

GGMselect: R package for estimating Gaussian graphical models

Annie Bouvier, Christophe Giraud, Sylvie Huet, and Nicolas Verzelen

INRA, MalAGE, 78352 Jouy-en-Josas Cedex, FRANCE
e-mail: Annie.Bouvier@inra.fr

Ecole Polytechnique, CMAP, UMR 7641
Route de Saclay 91128 Palaiseau Cedex, FRANCE
e-mail: Christophe.Giraud@polytechnique.edu

INRA, MalAGE, 78352 Jouy-en-Josas Cedex, FRANCE
e-mail: Sylvie.Huet@inra.fr

Université Paris Sud, Laboratoire de Mathématiques, UMR 8628
Orsay Cedex F-91405
e-mail: nicolas.verzelen@math.u-psud.fr

Contact: Sylvie.Huet@inra.fr

Contents

1	Introduction	1
2	Estimation procedure	2
2.1	Penalized criterion $\text{Crit}(\cdot)$	3
2.2	Families of candidate graphs $\hat{\mathcal{G}}$ available in GGMselect	3
3	User guide	6
3.1	Graph selection with <code>selectFast</code>	6
3.2	Graph selection with <code>selectQE</code>	8
3.3	Graph selection with <code>selectMyFam</code>	9
4	Auxiliary functions	9
4.1	Random graph generator <code>simulateGraph</code>	9
4.2	Penalty function <code>penalty</code>	10
4.3	Graph converter <code>convertGraph</code>	10
5	Examples	10
	References	13

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, \dots, X_p) assumed

to be distributed as a multivariate Gaussian $\mathcal{N}(0, \Sigma)$. The graph G_Σ 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

$$a \stackrel{G_\Sigma}{\sim} b.$$

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

GGMselect is a two-stage procedure:

1. A family $\widehat{\mathcal{G}}$ of candidate graphs is built using either some data-driven method or some prior knowledge on the true graph.
2. A graph \widehat{G} is selected among this family $\widehat{\mathcal{G}}$ 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://genome.jouy.inra.fr/logiciels/GGMselect/>

Notations. We set $\Gamma = \{1, \dots, p\}$ and for any graph G with nodes indexed by Γ , 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 $\|\cdot\|_n$ for the Euclidean norm on \mathbb{R}^n divided by \sqrt{n} and for any $\beta \in \mathbb{R}^p$ we define $\text{supp}(\beta)$ 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, \dots, X_p) . We write x_a for the a^{th} column of X .
- dmax A vector of p integers: for each $a \in \Gamma$, $\text{dmax}[a]$ is the maximum degree of the node a within the graphs of the family $\widehat{\mathcal{G}}$. For each $a \in \Gamma$, $\text{dmax}[a]$ 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 $\widehat{\mathcal{G}}$ of candidate graphs.
2. Select \widehat{G} as any minimizer of $\text{Crit}(\cdot)$ over $\widehat{\mathcal{G}}$:

$$\widehat{G} = \underset{G \in \widehat{\mathcal{G}}}{\text{argmin}} \text{Crit}(G).$$

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

2.1. Penalized criterion $\text{Crit}(\cdot)$

For any graph G in $\widehat{\mathcal{G}}$, we associate the $p \times p$ matrix $\widehat{\theta}^G$ by

$$\widehat{\theta}^G = \operatorname{argmin} \left\{ \sum_{i,a} [\mathbf{X} - \mathbf{X}\theta]_{i,a}^2, \theta \in \Theta_G \right\},$$

where Θ_G is the set of $p \times p$ matrices θ 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

$$\text{Crit}(G) = \sum_{a=1}^p \left[\|\mathbf{X}_a - \sum_b \mathbf{X}_b \widehat{\theta}_{a,b}^G\|_n^2 \left(1 + \frac{\text{pen}[d_a(G)]}{n - d_a(G)} \right) \right], \quad (1)$$

where the penalty function is defined by

$$\text{pen}(d) = \kappa \frac{n-d}{n-d-1} \text{EDKHi} \left[d+1, n-d-1, \left(\binom{p-1}{d} (d+1)^2 \right)^{-1} \right]. \quad (2)$$

The function $\text{EDKHi}[d, N, \cdot]$ is the inverse of the function

$$x \mapsto \mathbb{P} \left(F_{d+2, N} \geq \frac{x}{d+2} \right) - \frac{x}{d} \mathbb{P} \left(F_{d, N+2} \geq \frac{N+2}{Nd} x \right),$$

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 $\widehat{\mathcal{G}}$ available in GGMselect

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

The user can also minimize the criterion (1) over his own family $\widehat{\mathcal{G}}$ by using the function `selectMyFam`.

2.2.1. C01 family $\widehat{\mathcal{G}}_{\text{C01}}$ (with `selectFast`)

The family $\widehat{\mathcal{G}}_{\text{C01}}$ 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 " $\text{cov}(X_a, X_b|X_c) = 0$ " 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 $\widehat{G}_{01,\alpha}$ is defined by $a \stackrel{\widehat{G}_{01,\alpha}}{\sim} b \iff P_{\max}(a, b) \leq \alpha$ and the family $\widehat{\mathcal{G}}_{\text{C01}}$ is the family of nested graphs

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

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 $\{\widehat{G}_{01,\alpha} : \alpha > 0\}$.
4. Stop as soon as there is a node a for which the number of neighbours exceeds $\text{dmax}[a]$.

2.2.2. Lasso-And family $\widehat{\mathcal{G}}_{LA}$ (with `selectFast`)

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

For any $\lambda > 0$, we define the $p \times p$ matrix $\widehat{\theta}^\lambda$ by

$$\widehat{\theta}^\lambda = \operatorname{argmin} \left\{ \sum_{i,a} [\mathbf{x} - \mathbf{x}\theta]_{i,a}^2 + \lambda \sum_{a \neq b} |\theta_{a,b}|, \text{ for } \theta \in \Theta \right\}, \quad (3)$$

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

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

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

LA Algorithm

1. Compute with LARS the $\widehat{\theta}^\lambda$ for all the values λ where the support of $\widehat{\theta}^\lambda$ changes.
2. Compute the graphs $\widehat{\mathcal{G}}_{\text{and}}^\lambda$ for all $\lambda > \widehat{\lambda}_{\text{and,dmax}}$.

2.2.3. Adaptive lasso family $\widehat{\mathcal{G}}_{EW}$ (with `selectFast`)

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

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

$$\widehat{\theta}_a^{EW} = \int_{H_a} v e^{-\beta \|\mathbf{x}_a - \mathbf{x}v\|_n^2} \prod_j (1 + (v_j/\tau)^2)^{-1} \frac{dv}{\mathcal{Z}_a}, \quad (4)$$

with $\mathcal{Z}_a = \int_{H_a} e^{-\beta \|\mathbf{x}_a - \mathbf{x}v\|_n^2} \prod_j (1 + (v_j/\tau)^2)^{-1} dv$ and $\beta, \tau > 0$.

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

$$\widehat{\theta}^{EW,\lambda} = \operatorname{argmin} \left\{ \sum_{i,a} [\mathbf{x} - \mathbf{x}\theta]_{i,a}^2 + \lambda \sum_{a \neq b} |\theta_{a,b}/\widehat{\theta}_{a,b}^{EW}|, \text{ for } \theta \in \Theta \right\},$$

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

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

The Exponential Weight estimator $\widehat{\theta}^{EW}$ can be computed with a Langevin Monte-Carlo algorithm. We refer to Section 3.1 and Dalalyan & Tsybakov [3] for details. Once $\widehat{\theta}^{EW}$ is computed, the family $\widehat{\mathcal{G}}_{EW}$ is obtained as $\widehat{\mathcal{G}}_{LA}$ 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 λ where the support of $\hat{\theta}^{EW,\lambda}$ changes.
3. Compute the graphs $\hat{G}_{or}^{EW,\lambda}$ for all $\lambda > \hat{\lambda}_{or,dmax}^{EW}$.

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

$$\widehat{ne}(a) = \operatorname{argmin} \left\{ \|X_a - \operatorname{Proj}_{V_S}(X_a)\|_n^2 \left(1 + \frac{\operatorname{pen}(|S|)}{n - |S|} \right) : S \subset \Gamma \setminus \{a\} \text{ and } |S| \leq \operatorname{dmax}[a] \right\},$$

where $\operatorname{pen}(\cdot)$ is the penalty function (2) and $\operatorname{Proj}_{V_S}$ denotes the orthogonal projection from \mathbb{R}^p onto $V_S = \{X\beta : \beta \in \mathbb{R}^p \text{ and } \operatorname{supp}(\beta) = S\}$. We then build two nested graphs \hat{G}_{and} and \hat{G}_{or} as in Meinshausen and Bühlmann [7]

$$\begin{aligned} a \stackrel{\hat{G}_{and}}{\sim} b &\iff a \in \widehat{ne}(b) \text{ \underline{and} } b \in \widehat{ne}(a), \\ a \stackrel{\hat{G}_{or}}{\sim} b &\iff a \in \widehat{ne}(b) \text{ \underline{or} } b \in \widehat{ne}(a), \end{aligned}$$

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

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

QE Algorithm

1. Compute $\widehat{ne}(a)$ 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.
<code>dmax</code>	integer or p -dimensional vector of integers smaller or equal to $\min(n-3, p-1)$. When <code>dmax</code> is an integer, it corresponds to the maximum degree of the graphs in the family $\hat{\mathcal{G}}$. When <code>dmax</code> is a p -dimensional vector, then <code>dmax[a]</code> 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.
<code>family</code>	one or several values among "C01", "LA", "EW". When <code>family="EW"</code> (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 <code>family="EW"</code> .

Other arguments:

<code>min.ev</code>	minimum eigenvalue for matrix inversion. The rank of the matrix is calculated as the number of eigenvalues greater than <code>min.ev</code> . The value of <code>min.ev</code> must be positive and smaller than 0.01. Default value is <code>min.ev=10**(-8)</code> .
<code>verbose</code>	logical. If TRUE a trace of the current process is displayed in real time.
<code>...</code>	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:

<code>Neighb</code>	array of size $p \times \max(\text{dmax}) \times \text{length}(K)$. The vector <code>Neighb[j, i_K]</code> gives the nodes connected to j for $K[i_K]$.
<code>crit.min</code>	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 <code>G[, i_K]</code> (for $i_K = 1$ to $\text{length}(K)$) is the adjacency matrix of the graph for $K=K[i_K]$.

Warning: `Neighb` and G are matrices if $\text{length}(K)=1$.

Complexity of C01 family: the complexity of `selecFast` 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 LARS package. The complexity of `selecFast` with option `family="LA"` is usually of order $p^2 n \min(n, p)$.

Complexity of EW family and choice of some specific arguments: The Exponential Weight estimator $\hat{\theta}_a^{EW}$ defined by (4) is computed using a Langevin Monte Carlo algorithm introduced in [3]. This algorithm involves several parameters denoted T_0 , h , `max.iter` and `eps` (see below).

The Langevin Monte Carlo algorithm is based on the formula

$$\hat{\theta}_a^{\text{EW}} = \lim_{T \rightarrow \infty} \frac{1}{T - T_0} \int_{T_0}^T v_t dt,$$

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 $F(v) = (F_1(v), \dots, F_p(v))$ is defined by

$$F_a(v) = 0 \text{ and } F_j(v) = \frac{2\beta}{n} [X^T(X_a - Xv)]_j - \frac{2v_j}{\tau^2 + v_j^2}, \text{ for } j \neq a.$$

The process $(v_t)_{t \geq 0}$ is approximated via an Euler discretization scheme:

$$v^{(0)} = 0 \text{ and } v^{(k+1)} = v^{(k)} + hF(v^{(k)}) + \sqrt{2h} W_k, \quad (5)$$

with W_1, W_2, \dots 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

$$\bar{v}_T = \frac{1}{[(T - T_0)/h]} \sum_{k=[T_0/h]}^{[T/h]-1} v^{(k)}.$$

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 `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:

<code>beta</code>	positive real number. Tuning parameter β of $\hat{\theta}_a^{\text{EW}}$, see (4). Default value is <code>beta = n²/2</code>
<code>tau</code>	positive real number. Tuning parameter τ of $\hat{\theta}_a^{\text{EW}}$, see (4). Default value is <code>tau = (n(p - 1))^{-1/2}</code>
<code>h</code>	(small) positive real number. Discretization parameter h of the Euler scheme (5). Default value is <code>h=0.001</code>
<code>T0</code>	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 <code>T0=10</code>
<code>max.iter</code>	positive integer. Maximal value of m for T_m . When m reaches the value <code>max.iter</code> , the parameter $\hat{\theta}_a^{\text{EW}}$ is set to $\bar{v}_{T_{\text{max.iter}}}$ and a warning is displayed. Default value is <code>max.iter=200</code>
<code>eps</code>	(small) positive real number. Tuning parameter of the convergence criterion (6). Default value is <code>eps=0.01</code>

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 `max.iter` is reached you can either choose a larger `T0` or increase the value of `max.iter`. You may also choose a smaller value for `h`.
- Choosing a smaller value for `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 `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`, `T0`, `max.iter`, `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 $\hat{\mathcal{G}}$. 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 $\min(3, n-3, p-1)$.
- 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 $\text{min.ev}=10^{**}(-8)$.
- verbose** logical. If TRUE a trace of the current process is displayed in real time.
- max.size** integer. Maximum number of subsets S for estimating \hat{G}_{and} and \hat{G}_{or} , see Section 2.2.6. If $\sum_{a=1}^p \sum_{d=1}^{\text{dmax}[a]} C_{p-1}^d$ is greater than **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 $np^{D+1}D^3 + npD \text{card}(\hat{\mathcal{G}}_{\text{QE}})$, where D is the maximum of **dmax**. 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 $\hat{\mathcal{G}}_{\text{QE}}$ can be large and lead to memory size (and computational time) problems. In that case, the research between \hat{G}_{and} and \hat{G}_{or} is stopped and prolonged by a stepwise procedure. Let us denote by $\hat{\mathcal{G}}_q$ the collection of graphs G with q edges that belong to $\hat{\mathcal{G}}_{\text{QE}}$ and by \hat{G}_q the minimizer of `Crit` over $\hat{\mathcal{G}}_q$. The exhaustive search between \hat{G}_{and} and \hat{G}_{or} stops as soon as the number of graphs in $\hat{\mathcal{G}}_q$ is either greater than a threshold denoted **max.nG**, or greater than the maximum allowed memory size. Let q^{stop} be the value of q before stopping the exhaustive procedure and let \hat{G}^{stop} be the minimizer of `Crit` over $\{\hat{G}_{\text{and}}\} \cup \{\hat{G}_q, q \leq q^{\text{stop}}\}$. 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 `Crit` over \hat{G}^{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:

<code>X</code>	$n \times p$ 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.
<code>MyFamily</code>	list of $p \times p$ adjacency matrices corresponding to candidate graphs with degree less or equal to $n - 3$
<code>K</code>	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.
<code>min.ev</code>	minimum eigenvalue for matrix inversion. The rank of the matrix is calculated as the number of eigenvalues greater than <code>min.ev</code> . The value of <code>min.ev</code> must be positive and smaller than 0.01. Default value is <code>min.ev=10**(-8)</code>

Output:

<code>Neighb</code>	see <code>selectFast</code> .
<code>min.crit</code>	see <code>selectFast</code> .
<code>G</code>	see <code>selectFast</code> .

Complexity: The complexity of `selectMyFam` is of maximum order $np \min(n, p) \times \text{card}(\text{MyFamily})$.

4. Auxiliary functions

4.1. Random graph generator `simulateGraph`

The function `simulateGraph` generates random covariance matrices C with sparse inverse Ω . The Gaussian law $\mathcal{N}(0, C)$ 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 $\Omega = BB^T + D$, where B is a random sparse lower triangular matrix and D is a diagonal matrix with random entries sampled uniformly between 10^{-3} and $5 \cdot 10^{-3}$. The latter matrix D prevents Ω from having too small eigenvalues. To generate B , the set $\{1, \dots, p\}$ 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 $1 - \eta_{\text{int}}$, where $\eta_{\text{int}} = \text{eta} + (1 - \text{eta}) * \text{extraeta}$. For any $i < j$ belonging to two different sets, the entry $B_{i,j}$ is set to 0 with probability $1 - \eta_{\text{ext}}$, where $\eta_{\text{ext}} = \text{extraeta}$. Then, the lower diagonal values that have not been set to 0 are drawn according to a uniform law on $[-1/\sqrt{\varepsilon}, 1/\sqrt{\varepsilon}]$ and the diagonal values are drawn according to a uniform law on $[0, \sqrt{\varepsilon}]$. 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:

<code>p</code>	integer. Number of rows and columns of C . Should be greater than 1.
<code>eta</code>	real number in $(0,1)$. The proportion of edges in the three subgroups is $\text{eta} + (1 - \text{eta}) * \text{extraeta}$. Small values of <code>eta</code> give sparse graphs.
<code>extraeta</code>	real number in $(0,1)$. Proportion of edges inter groups.

4.2. Penalty function `penalty`

The function `penalty` compute 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 A_G where $A_G[a,]$ 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, \dots, G_{NG} are given as input through a single $p \times D_{max} \times NG$ array `Graph`, defined by $Graph[, i] = A_{G_i}$. This function can be useful to generate the argument `MyFamily` of `selectMyFam`.

Usage: `convertGraph(Graph)`

Argument:

- `Graph` array of dimension $p \times D_{max} \times NG$, where D_{max} is the maximum degree of the NG graphs. When `NG` equals 1, `Graph` can be a matrix of dimension $p \times D_{max}$.
`Graph[a, 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`.

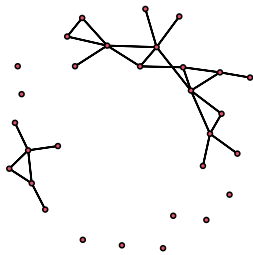
5. Examples

```
> 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)
>
```

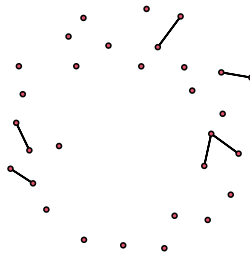
```

> 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")
>

```



Simulated graph

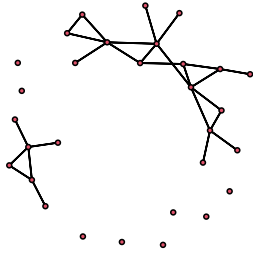


Graph selected with C01 family

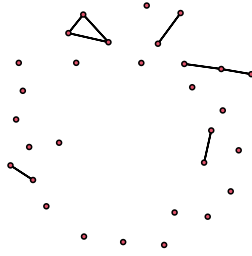
```

> # -----
> # Graph selection with family QE: use of selectQE
> # -----
> GQE <- selectQE(X)
> # -----
> # Plot the result
> # -----
>

```



Simulated graph

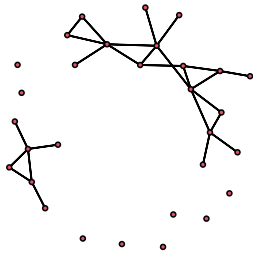


Graph selected with QE family

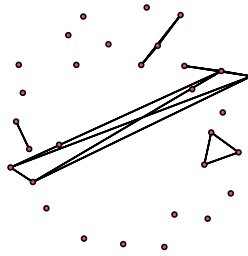
```

> # -----
> # 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

- [1] Y. Baraud, C. Giraud, and S. Huet. Gaussian model selection with an unknown variance. *Ann. Statist.*, 37(2):630–672, 2009.
- [2] A. Dalayan and A. Tsybakov. Aggregation by exponential weighting, sharp oracle inequalities and sparsity. *Machine Learning*, 72(1-2):39– 61, 2008.
- [3] A. Dalayan and A. Tsybakov. Sparse regression learning by aggregation and langevin monte-carlo. arXiv:0903.1223, 2009.
- [4] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Ann. Statist.*, 32(2):407–499, 2004. With discussion, and a rejoinder by the authors.
- [5] C. Giraud. Estimation of Gaussian graphs by model selection. *Electron. J. Stat.*, 2:542–563, 2008.
- [6] C. Giraud, S. Huet, and N. Verzelen. Graph selection with GGMselect. arXiv:0907.0619, 2009.
- [7] N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the lasso. *Ann. Statist.*, 34(3):1436–1462, 2006.
- [8] A. Wille and P. Bühlmann. Low-order conditional independence graphs for inferring genetic networks. *Stat. Appl. Genet. Mol. Biol.*, 5:Art. 1, 34 pp. (electronic), 2006.
- [9] H. Zou. The adaptive lasso and its oracle properties. *J. Amer. Statist. Assoc.*, 101(476):1418–1429, 2006.