of Machine Learning Research*, **10**, 2295–2328.
- Maechler M (2022). **sfsmisc**: *Utilities from ‘Seminar fuer Statistik’ ETH Zurich*. R package version 1.1-13, URL <https://CRAN.R-project.org/package=sfsmisc>.
- Mes SW, Van Velden FHP, Peltenburg B, Peeters CFW, Te Beest DE, Van de Wiel MA, Mekke J, Mulder DC, Martens RM, Castelijn JA, Pameijer FA, De Bree R, Boellaard R, Leemans R, Brakenhoff RH, De Graaf P (2020). “Outcome Prediction of Head and Neck Squamous Cell Carcinoma by MRI Radiomic Signatures.” *European Radiology*, **30**, 6311–6321. doi:10.1007/s00330-020-06962-y.
- Newman MEJ (2010). *Networks: An Introduction*. Oxford University Press, Oxford.
- Newman MEJ, Girvan M (2004). “Finding and Evaluating Community Structure in Networks.” *Physical Review E*, **69**, 026113. doi:10.1103/physreve.69.026113.
- Peeters CFW, Bilgrau AE, Van Wieringen WN (2022). **rags2ridges**: *Ridge Estimation of Precision Matrices from High-Dimensional Data*. R package version 2.2.6, URL <https://CRAN.R-project.org/package=rags2ridges>.
- Peeters CFW, Van de Wiel MA, Van Wieringen WN (2020). “The Spectral Condition Number Plot for Regularization Parameter Evaluation.” *Computational Statistics*, **35**(2), 629–646. doi:10.1007/s00180-019-00912-z.
- Price BS (2014). **RidgeFusion**: *R Package for Ridge Fusion in Statistical Learning*. R package version 1.0-3, URL <https://CRAN.R-project.org/package=RidgeFusion>.
- Price BS, Geyer CJ, Rothman AJ (2015). “Ridge Fusion in Statistical Learning.” *Journal of Computational and Graphical Statistics*, **24**(2), 439–454. doi:10.1080/10618600.2014.920709.
- Qiu Y (2019). **RSpectra**: *Solvers for Large-Scale Eigenvalue and SVD Problems*. R package version 0.16-0, URL <https://CRAN.R-project.org/package=RSpectra>.
- R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. URL <https://www.R-project.org/>.
- Schaefer J, Opgen-Rhein R, Strimmer K (2021). **GeneNet**: *Modeling and Inferring Gene Networks*. R package version 1.2.16, URL <https://CRAN.R-project.org/package=GeneNet>.
- Schäfer J, Strimmer K (2005). “A Shrinkage Approach to Large-Scale Covariance Estimation and Implications for Functional Genomics.” *Statistical Applications in Genetics and Molecular Biology*, **4**, 32. doi:10.2202/1544-6115.1175.

- Van Rossum G, et al. (2011). *Python Programming Language*. URL <https://www.python.org/>.
- Van Wieringen WN (2017). “On the Mean Squared Error of the Ridge Estimator of the Covariance and Precision Matrix.” *Statistics & Probability Letters*, **123**, 88–92. doi: [10.1016/j.spl.2016.12.002](https://doi.org/10.1016/j.spl.2016.12.002).
- Van Wieringen WN (2020). **ragt2ridges**: *Ridge Estimation of Vector Auto-Regressive (VAR) Processes*. R package version 0.3.4, URL <https://CRAN.R-project.org/package=ragt2ridges>.
- Van Wieringen WN, Peeters CFW (2015). “Application of a New Ridge Estimator of the Inverse Covariance Matrix to the Reconstruction of Gene-Gene Interaction Networks.” In C Di Serio, P Liò, A Nonis, R Tagliaferri (eds.), *Computational Intelligence Methods for Bioinformatics and Biostatistics. CIBB 2014.*, volume 8623 of *Lecture Notes in Computer Science*, pp. 170–179. Springer-Verlag. doi: [10.1007/978-3-319-24462-4\\_15](https://doi.org/10.1007/978-3-319-24462-4_15).
- Van Wieringen WN, Peeters CFW (2016). “Ridge Estimation of Inverse Covariance Matrices From High-Dimensional Data.” *Computational Statistics & Data Analysis*, **103**, 284–303. doi: [10.1016/j.csda.2016.05.012](https://doi.org/10.1016/j.csda.2016.05.012).
- Van Wieringen WN, Stam KA, Peeters CFW, Van de Wiel MA (2020). “Updating of the Gaussian Graphical Model Through Targeted Penalized Estimation.” *Journal of Multivariate Analysis*, **178**, 104621. doi: [10.1016/j.jmva.2020.104621](https://doi.org/10.1016/j.jmva.2020.104621).
- Wickham H (2007). “Reshaping Data with the **reshape** Package.” *Journal of Statistical Software*, **21**(12), 1–20. doi: [10.18637/jss.v021.i12](https://doi.org/10.18637/jss.v021.i12).
- Wickham H (2016). **ggplot2**: *Elegant Graphics for Data Analysis*. Springer-Verlag, New York. doi: [10.1007/978-0-387-98141-3](https://doi.org/10.1007/978-0-387-98141-3).
- Wickham H (2022). **reshape**: *Flexibly Reshape Data*. R package version 0.8.9, URL <https://CRAN.R-project.org/package=reshape>.
- Wickham H, Chang W, Henry L, Pedersen TL, Takahashi K, Wilke C, Woo K, Yutani H, Dunnington D (2020). **ggplot2**: *Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.3.2, URL <https://CRAN.R-project.org/package=ggplot2>.

**Affiliation:**

Carel F. W. Peeters  
Mathematical & Statistical Methods group (Biometris)  
Wageningen University & Research  
6708 PB Wageningen, The Netherlands  
E-mail: [carel.peeters@wur.nl](mailto:carel.peeters@wur.nl)  
URL: <https://cfwp.github.io/>

Anders E. Bilgrau  
Department of Mathematical Sciences  
Aalborg University  
URL: <https://bilgrau.com/>

Wessel N. van Wieringen  
Amsterdam UMC  
Vrije Universiteit Amsterdam  
Department of Epidemiology & Data Science  
Amsterdam Public Health research institute  
Amsterdam, The Netherlands  
*and*  
Department of Mathematics  
Vrije Universiteit Amsterdam  
Amsterdam, The Netherlands  
URL: <http://www.math.vu.nl/~wvanwie/>