1. Introduction

Biotechnological developments in proteomics or transcriptomics enable to produce a huge amount of data. One of the challenges for the statistician is to infer from these data the regulation network of a family of genes (or proteins). Gaussian graphical models are promising probabilistic tools to achieve this challenge.

Graphical modeling is based on the conditional independence concept: a direct relation between two variables exists if those two variables are conditionally dependent given all the remaining variables. These direct relations are represented by a graph: a node is associated to each variable and an edge is set between two nodes when they are in direct relation. In the Gaussian setting, a direct relation between two variables corresponds to a non-zero entry in the partial correlation matrix, or equivalently to a non-zero entry in the inverse of the covariance matrix.

Let us consider $p$ genes that will compose the nodes of the graph. For each gene we observe some random response such as the differential expression on microarray experiment. The $p$ nodes of the graph are thus identified with $p$ random variables denoted by $(X_1, \ldots, X_p)$ assumed
to be distributed as a multivariate Gaussian $N(0, \Sigma)$. The graph $G_{\Sigma}$ of conditional dependences is defined as follows: there exists an edge between nodes $a$ and $b$ if and only if the variables $X_a$ and $X_b$ are dependent given all the remaining variables. This will be denoted 

$$a \overset{G_{\Sigma}}{\to} b.$$ 

GGMselect [6] is dedicated to the estimation of the graph $G_{\Sigma}$ on the basis of a $n$-sample from $N(0, \Sigma)$. In the following, a graph $G$ will be identified with the set of its edges.

GGMselect is a two-stage procedure:

1. A family $\hat{G}$ of candidate graphs is built using either some data-driven method or some prior knowledge on the true graph.
2. A graph $\hat{G}$ is selected among this family $\hat{G}$ by minimizing an empirical criterion based on conditional least-squares.

GGMselect is specially designed to handle the case where the sample size $n$ is smaller than the number of variables $p$. Its performances have been assessed in [6]. It has been shown to be consistent even when $p$ is much larger than $n$, and its risk is controlled by a non-asymptotic oracle-like inequality. The assumptions needed to establish these results are weaker than those commonly used in the literature. In addition, numerical experiments have shown a nice behavior on simulated examples.

Download: http://w3.jouy.inra.fr/unites/mlaj/public/logiciels/GGMselect/

Notations. We set $\Gamma = \{1, \ldots, p\}$ and for any graph $G$ with nodes indexed by $\Gamma$, we write $d_a(G)$ for the degree of the node $a$ in the graph $G$ (which is the number of edges incident to $a$) and $\deg(G) = \max_{a \in \Gamma} d_a(G)$ for the degree of $G$. We also write $\|\|_n$ for the Euclidean norm on $\mathbb{R}^n$ divided by $\sqrt{n}$ and for any $\beta \in \mathbb{R}^p$ we define supp($\beta$) as the set of the labels $a \in \Gamma$ such that $\beta_a \neq 0$.

2. Estimation procedure

The main inputs are:

- $X$ A data matrix $X$ of size $n \times p$. Each row corresponds to an independent observation of the vector $(X_1, \ldots, X_p)$. We write $X_a$ for the $a^{th}$ column of $X$.
- $d_{max}$ A vector of $p$ integers: for each $a \in \Gamma$, $d_{\text{max}}[a]$ is the maximum degree of the node $a$ within the graphs of the family $\hat{G}$. For each $a \in \Gamma$, $d_{\text{max}}[a]$ must be smaller than $\min(n - 3, p - 1)$.
- $K$ A scale-free tuning parameter $K > 1$. Default value is $K = 2.5$.

**GGMselect Algorithm**

1. Build a (possibly data-driven) family $\hat{G}$ of candidate graphs.
2. Select $\hat{G}$ as any minimizer of Crit(.) over $\hat{G}$:

$$\hat{G} = \arg\min_{G \in \hat{G}} \text{Crit}(G).$$

Step 2 and Crit($G$) are described in Section 2.1 and the six families $\hat{G}$ of graphs available in the package are described in Section 2.2.
2.1. Penalized criterion \( \text{Crit}(\cdot) \)

For any graph \( G \) in \( \hat{G} \), we associate the \( p \times p \) matrix \( \hat{\theta}^G \) by

\[
\hat{\theta}^G = \arg\min_{\theta \in \Theta_G} \left\{ \sum_{i,a} \| X_i - X_i \theta_{i,a} \|^2, \theta \in \Theta_G \right\}.
\]

where \( \Theta_G \) is the set of \( p \times p \) matrices \( \theta \) such that \( \theta_{a,b} \) is non-zero if and only if there is an edge between \( a \) and \( b \) in \( G \). See [5] for more details.

Then, we define the criterion \( \text{Crit}(G) \) by

\[
\text{Crit}(G) = \sum_{a=1}^{p} \left\| X_a - \sum_b X_b \hat{G}_{a,b} \right\|^2 \left( 1 + \frac{\text{pen}[d_a(G)]}{n - d_a(G)} \right),
\]

where the penalty function is defined by

\[
\text{pen}(d) = K_n \left( \frac{n - d}{n - d - 1} \right) \text{EDKhi} \left[ d + 1, n - d - 1, \left( \left( \frac{p - 1}{d} \right) (d + 1)^2 \right)^{-1} \right].
\]

The function \( \text{EDKhi}[d, N, .] \) is the inverse of the function \( x \mapsto \frac{F_{d,N+2} \geq x}{d + 2} - \frac{x}{d} \frac{F_{d,N+2} \geq \frac{N + 2}{Nd} x}{N + 2} \),

where \( F_{d,N} \) denotes a Fisher random variable with \( d \) and \( N \) degrees of freedom. See [1] Sect.6.1 for details.

2.2. Families of candidate graphs \( \hat{G} \) available in GGMselect

Six families are available in GGMselect. Depending on the option family, the function \( \text{selectFast} \) uses one or several of the families \( \hat{G}_{C01}, \hat{G}_{L_A}, \hat{G}_{EW}, \hat{G}_{C01.LA}, \hat{G}_{C01.LA.EW} \). The function \( \text{selectQE} \) uses the family \( \hat{G}_{QE} \).

The user can also minimize the criterion (1) over his own family \( \hat{G} \) by using the function \( \text{selectMyFam} \).

2.2.1. C01 family \( \hat{G}_{C01} \) (with \( \text{selectFast} \))

The family \( \hat{G}_{C01} \) derives from the estimation procedure proposed in Wille and Bühlmann [8].

We write \( P(a, b|c) \) for the \( p \)-value of the likelihood ratio test of the hypothesis "cov\((X_a, X_b|X_c) = 0"\) and set

\[
P_{\max}(a, b) = \max \{ P(a, b|c), c \in \{\emptyset\} \cup \Gamma \setminus \{a, b\} \}.
\]

For any \( \alpha > 0 \), the graph \( \hat{G}_{01,\alpha} \) is defined by \( \hat{G}_{01,\alpha} \sim \hat{G}_{01,\alpha} \iff P_{\max}(a, b) \leq \alpha \) and the family \( \hat{G}_{C01} \) is the family of nested graphs

\[
\hat{G}_{C01} = \left\{ \hat{G}_{01,\alpha}, \alpha > 0 \text{ and } d_a(\hat{G}_{01,\alpha}) \leq d_{\text{max}}[a] \text{ for all } a \in \Gamma \right\}.
\]

C01 Algorithm

1. Compute the \( p(p-1)/2 \) values \( P_{\max}(a, b) \).
2. Order them.
3. Extract from these values the nested graphs \( \{ \hat{G}_{01,\alpha} : \alpha > 0 \} \).
4. Stop as soon as there is a node \( a \) for which the number of neighbours exceeds \( d_{\text{max}}[a] \).
2.2.2. Lasso-And family \( \hat{G}_{LA} \) (with selectFast)

The Lasso-And family \( \hat{G}_{LA} \) derives from the estimation procedure proposed by Meinshausen and Bühlmann [7].

For any \( \lambda > 0 \), we define the \( p \times p \) matrix \( \hat{\theta}^{\lambda} \) by

\[
\hat{\theta}^{\lambda} = \arg\min_{\theta} \left\{ \sum_{i,a} [X - X\theta]_{i,a}^2 + \lambda \sum_{a \neq b} |\theta_{a,b}|, \text{ for } \theta \in \Theta \right\},
\]

where \( \Theta \) is the set of \( p \times p \) matrices with 0 on the diagonal. Then, we define the graph \( \hat{G}_{LA}^{\lambda} \) by setting an edge between \( a \) and \( b \) if both \( \hat{\theta}_{a,b} \) and \( \hat{\theta}_{b,a} \) are non-zero. Finally, we define the family \( \hat{G}_{LA} \) as the set of graphs \( \hat{G}_{LA}^{\lambda} \) with \( \lambda \) large enough to ensure that \( d_a(\hat{G}_{LA}^{\lambda}) \leq \text{dmax}[a] \) for all \( a \in \Gamma \):

\[
\hat{G}_{LA} = \left\{ \hat{G}_{LA}^{\lambda}, \lambda > \hat{\lambda}_{\text{and,\text{dmax}}} \right\}, \quad \text{where } \hat{\lambda}_{\text{and,\text{dmax}}} = \sup \left\{ \lambda : \exists a \in \Gamma, \ d_a(\hat{G}_{LA}^{\lambda}) > \text{dmax}[a] \right\}.
\]

This family is efficiently computed with the LARS algorithm [4], see [6] Sect.2.

**LA Algorithm**

1. Compute with LARS the \( \hat{\theta}^{\lambda} \) for all the values \( \lambda \) where the support of \( \hat{\theta}^{\lambda} \) changes.
2. Compute the graphs \( \hat{G}_{LA}^{\lambda} \) for all \( \lambda > \hat{\lambda}_{\text{and,\text{dmax}}} \).

2.2.3. Adaptive lasso family \( \hat{G}_{EW} \) (with selectFast)

The family \( \hat{G}_{EW} \) is a modified version of \( \hat{G}_{LA} \) inspired by the adaptive lasso of Zou [9]. The major difference between \( \hat{G}_{EW} \) and \( \hat{G}_{LA} \) lies in the replacement of \( \sum |\theta_{a,b}| \) in (3) by \( \sum |\theta_{a,b}/\hat{\theta}_{a,b}^{\text{EW}}| \), where \( \hat{\theta}_{a,b}^{\text{EW}} \) is a preliminary estimator.

To build the family \( \hat{G}_{EW} \), we start by computing the Exponential Weight estimator \( \hat{\theta}_{a,b}^{\text{EW}} \) of [2]. For each \( a \in \Gamma \), we set \( H_a = \{ v \in \mathbb{R}^p : v_a = 0 \} \) and

\[
\hat{\theta}_{a,b}^{\text{EW}} = \int_{H_a} v e^{-\beta \|X_a - Xv\|_2^2} \prod_j \left( 1 + (v_j/\tau)^2 \right)^{-1} \frac{dv}{Z_a},
\]

with \( Z_a = \int_{H_a} e^{-\beta \|X_a - Xv\|_2^2} \prod_j \left( 1 + (v_j/\tau)^2 \right)^{-1} \frac{dv} \).

The construction of \( \hat{G}_{EW} \) is now similar to the construction of \( \hat{G}_{LA} \). For any \( \lambda > 0 \), we set

\[
\hat{\theta}_{a,b}^{\text{EW},\lambda} = \arg\min_{\theta} \left\{ \sum_{i,a} [X - X\theta]_{i,a}^2 + \lambda \sum_{a \neq b} |\theta_{a,b}/\hat{\theta}_{a,b}^{\text{EW}}|, \text{ for } \theta \in \Theta \right\},
\]

and we define the graph \( \hat{G}_{EW}^{\text{or,\lambda}} \) by setting an edge between \( a \) and \( b \) if either \( \hat{\theta}_{b,a}^{\text{EW},\lambda} \) or \( \hat{\theta}_{a,b}^{\text{EW},\lambda} \) is non-zero. Finally, the family \( \hat{G}_{EW} \) is given by

\[
\hat{G}_{EW} = \left\{ \hat{G}_{EW}^{\text{or,\lambda}}, \lambda > \hat{\lambda}_{\text{or,\text{dmax}}} \right\}, \quad \text{where } \hat{\lambda}_{\text{or,\text{dmax}}} = \sup \left\{ \lambda : \exists a \in \Gamma, \ d_a(\hat{G}_{EW}^{\text{or,\lambda}}) > \text{dmax}[a] \right\}.
\]

The Exponential Weight estimator \( \hat{\theta}_{a,b}^{\text{EW}} \) can be computed with a Langevin Monte-Carlo algorithm. We refer to Section 3.1 and Dalalyan & Tsybakov [3] for details. Once \( \hat{\theta}_{a,b}^{\text{EW}} \) is computed, the family \( \hat{G}_{EW} \) is obtained as \( \hat{G}_{LA} \) with the help of the LARS-lasso algorithm.
EW Algorithm

1. Compute $\hat{\theta}_{EW}$ with a Langevin Monte-Carlo algorithm.
2. Compute with LARS the $\hat{\theta}_{EW,\lambda}$ for all the values $\lambda$ where the support of $\hat{\theta}_{EW,\lambda}$ changes.
3. Compute the graphs $\hat{G}_{EW,\lambda}$ for all $\lambda > \hat{\lambda}_{EW}$. 

2.2.4. Mixed family $\hat{G}_{C01,LA}$ (with selectFast)

This family is defined by $\hat{G}_{C01,LA} = \hat{G}_{C01} \cup \hat{G}_{LA}$. 

2.2.5. Mixed family $\hat{G}_{C01,LA,EW}$ (with selectFast)

This family is defined by $\hat{G}_{C01,LA,EW} = \hat{G}_{C01} \cup \hat{G}_{LA} \cup \hat{G}_{EW}$. 

2.2.6. Quasi-exhaustive family $\hat{G}_{QE}$ (with selectQE)

For each node $a \in \Gamma$, we estimate the neighborhood of $a$ by $\hat{\nu}_{a} = \arg\min \left\{ \|X_{a} - \text{Proj}_{V_{S}}(X_{a})\|^{2} n + \frac{\text{pen}(|S|)}{n - |S|} : S \subset \Gamma \setminus \{a\} \text{ and } |S| \leq \text{dmax}[a] \right\}$, where $\text{pen}(\cdot)$ is the penalty function (2) and $\text{Proj}_{V_{S}}$ denotes the orthogonal projection from $\mathbb{R}^{n}$ onto $V_{S} = \{x\beta : \beta \in \mathbb{R}^{p} \text{ and supp}(\beta) = S\}$. We then build two nested graphs $\hat{G}_{and}$ and $\hat{G}_{or}$ as in Meinshausen and Bühlmann [7] 

$a \overset{G_{\text{and}}}{\sim} b \iff a \in \hat{\nu}_{b} \text{ and } b \in \hat{\nu}_{a}$ ,

$a \overset{G_{\text{or}}}{\sim} b \iff a \in \hat{\nu}_{b} \text{ or } b \in \hat{\nu}_{a}$ ,

and define the family $\hat{G}_{QE}$ as the family of all the graphs that lie between $\hat{G}_{and}$ and $\hat{G}_{or}$

$\hat{G}_{QE} = \left\{ G : \hat{G}_{and} \subset G \subset \hat{G}_{or} \text{ and } d_{a}(G) \leq \text{dmax}[a] \text{ for all } a \in \Gamma \right\}$. 

QE Algorithm

1. Compute $\hat{\nu}(a)$ for all $a \in \Gamma$. 
2. Compute the graphs $\hat{G}_{and}$ and $\hat{G}_{or}$. 
3. Work out the family $\hat{G}_{QE}$. 

3. User guide

3.1. Graph selection with selectFast

Usage: selectFast(X, dmax=min(floor(nrow(X)/3),nrow(X)-3,ncol(X)-1), K=2.5, family="EW", min.ev=10**(-8), verbose=FALSE, ...)

Main arguments:

X  \( n \times p \) matrix where \( n \) is the sample size and \( p \) the number of variables (nodes). The sample size \( n \) should be larger than 3 and the number of variables \( p \) larger than 1.

dmax integer or \( p \)-dimensional vector of integers smaller or equal to \( \min(n-3,p-1) \). When \( dmax \) is an integer, it corresponds to the maximum degree of the graphs in the family \( \hat{\mathcal{G}} \). When \( dmax \) is a \( p \)-dimensional vector, then \( dmax[a] \) corresponds to the maximum number of neighbors of \( a \) in the graphs \( G \in \hat{\mathcal{G}} \). Default value is \( \min(\text{floor}(n/3),n-3,p-1) \).

K scalar (or vector) with values larger than 1. Tuning parameter of the penalty function (2). Default value is \( K = 2.5 \) and typical values are between 1 and 3. Increasing the value of \( K \) gives more sparse graphs.

family one or several values among "C01", "LA", "EW". When family="EW" (respectively "LA", "C01"), the criterion (1) is minimized over the family \( \hat{\mathcal{G}}_{EW} \) (respectively \( \hat{\mathcal{G}}_{LA}, \hat{\mathcal{G}}_{C01} \)). In addition, when both families "C01" and "LA" are set, the criterion (1) is also minimized over the family \( \hat{\mathcal{G}}_{C01,LA} \). When the three families "C01", "LA" and "EW" are set, the criterion (1) is also minimized over the family \( \hat{\mathcal{G}}_{C01,LA,EW} \). Default value is family="EW".

Other arguments:

min.ev minimum eigenvalue for matrix inversion. The rank of the matrix is calculated as the number of eigenvalues greater than min.ev. The value of min.ev must be positive and smaller than 0.01. Default value is min.ev=10**(-8).

verbose logical. If TRUE a trace of the current process is displayed in real time.

... arguments specific to "EW" (see below).

Output: A list with components: EW, LA, C01, C01.LA, C01.LA.EW

The list EW reports the results obtained for the family EW, etc. Each list has components:

\[ \text{Neighb} \] array of size \( p \times \max(dmax) \times \text{length}(K) \). The vector Neighb\( [j,i_K] \) gives the nodes connected to \( j \) for \( K[i_K] \).

\[ \text{crit.min} \] vector of dimension \text{length}(K). It gives the minimal values of the criterion for each value of \( K \).

\[ G \] adjacency matrix with dimension \( p \times p \times \text{length}(K) \) Each slice \( G[i,,i_K] \) (for \( i_K = 1 \) to \text{length}(K)) is the adjacency matrix of the graph for \( K=i_K \).

Warning: Neighb and G are matrices if length(k)=1.

Complexity of C01 family: the complexity of selectFast with option family="C01" is of order \( np^3 \).

Complexity of LA family: the family \( \hat{\mathcal{G}}_{LA} \) is build with the help of the \LAR S package. The complexity of selectFast with option family="LA" is usually of order \( p^2n \min(n,p) \).

Complexity of EW family and choice of some specific arguments: The Exponential Weight estimator \( \hat{\theta}_{EW} \) defined by (4) is computed using a Langevin Monte Carlo algorithm introduced in [3]. This algorithm involves several parameters denoted \( T_0, h, \text{max.iter} \) and eps (see below).
The Langevin Monte Carlo algorithm is based on the formula
\[
\hat{\theta}^{\text{EW}}_a = \lim_{T \to \infty} \frac{1}{T - T_0} \int_{T_0}^{T} v_t \, dt,
\]
with \((v_t)_{t \geq 0}\) solution of the Langevin equation \(dv_t = F(v_t)dt + \sqrt{2} \, dW_t\), where \(W\) is a Brownian motion on \(H_a = \{v \in \mathbb{R}^p : v_a = 0\}\) and where \(F(v) = (F_1(v), \ldots, F_p(v))\) is defined by
\[
F_a(v) = 0 \quad \text{and} \quad F_j(v) = \frac{2 \beta}{n} [X^T (x_a - x)]_j - \frac{2 v_j}{\tau^2 + v_j^2}, \quad \text{for } j \neq a.
\]
The process \((v_t)_{t \geq 0}\) is approximated via an Euler discretization scheme:
\[
v^{(0)} = 0 \quad \text{and} \quad v^{(k+1)} = v^{(k)} + hF(v^{(k)}) + \sqrt{2h} \, W_k, \quad (5)
\]
with \(W_1, W_2, \ldots\) i.i.d. standard Gaussian random variables on \(H_a\). Then, the average \(\frac{1}{T - T_0} \int_{T_0}^{T} v_t \, dt\) is approximated by
\[
\bar{v}_T = \frac{1}{\lfloor (T - T_0)/h \rfloor} \sum_{k = [T_0/h]}^{\lfloor T/h \rfloor - 1} v^{(k)}.
\]
Finally, we set \(\hat{\theta}_a^{\text{EW}} = \bar{v}_{T_m}\), where \(T_m = (1 + m)T_0\) with \(m\) chosen as follows: the integer \(m\) is the minimum between \(\text{max.iter}\) and the smallest \(m\) such that
\[
\|\bar{v}_{T_m} - \bar{v}_{T_{m-1}}\|^2 < \text{eps} \|\bar{v}_{T_{m-1}}\|^2. \quad (6)
\]
The parameters involved in the computation of \(\hat{\theta}_a^{\text{EW}}\) are:

- **beta** positive real number. Tuning parameter \(\beta\) of \(\hat{\theta}_a^{\text{EW}}\), see (4). Default value is \(\text{beta} = n^2/2\).
- **tau** positive real number. Tuning parameter \(\tau\) of \(\hat{\theta}_a^{\text{EW}}\), see (4). Default value is \(\text{tau} = (n(p-1))^{-1/2}\).
- **h** (small) positive real number. Discretization parameter \(h\) of the Euler scheme (5). Default value is \(h=0.001\).
- **T0** positive integer. Heating parameter \(T_0\). The average \(\bar{v}_{T_m}\) is computed for times \(T_m\) multiple of \(T_0\). Default value is \(T_0=10\).
- **max.iter** positive integer. Maximal value of \(m\) for \(T_m\). When \(m\) reaches the value \(\text{max.iter}\), the parameter \(\hat{\theta}_a^{\text{EW}}\) is set to \(\bar{v}_{T_{\text{max.iter}}}\) and a warning is displayed. Default value is \(\text{max.iter}=200\).
- **eps** (small) positive real number. Tuning parameter of the convergence criterion (6). Default value is \(\text{eps}=0.01\).

**Choice of the Langevin Monte Carlo parameters h, T0, max.iter, eps:**

- The Markov process \((v^{(k)} : k \geq 0)\) can be transient when \(h\) is too large and the average \(\bar{v}_T\) then blows up. In this case, choose a smaller value for \(h\).
- When \(\text{max.iter}\) is reached you can either choose a larger \(T_0\) or increase the value of \(\text{max.iter}\). You may also choose a smaller value for \(h\).
- Choosing a smaller value for \(\text{eps}\) or \(h\) increases the precision of the computation of \(\hat{\theta}_a^{\text{EW}}\) but it can significantly slow down the algorithm.
- We refer to [3] for a discussion on the choice of the parameters \(\text{beta, tau, h, T0}\) (Section 5) and a discussion on the convergence of the Euler scheme (Section 4).

The complexity of the Langevin Monte Carlo algorithm heavily depends on the choice of the parameters \(h, T_0, \text{max.iter}, \text{eps}\). The maximum complexity is of order \(p^2 T_0/h \times \text{max.iter} + np^2\). Some examples of CPU times are given in [6] Table 1 Sect. 4.
3.2. Graph selection with `selectQE`

**Usage:**
```
selectQE(X, dmax=min(3,nrow(X)-3,ncol(X)-1), K=2.5, min.ev=10**(-8),
verbose=FALSE, max.size=10**8, max.iter=10**6, max.nG=10**8)
```

**Main arguments:**
- **X** \(n \times p\) matrix where \(n\) is the sample size and \(p\) the number of variables (nodes). The sample size \(n\) should be larger than 3 and the number of variables \(p\) larger than 1.
- **dmax** integer or \(p\)-dimensional vector of integers smaller or equal to \(\min(n-3,p-1)\). When \(dmax\) is an integer, it corresponds to the maximum degree of the graphs in the family \(\hat{G}\). When \(dmax\) is a \(p\)-dimensional vector, then \(dmax[a]\) corresponds to the maximum number of neighbors of \(a\) in the graphs \(G \in \hat{G}\). Default value is \(\min(3,n-3,p-1)\).
- **K** scalar (or vector) with values larger than 1. Tuning parameter of the penalty function (2). Default value is \(K=2.5\) and typical values are between 1 and 3. Increasing the value of \(K\) gives more sparse graphs.

**Other arguments:**
- **min.ev** minimum eigenvalue for matrix inversion. The rank of the matrix is calculated as the number of eigenvalues greater than \(\min.ev\). The value of \(\min.ev\) must be positive and smaller than 0.01. Default value is \(\min.ev=10^{-8}\).
- **verbose** logical. If \(\text{TRUE}\) a trace of the current process is displayed in real time.
- **max.size** integer. Maximum number of subsets \(S\) for estimating \(\hat{G}_{\text{and}}\) and \(\hat{G}_{\text{or}}\), see Section 2.2.6. If \(\sum_{a=1}^{p} \sum_{d=1}^{dmax[a]} C_{d}^{p-1}\) is greater than \(\text{max.size}\), then execution is stopped. Default value is \(\text{max.size}=10^{8}\).

**Output:**
- **Neighb** see `selectFast`
- **min.crit** see `selectFast`
- **G** see `selectFast`

**Complexity and choice of some advanced arguments.**

The complexity of the QE algorithm is of order \(np^{D+1}D^{3} + npD\text{card}(\hat{G}_{\text{QE}})\), where \(D\) is the maximum of \(d_{max}\). Thus, the QE algorithm cannot be used for large values of \(p\) and \(D\). Furthermore, even for moderate values of \(p\) and \(D\) the cardinality of \(\hat{G}_{\text{QE}}\) can be large and lead to memory size (and computational time) problems. In that case, the research between \(\hat{G}_{\text{and}}\) and \(\hat{G}_{\text{or}}\) is stopped and prolonged by a stepwise procedure. Let us denote by \(\hat{G}_{q}\) the collection of graphs \(G\) with \(q\) edges that belong to \(\hat{G}_{\text{QE}}\) and by \(\hat{G}_{q}\) the minimizer of \(\text{Crit}\) over \(\hat{G}_{q}\). The exhaustive search between \(\hat{G}_{\text{and}}\) and \(\hat{G}_{\text{or}}\) stops as soon as the number of graphs in \(\hat{G}_{q}\) is either greater than a threshold denoted \(\text{max.nG}\), or greater than the maximum allowed memory size. Let \(q_{\text{stop}}\) be the value of \(q\) before stopping the exhaustive procedure and let \(\hat{G}_{q_{\text{stop}}}\) be the minimizer of \(\text{Crit}\) over \(\hat{G}_{q_{\text{stop}}}\). A forward/backward stepwise procedure is taking over to explore graphs between \(\hat{G}_{q_{\text{stop}}}\) and \(\hat{G}_{or}\). Finally \(\hat{G}\) is the minimizer of \(\text{Crit}\) over \(\hat{G}_{q_{\text{stop}}}\) and the graph stemmed from the stepwise procedure.

The stepwise procedure involves the following arguments:
- **max.iter** integer. Maximum number of stepwise iterations. Default value is \(\text{max.iter}=10^{6}\).
- **max.nG** integer. Maximum number of graphs considered in the exhaustive search. Default value is \(\text{max.nG}=10^{8}\).
3.3. Graph selection with `selectMyFam`

**Usage:**
`selectMyFam(X, MyFamily, K=2.5, min.ev=10**(-8))`

**Arguments:**
- **X**
  - `n × p` matrix where `n` is the sample size and `p` the number of variables (nodes). The sample size `n` should be larger than 3 and the number of variables `p` larger than 1.
- **MyFamily**
  - List of `p × p` adjacency matrices corresponding to candidate graphs with degree less or equal to `n−3`.
- **K**
  - Scalar (or vector) with values larger than 1. Tuning parameter of the penalty function (2). Default value is `K=2.5` and typical values are between 1 and 3. Increasing the value of `K` gives more sparse graphs.
- **min.ev**
  - Minimum eigenvalue for matrix inversion. The rank of the matrix is calculated as the number of eigenvalues greater than `min.ev`. The value of `min.ev` must be positive and smaller than 0.01. Default value is `min.ev=10**(-8)`.

**Output:**
- `Neighb` see `selectFast`
- `min.crit` see `selectFast`
- `G` see `selectFast`

**Complexity:** The complexity of `selectMyFam` is of maximum order `np min(n,p) × card(MyFamily)`.

4. Auxiliary functions

4.1. Random graph generator `simulateGraph`

The function `simulateGraph` generates random covariance matrices `C` with sparse inverse `Ω`. The Gaussian law `N(0,C)` is then a sparse (non-uniform) Gaussian Graphical Model. The arguments of the function `simulateGraph` are the number of nodes `p` and two real numbers `eta` and `extraeta` between 0 and 1.

The inverse covariance matrix `Ω` is defined by `Ω = BB^T + D`, where `B` is a random sparse lower triangular matrix and `D` is a diagonal matrix with random entries sampled uniformly between 10^{-3} and 5.10^{-3}. The latter matrix `D` prevents `Ω` from having too small eigenvalues. To generate `B`, the set `{1,...,p}` is split into three consecutive sets `I_1, I_2, I_3`. For any `i < j` in the same set `I_k`, the entry `B_{i,j}` is set to 0 with probability `1 − η_{int}`, where `η_{int} = eta + (1−eta) * extraeta`. For any `i < j` belonging to two different sets, the entry `B_{i,j}` is set to 0 with probability `1 − η_{ext}`, where `η_{ext} = extraeta`. Then, the lower diagonal values that have not been set to 0 are drawn according to a uniform law on `[-1/√ε,1/√ε]` and the diagonal values are drawn according to a uniform law on `[0, √ε]`. The value `ε` is set to 1/10.

Finally, the matrix `C` is obtained by first inverting `Ω` and then rescaling this inverse in order to have 1 on the diagonal (with the function `cov2cor` of the R package `stats`).

**Usage:**
`simulateGraph(p, eta, extraeta = eta/5)`

**Arguments:**
- **p**
  - Integer. Number of rows and columns of C. Should be greater than 1.
- **eta**
  - Real number in (0,1). The proportion of edges in the three subgroups is `eta+(1−eta)*extraeta`. Small values of `eta` give sparse graphs.
- **extraeta**
  - Real number in (0,1). Proportion of edges inter groups.
4.2. **Penalty function** `penalty`

The function `penalty` computes the penalty function (2).

**Usage:** `penalty(p, n, dmax = min(3, n - 3, p - 1), K = 2.5)`

**Arguments:**
- `p` the number of variables. `p` should be greater than 1.
- `n` the sample size. `n` should be greater than 3.
- `dmax` integer or `p`-dimensional vector of integers smaller or equal to `min(n - 3, p - 1)`. When `dmax` is a scalar, it gives the maximum degree of the estimated graph. When `dmax` is a vector, `dmax[a]` gives the maximum degree of the node `a`. Default value: `min(3, n - 3, p - 1)`.
- `K` scalar or vector of real numbers larger than 1. Tuning parameter of the penalty function (2).

4.3. **Graph converter** `convertGraph`

A graph `G` can either be represented by an adjacency matrix or by an array `AG` where `AG[a,]` gives the list of all the nodes connected to the node `a` in the graph. The function `convertGraph` converts `NG` graphs represented by list of nodes into adjacency matrices. The `NG` graphs `G_1, ..., G_NG` are given as input through a single `p × Dmax × NG` array `Graph`, defined by `Graph[,1] = AG_i`. This function can be useful to generate the argument `MyFamily` of `selectMyFam`.

**Usage:** `convertGraph(Graph)`

**Argument:**
- `Graph` array of dimension `p × Dmax × NG`, where `Dmax` is the maximum degree of the `NG` graphs. When `NG` equals 1, `Graph` can be a matrix of dimension `p × Dmax`. `Graph[a,1,iG]` should be the indices of the nodes connected to the node `a`, for the graph `iG`. `Graph[a,1,iG]` should be equal to 0 if there is no node connected to the node `a`.

5. **Examples**

```r
> library("GGMselect")
> p=30
> n=30
> # ----------------------------------------
> # Random graph generator: use of simulateGraph
> # ----------------------------------------
> eta=0.11
> Gr <- simulateGraph(p,eta)
> X <- rmvnorm(n, mean=rep(0,p), sigma=Gr$C)
> # ----------------------------------------
> # Graph selection with family C01: use of selectFast
> # ----------------------------------------
> GRest <- selectFast(X, family="C01")
> # ----------------------------------------
> # Plot the result with the help of the package network
> # ----------------------------------------
> library(network)
```

```r
> gV <- network(Gr$G)
> g <- network(GRest$C01$G)
> par(mfrow=c(1,2), pty = "s")
> a <- plot(gV, usearrows = FALSE)
> title(sub="Simulated graph")
> plot(g, coord=a, usearrows = FALSE)
> title(sub="Graph selected with C01 family")
>
> # ----------------------------------------
> # Graph selection with family QE: use of selectQE
> # ----------------------------------------
> GQE <- selectQE(X)
> # ----------------------------------------
> # Plot the result
> # ----------------------------------------
```
> # ----------------------------------------
> # Graph selection with selectMyFam
> # ----------------------------------------
> # generate a family of candidate graphs with glasso
> library("glasso")
> MyFamily <- NULL
> for (j in 1:3){
+   MyFamily[[j]] <- abs(sign(glasso(cov(X),rho=j/5)$wi))
+   diag(MyFamily[[j]]) <- 0
+ }
> # select a graph within MyFamily
> GMF <- selectMyFam(X,MyFamily)
> # ----------------------------------------
> # Plot the result
> # ----------------------------------------
Simulated graph  Graph selected with MyFam

References