The Stata Journal (2022)
**22**, Number 3, pp. 625–642 DOI: 10.1177/1536867X221124538

# graphiclasso: Graphical lasso for learning sparse inverse-covariance matrices

Aramayis Dallakyan
StataCorp
College Station, TX
adallakyan@stata.com

**Abstract.** In modern multivariate statistics, where high-dimensional datasets are ubiquitous, learning large (inverse-) covariance matrices is imperative for data analysis. A popular approach to estimating a large inverse-covariance matrix is to regularize the Gaussian log-likelihood function by imposing a convex penalty function. In a seminal article, Friedman, Hastie, and Tibshirani (2008, *Biostatistics* 9: 432–441) proposed a graphical lasso (Glasso) algorithm to efficiently estimate sparse inverse-covariance matrices from the convex regularized log-likelihood function. In this article, I first explore the Glasso algorithm and then introduce a new `graphiclasso` command for the large inverse-covariance matrix estimation. Moreover, I provide a useful command for tuning parameter selection in the Glasso algorithm using the extended Bayesian information criterion, the Akaike information criterion, and cross-validation. I demonstrate the use of Glasso using simulation results and real-world data analysis.

**Keywords:** st0685, graphiclasso, graphiclassocv, graphiclassoplot, datafromicov, compareicov, graphical lasso, graphical models, inverse-covariance matrix

## 1 Introduction

Recently, applications with datasets where the number of variables is higher than the number of observations are prevalent. The critical challenge in this setting is to develop a method that incorporates the complex relationships present in the dataset. Whereas the entries of a covariance matrix quantify pairwise or marginal dependence, those of the precision or inverse-covariance matrix specify multivariate relationships among the variables in a $p$-dimensional random vector $\mathbf{X} = (X_1, \ldots, X_p)^t \in R^p$ with a positive-definite covariance matrix $\mathbf{\Sigma}$. More precisely, when $\mathbf{X}$ follows a Gaussian distribution, a zero off-diagonal entry of $\mathbf{\Omega} = \mathbf{\Sigma}^{-1}$, $\omega_{jk} = 0$, implies that $X_j$ and $X_k$ are conditionally independent given all other variables (Whittaker 1990). When the number of observations $n$ is less than the number of variables $p$, it is reasonable to impose structure or regularize $\Omega$ directly in the search for sparsity (Banerjee, El Ghaoui, and d'Aspremont 2008; Friedman, Hastie, and Tibshirani 2008); see Pourahmadi (2013) for an overview.

Meinshausen and Bühlmann (2006) impose sparsity on $\mathbf{\Omega}$ by fitting a lasso model (Tibshirani 1996) to each variable, using the rest of the variables as predictors. Then, if either the estimated coefficient of variable $i$ on $j$ or the estimated coefficient of variable $j$ on $i$ is nonzero, the $\omega_{ij}$ element is estimated to be nonzero. Banerjee, El Ghaoui, and d'Aspremont (2008) and Friedman, Hastie, and Tibshirani (2008) regularize the log-

 

likelihood function by imposing an $\ell_1$ penalty on the elements of $\mathbf{\Omega}$. The graphical lasso, proposed in Friedman, Hastie, and Tibshirani (2008), uses the block coordinate descent algorithm to estimate sparse inverse-covariance matrices from the regularized likelihood function. The word "graphical" in graphical lasso (Glasso) characterizes the salient relationship between the inverse-covariance matrix and undirected graphical models. In particular, the absence of the edge between the $X_i$ and $X_j$ variables corresponds to the zero entry of the inverse-covariance matrix $\omega_{ij}$ (for example, see Whittaker [1990]). I illustrate this equivalence in section 4.

This article relies on the Glasso algorithm to introduce the `graphiclasso` command in Stata. Moreover, I provide the `graphiclassocv` command for selecting tuning parameter $\lambda$ using the extended Bayesian information criterion (eBIC) (Foygel and Drton 2010) and the cross-validation (CV) criterion.

The remainder of the article is organized as follows: Section 2 introduces the graphical lasso algorithm and methods for selecting a tuning parameter. Section 3 provides syntax for the `graphiclasso` and `graphiclassocv` commands. Section 4 contains numerical studies. Section 5 concludes with a discussion. The appendix contains syntax for two additional commands that are used to generate multivariate Gaussian data from the random inverse-covariance matrix with a prespecified sparsity level and compare the true inverse covariance with the estimated inverse-covariance matrix in terms of the true-positive rate (TPR), false-positive rate (FPR), and true-discovery rate (TDR).

## 2 Glasso

We assume sample $\mathbf{X}_1, \ldots, \mathbf{X}_n \sim N_p(\mathbf{0}, \mathbf{\Sigma})$ with the sample covariance matrix $\mathbf{S} = n^{-1} \sum_{i=1}^{n} \mathbf{X}_i \mathbf{X}_i'$. Then the log-likelihood function of data can be written as

$$\ell(\mathbf{\Omega}) = \log |\mathbf{\Omega}| - \mathrm{tr}(\mathbf{S}\mathbf{\Omega}) \tag{1}$$

where we ignore constant terms and $|\cdot|$ and $\mathrm{tr}(\cdot)$ are the determinant and the trace of the matrix, respectively. The sparsity in $\mathbf{\Omega}$ is achieved by imposing an $\ell_1$ norm penalty on (1) and maximizing the penalized log-likelihood function

$$\mathcal{L}(\mathbf{\Omega}) = \ell(\mathbf{\Omega}) - \lambda \|\mathbf{\Omega}\|_1 \tag{2}$$

where $\|\mathbf{\Omega}\|_1$ is the sum of the absolute values of elements of $\mathbf{\Omega}$ and maximization is over the space of nonnegative definite matrices. The tuning parameter $\lambda$ controls the sparsity level; that is, the larger the $\lambda$, the sparser the $\mathbf{\Omega}$. Note that the negative of (2) is a convex function of $\mathbf{\Omega}$ (Banerjee, El Ghaoui, and d'Aspremont 2008) and the global maximum is achievable.

From Karush–Kuhn–Tucker conditions, the subdifferential (Bertsekas 2016, sec. B.5) for maximizing (2) is

$$\mathbf{\Omega}^{-1} - \mathbf{S} - \lambda \mathbf{\Gamma} = \mathbf{0} \tag{3}$$

where the $\gamma_{ij}$ element of the subgradient matrix $\mathbf{\Gamma}$ takes the following form: $\gamma_{ij} = \mathrm{sign}(\omega_{ij})$ if $\omega_{ij} \neq 0$ and $\gamma_{ij} \in [-1, 1]$ if $\omega_{ij} = 0$.

Relying on the framework developed in Banerjee, El Ghaoui, and d'Aspremont (2008), Friedman, Hastie, and Tibshirani (2008) show that $\mathbf{\Omega}$ and its inverse $\mathbf{W} = \mathbf{\Omega}^{-1}$ can be iteratively estimated by solving lasso regression one row and column at a time. To illustrate, I discuss the algorithm by focusing on the last row and column.

The ingenuity of the algorithm follows from exploiting the partition of $\mathbf{W}$ and its inverse $\mathbf{\Omega}$. In particular,

$$\begin{bmatrix} \mathbf{W}_{11} & \mathbf{w}_{12} \\ \mathbf{w}_{12}^t & w_{22} \end{bmatrix} \begin{bmatrix} \mathbf{\Omega}_{11} & \boldsymbol{\omega}_{12} \\ \boldsymbol{\omega}_{12}^t & \omega_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0}^t & 1 \end{bmatrix} \tag{4}$$

and from (4)

$$\mathbf{w}_{12} = -\mathbf{W}_{11}\frac{\boldsymbol{\omega}_{12}}{\omega_{22}} = \mathbf{W}_{11}\boldsymbol{\beta} \tag{5}$$

where $\boldsymbol{\beta} = \boldsymbol{\omega}_{12}/\omega_{22}$. After a similar partition of (3), the corresponding upper right block can be written as

$$\mathbf{w}_{12} - \mathbf{s}_{12} - \lambda\boldsymbol{\gamma}_{\mathbf{12}} = \mathbf{0} \tag{6}$$

After substituting (5) into (6), we obtain

$$\mathbf{W}_{11}\boldsymbol{\beta} - \mathbf{s}_{12} + \lambda\text{sign}(\boldsymbol{\beta}) = \mathbf{0} \tag{7}$$

where we used the fact that $\boldsymbol{\beta}$ and $\boldsymbol{\omega}_{12}$ have opposite signs. After some algebra, Friedman, Hastie, and Tibshirani (2008) show that (7) is equivalent to lasso regression. For each column, authors resort to the pathwise coordinate descent algorithm (Friedman et al. 2007) to solve the modified lasso problem (7). Letting $\mathbf{V} = \mathbf{W}_{11}$, we can obtain the closed-form updates by solving for $j = 1, 2, \ldots, p-1, \ldots$ until convergence,

$$\widehat{\beta}_j = S\left(s_{12j} - \sum_{k \neq j} V_{kj}\widehat{\beta}_k, \lambda\right)/V_{jj}$$

where $S(x, \lambda) = \text{sign}(x)(|x| - \lambda)_+$ is the soft-threshold operator. I summarize the main steps of Glasso in algorithm 1.

**Algorithm 1** Glasso

1: *Input*:
2: $\mathbf{S}, \lambda \leftarrow$ Sample covariance matrix and penalty parameter
3: *Top*:
4: Initialize $\mathbf{W} = \mathbf{S} + \lambda\mathbf{I}$
5: Repeat for $j = 1, 2, \ldots, p$ until convergence
6: (a) Solve the modified lasso problem (7)
7: (b) Update $\mathbf{w}_{12} = \mathbf{W}_{11}\widehat{\boldsymbol{\beta}}$
8: In the final cycle, solve $\widehat{\boldsymbol{\omega}}_{12} = -\widehat{\boldsymbol{\beta}} \cdot \widehat{\omega}_{22}$
9: *Output*:
10: $\boldsymbol{\Omega}, \mathbf{W}$

Note that diagonal elements $w_{jj}$ of the solution matrix $\mathbf{W}$ are equal to $s_{jj} + \lambda$ and can be fixed in line 4 of algorithm 1. Alternatively, one may choose not to penalize diagonal elements of $\boldsymbol{\Omega}$ in (2). In that case, $w_{jj} = s_{jj}$, and the rest of the algorithm remains the same. As I show in section 3, the `graphiclasso` command allows such flexibility for estimation.

## 2.1 Tuning parameter selection

In real-world applications, the value of penalty parameter $\lambda$ is unknown and, traditionally, is treated as a tuning parameter to be selected from data. The value of $\lambda$ is directly connected to the sparsity of $\boldsymbol{\Omega}$; that is, the higher the $\lambda$, the sparser the inverse-covariance matrix $\boldsymbol{\Omega}$. In graphical model literature, $\lambda$ is intimately related to the Gaussian graphical model-selection problem (for example, see Hastie, Tibshirani, and Friedman [2009, chap. 17]). In this section, I discuss two popular methods for tuning parameter selection: CV and eBIC.

For $K$-fold CV, we randomly split the full dataset $\mathcal{D}$ into $K$ subsets of about the same size, denoted by $\mathcal{D}^\nu$, $\nu = 1, \ldots, K$. For each $\nu$, $\mathcal{D} - \mathcal{D}^\nu$ is used to estimate parameters and $\mathcal{D}^\nu$ to validate. The performance of the model is measured using the log-likelihood criterion. We choose the tuning parameter $\lambda$ as a minimum of the $K$-fold cross-validated log-likelihood criterion over the grid

$$\text{CV}(\lambda) = \frac{1}{K}\sum_{\nu=1}^{K}\left(-d_\nu \log|\widehat{\boldsymbol{\Omega}}_{-\nu}| + \sum_{I_\nu} y_i^t \widehat{\boldsymbol{\Omega}}_{-\nu} y_i\right) \tag{8}$$

where $\widehat{\boldsymbol{\Omega}}_{-\nu}$ is the estimated inverse-covariance matrix using the dataset $\mathcal{D} - \mathcal{D}^\nu$, $i \in I_\nu$ is the index set of the data in $\mathcal{D}$, $d_\nu$ is the size of $I_\nu$, and $y_i$ is the $i$th observation of the dataset $\mathcal{D}$.

For the Glasso, the Akaike information criterion (AIC) has the form

$$\text{AIC} = -n \left\{ \log |\mathbf{\Omega}| + \text{tr}\left(\mathbf{S}\mathbf{\Omega}\right) \right\} + E$$

where $E$ is the number of nonzero off-diagonal elements of the inverse-covariance matrix $\mathbf{\Omega}$.

Similarly, the eBIC criterion, introduced in Foygel and Drton (2010), takes the form

$$\text{eBIC}_\gamma = -n \left\{ \log |\mathbf{\Omega}| + \text{tr}\left(\mathbf{S}\mathbf{\Omega}\right) \right\} + E \log n + 4E\gamma \log p$$

The criterion is indexed by a parameter $\gamma \in [0, 1]$. It is easy to see that the $\gamma = 0$ case is the classical Bayesian information criterion (BIC) (Schwarz 1978). A positive $\gamma$ leads to the stronger penalization of large inverse-covariance matrices and results in a model-selection criterion with good theoretical properties (Foygel and Drton 2010). Resorting to simulation results, the authors suggest $\gamma = 0.5$ as a proposed value.

# 3 Commands

## 3.1 Syntax for graphiclasso

The command `graphiclasso` estimates a large inverse-covariance matrix by imposing $\ell_1$ penalization on the log-likelihood function. The observed data are supplied to `graphiclasso` either as a list of variables (*varlist*) or as a matrix (*matname*). The syntax for the Glasso algorithm is

`graphiclasso` *varlist* [ *if* ] [ *in* ] [ , *options* ]

The syntax for using a matrix as data input is

`graphiclasso` *matname* [ , *options* ]

| *options* | Description |
|---|---|
| `lambda(#)` | (nonnegative) penalty parameter; default is `lambda(0.1)` |
| `max_iter(#)` | maximum number of iterations of outer loop; default is `max_iter(100)` |
| `tolerance(#)` | maximum tolerance for convergence; default is `tolerance(1e-5)` |
| `diag` | whether diagonal should be penalized; default is false |

The penalization level is controlled through `lambda()`; the default is `lambda(0.1)`. The optimization parameters `max_iter()` and `tolerance()` control the maximum number of iterations and the maximum tolerance for convergence. Finally, `diag` specifies whether to penalize diagonal elements of the inverse-covariance matrix.

### 3.1.1 Stored results

`graphiclasso` stores the following in `e()`:

Matrices

| | |
|---|---|
| `e(lambda)` | tuning parameter |
| `e(Omega)` | inverse-covariance matrix |
| `e(Sigma)` | covariance matrix |

## 3.2 Syntax for graphiclassocv

As discussed in section 2, the tuning parameter $\lambda$ is frequently selected through CV, AIC, or eBIC. I combine these methods under one umbrella command, `graphiclassocv`, with the option to choose any of the criteria. Similarly to `graphiclasso`, the input dataset can be supplied to the command as either a *varlist* or a *matname*. The syntaxes for the `graphiclassocv` command are

`graphiclassocv` *varlist* [ *if* ] [ *in* ] [ , *options* ]

or

`graphiclassocv` *matname* [ , *options* ]

| *options* | Description |
|---|---|
| `lamlist(`*numlist*`)` | grid of positive tuning parameters for penalty term; if provided, causes `graphiclassocv` to disregard `nlam()` |
| `nlam(`#`)` | number of generated tuning parameters for penalty term; default is `nlam(20)` |
| `max_iter(`#`)` | maximum number of iterations of outer loop; default is `max_iter(1000)` |
| `tolerance(`#`)` | maximum tolerance for convergence; default is `tolerance(1e-5)` |
| `nfold(`#`)` | number of folds used for $K$-fold CV |
| `crit(`*string*`)` | CV criterion (`loglik`, `eBIC`, or `AIC`); default is `crit(loglik)` |
| `start(`*string*`)` | type of initial values; default is `start(cold)`; `start(warm)` uses the solution of the previous $\lambda$ as an initial value |
| `gamma(`#`)` | parameter for eBIC criterion; `gamma(0)` corresponds to BIC (Foygel and Drton 2010); default is `gamma(0.5)`; activated if `crit()` is `eBIC` |
| `diag` | whether diagonal should be penalized; default is false |
| `verbose` | show the table of selected information criterion |

The `graphiclassocv` command borrows most of the options from the `graphiclasso` command. Additional options are `lamlist()`, which is the list of positive $\lambda$ values, and `nlam()`, which is the number of tuning parameters $\lambda$ that should be generated for the selection. `nlam()` is activated if `lamlist()` is not provided. The `nfold()` option specifies the folds for $K$-fold CV with the default value of `nfold(5)`. The `crit()` takes one of the three options `loglik`, `AIC`, or `eBIC`. If `loglik` is specified, the tuning parameter is selected based on (8). The `start()` option specifies the selection of the initial values. If `warm` is selected, the estimated solution from the previous $\lambda$ is selected as an initial value for the Glasso algorithm.

### 3.2.1 Stored results

`graphiclassocv` stores the following in `e()`:

Scalars
- `e(lambda)` tuning parameter

Matrices
- `e(Omega)` inverse-covariance matrix
- `e(Sigma)` covariance matrix
- `e(lamlist)` list of regularization parameters

## 3.3 Syntax for visualization

I provide two options to visualize the estimated inverse-covariance matrix: either as a matrix plot or as an undirected graph (for example, see figures 1 and 2). In the matrix plot, empty cells correspond to zero elements of the inverse-covariance matrix. As discussed, for the undirected graph, zeros in the inverse-covariance matrix are equivalent to missing edges in the corresponding graph. My command heavily relies on the `nwcommands` package (Grund 2015) and accepts all `nwplot` and `nwplotmatrix` options. The syntax for the command is

`graphiclassoplot` *matname* [ `,` *options*]

| *options* | Description |
|---|---|
| `type(`*string*`)` | type of the plot: `graph` or `matrix`; default is `type(graph)` |
| `newlabs(`*lab1 lab2 ...*`)` | labels for the plot |
| *nwplot_options* | options for undirected graph plot; for details, see Grund (2015) |
| *nwplotmatrix_options* | options for matrix plot; for details, see Grund (2015) |

The `graphiclassoplot` command accepts a square matrix as an input, and the `type()` parameter accepts two options, `matrix` or `graph` (the default). The `newlabs()`, *nwplot_options*, and *nwplotmatrix_options* parameters are borrowed from the pack-

age `nwcommands` and accept all inherited corresponding options. See Grund (2015) for details.

# 4 Numerical results

In this section, I demonstrate the use of the `graphiclasso` command through a variety of simulated and real-world datasets. Note that, by design, the Glasso algorithm depends on the scaling of variables. Therefore, data standardization was performed prior to the implementation of the algorithm. For simulations, I rely on my own `datafromicov` command to generate data from the random inverse-covariance matrix with a specified sparsity level. The sparsity is controlled through probability in the off-diagonal elements of the inverse-covariance matrix being nonzero. I measure the performance of Glasso based on three metrics: TPR, FPR, and TDR. These metrics are estimated using the `compareicov` command. The syntax and details for the `datafromicov` and `compareicov` commands are provided in the appendix.

For the real-world examples, I use flow-cytometry data (Hastie, Tibshirani, and Friedman 2009, chap. 17.3) and bank stock return volatility data (Demirer et al. 2018) to illustrate the use of the `graphiclasso` command.

## 4.1 Simulation result

In all simulations, the sample sizes are $n = 50$, 150 and dimension $p = 50$, covering settings where $p = n$ and $p < n$. Each of the two simulated datasets is centered to zero and scaled to unit variance. The tuning parameter is selected from the range $[0.1, 1]$ over 30 equally spaced grid points using the BIC, eBIC, AIC, and CV criteria. As an input parameter for the Glasso algorithm, we set tolerance equal to $10^{-4}$. Each simulation setting is run over 20 repetitions. The upper left parts in figures 1 and 2 illustrate examples of the simulated inverse-covariance matrix as an undirected graph and sparse matrix, respectively. The results for eBIC are similar to BIC, so we omit them.

```
. set seed 111
. // Case n = 300, p = 50,  prob = 0.2
. local n = 300
. local p = 50
. local prb = 0.2
. // Simulate data
. datafromicov, n(`n') p(`p') prob(`prb')
number of observations will be reset to 300
Press any key to continue, or Break to abort
Number of observations (_N) was 0, now 300.
. // Extract true inverse covariance matrix
. matrix trueOmega = r(Omega)
```

```
. // Select tuning parameter via CV
. graphiclassocv var1-var50, nlam(30) nfold(3)
. // Extract the estimated Omega and lambda
. matrix cvOmega = e(Omega)
. scalar cvlambda = e(lambda)
. // Select tuning parameter via AIC
. graphiclassocv var1-var50, nlam(30) crit(AIC) nfold(3)
. // Extract the estimated Omega and lambda
. matrix aicOmega = e(Omega)
. scalar aiclambda = e(lambda)
. // Select tuning parameter via BIC
. graphiclassocv var1-var50, nlam(30) gamma(0) crit(eBIC) nfold(3)
. // Extract the estimated Omega and lambda
. matrix bicOmega = e(Omega)
. scalar biclambda = e(lambda)
. // Plot combined undirected graph
. graphiclassoplot trueOmega, type(graph) lab layout(circle)
> title(True precision matrix, position(12)) saving(trueomegagr, replace)
Calculating node coordinates...
Plotting network...
file trueomegagr.gph saved
. graphiclassoplot cvOmega, type(graph) lab layout(circle)
> title(CV, position(12)) saving(cvomegagr, replace)
Calculating node coordinates...
Plotting network...
file cvomegagr.gph saved
. graphiclassoplot bicOmega, type(graph) lab layout(circle)
> title(BIC, position(12)) saving(bicomegagr, replace)
Calculating node coordinates...
Plotting network...
file bicomegagr.gph saved
. graphiclassoplot aicOmega, type(graph) lab layout(circle)
> title(AIC, position(12)) saving(aicomegagr, replace)
Calculating node coordinates...
Plotting network...
file aicomegagr.gph saved
. graph combine "trueomegagr" "cvomegagr" "bicomegagr" "aicomegagr"
. graph export "sim1graph.png", replace
file sim1graph.png saved as PNG format
. graphiclassoplot trueOmega, type(matrix)
> title(True precision matrix, position(12)) saving(trueomega, replace)
. graphiclassoplot cvOmega, type(matrix)  title(CV, position(12))
> saving(cvomega, replace)
. graphiclassoplot bicOmega, type(matrix) title(BIC, position(12))
> saving(bicomega, replace)
. graphiclassoplot aicOmega, type(matrix) title(AIC, position(12))
> saving(aicomega, replace)
. graph combine "trueomega" "cvomega" "bicomega" "aicomega"
. graph export "sim1matrix.png", replace
file sim1matrix.png saved as PNG format
. // Now let's compare the result
. compareicov cvOmega, true(trueOmega)
```

```
. matrix cvr = r(combine)
. compareicov bicOmega, true(trueOmega)
. matrix bicr = r(combine)
. compareicov aicOmega, true(trueOmega)
. matrix aicr = r(combine)
. matrix compresult = cvr,bicr,aicr
. matrix colnames compresult = CV BIC AIC
. matrix list compresult
compresult[3,3]
            CV        BIC        AIC
tpr          1  .00952381  .88571429
fpr  .76069869          0  .00960699
tdr  .10758197          1  .89423077
```

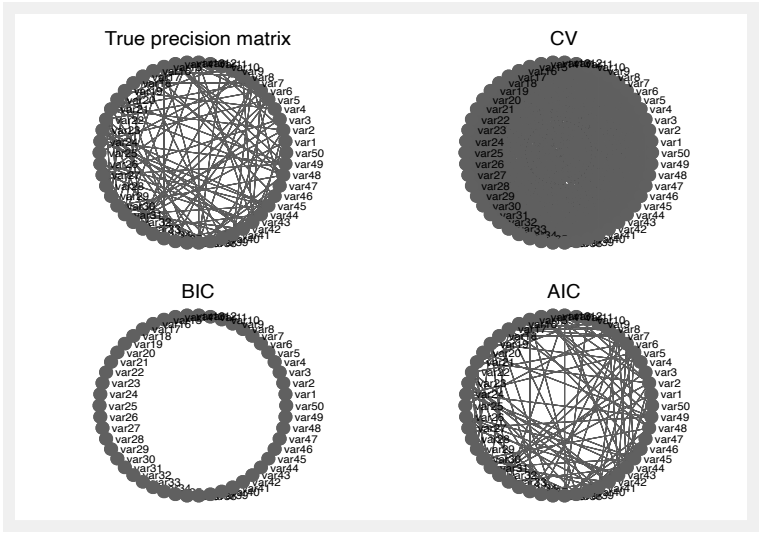


Figure 1. Illustration of Glasso solutions as undirected graphs for three different information criteria when $p = 50$, $n = 300$

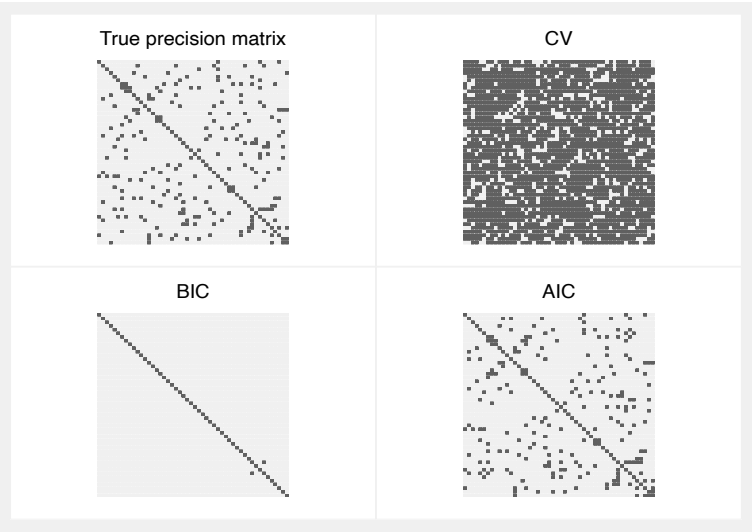


Figure 2. Illustration of Glasso solutions as a matrix for three different information criteria when $p = 50$, $n = 300$

In table 1, we report results only for $n = 300$, $p = 50$ because a similar result holds for the other simulation settings. We can see that AIC is preferred compared with the other two criteria. In particular, CV and BIC are respectively underestimating and overestimating the inverse-covariance matrix.

Table 1. Averages of three metrics over 20 simulated repetitions for the $n = 300$, $p = 50$ case

| | CV | BIC | AIC |
|---|---|---|---|
| TPR | 1 | 0.01 | 0.89 |
| FPR | 0.76 | 0 | 0.01 |
| TDR | 0.11 | 1 | 0.89 |

The values smaller than 0.001 are written as 0.

## 4.2 Flow-cytometry data

The flow-cytometry dataset, borrowed from Hastie, Tibshirani, and Friedman (2009), contains measures of 11 proteins on 7,466 cells.

```
. import delimited protain, clear
(encoding automatically selected: ISO-8859-2)
(11 vars, 7,466 obs)
. summarize
```

| Variable | Obs | Mean | Std. dev. | Min | Max |
|---|---|---|---|---|---|
| praf | 7,466 | 6.09e-06 | 247.5281 | -123.0719 | 4489.928 |
| pmek | 7,466 | -.0000317 | 377.0562 | -144.381 | 6959.619 |
| plcg | 7,466 | 3.35e-06 | 173.8598 | -53.85364 | 6153.146 |
| pip2 | 7,466 | .0000198 | 299.3475 | -150.1207 | 8906.88 |
| pip3 | 7,466 | 1.29e-06 | 43.04816 | -26.03496 | 1247.965 |
| p44_42 | 7,466 | 2.16e-06 | 45.82672 | -25.63119 | 2544.369 |
| pakts473 | 7,466 | 5.19e-06 | 137.7662 | -80.16721 | 3473.833 |
| pka | 7,466 | -.0000444 | 644.4593 | -624.7586 | 8270.241 |
| pkc | 7,466 | -3.46e-06 | 92.87002 | -29.34166 | 1580.658 |
| p38 | 7,466 | -8.18e-06 | 494.7688 | -134.0145 | 7363.985 |
| pjnk | 7,466 | -2.78e-06 | 215.6606 | -72.2675 | 4666.732 |

Figure 3 illustrates the result of applying the Glasso to the standardized flow-cytometry data for four different values of $\lambda$. As expected, the graph becomes sparse as the penalty parameter increases.

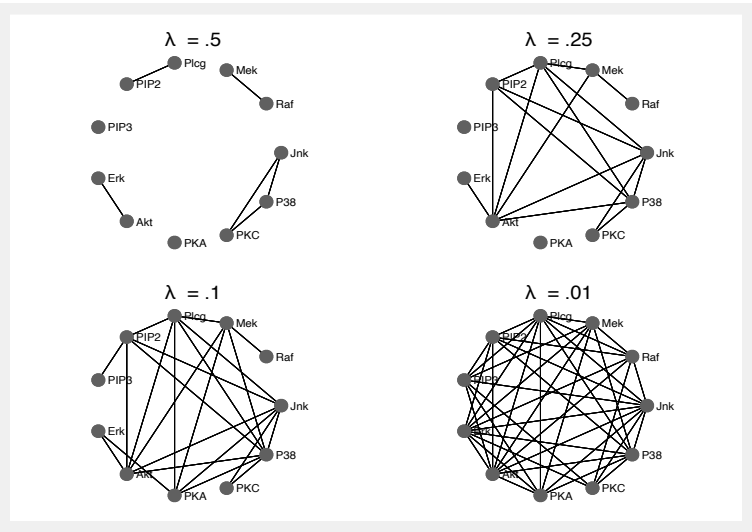


Figure 3. Glasso solutions for four different tuning parameters

Next I illustrate the estimated inverse-covariance matrix using CV and eBIC criteria.

```
. import delimited protain, clear
(encoding automatically selected: ISO-8859-2)
(11 vars, 7,466 obs)
. quietly ds
. local varlist `r(varlist)'
. foreach var in `varlist' {
  2. egen sd`var' = std(`var')
  3. }
. // Run graphiclassocv with eBIC
. graphiclassocv sdpraf-sdpjnk, gamma(0.5) nlam(20) crit(eBIC)
. matrix eBICOmega = e(Omega)
. local bic = round(e(lambda), 0.0001)
. // Run graphiclassocv with CV
. graphiclassocv sdpraf-sdpjnk, nlam(20) crit(loglik)
. matrix cvOmega = e(Omega)
. local cv = round(e(lambda), 0.0001)
. matrix lambda = `cv',`bic'
. // Plot the results
. graphiclassoplot cvOmega, type(graph) saving(cvprotaingraph,replace)
> layout(circle) newlabs("Raf" "Mek" "Plcg" "PIP2" "PIP3"
> "Erk" "Akt" "PKA" "PKC" "P38" "Jnk")
> lab title("CV, {&lambda} = `cv'")
Calculating node coordinates...
Plotting network...
file cvprotaingraph.gph saved
. graphiclassoplot eBICOmega, type(graph) saving(bicprotaingraph,replace)
> layout(circle) newlabs("Raf" "Mek" "Plcg" "PIP2" "PIP3"
> "Erk" "Akt" "PKA" "PKC" "P38" "Jnk")
> lab title("eBIC, {&lambda} = `bic'")
Calculating node coordinates...
Plotting network...
file bicprotaingraph.gph saved
. graphiclassoplot cvOmega, type(matrix) saving(cvprotainmat,replace)
. graphiclassoplot eBICOmega, type(matrix) saving(bicprotainmat,replace)
. graph combine "cvprotaingraph" "bicprotaingraph" "cvprotainmat"
> "bicprotainmat"
```

Figure 4 reports the result. The left and right columns correspond to the solution from the CV and eBIC criteria, respectively.

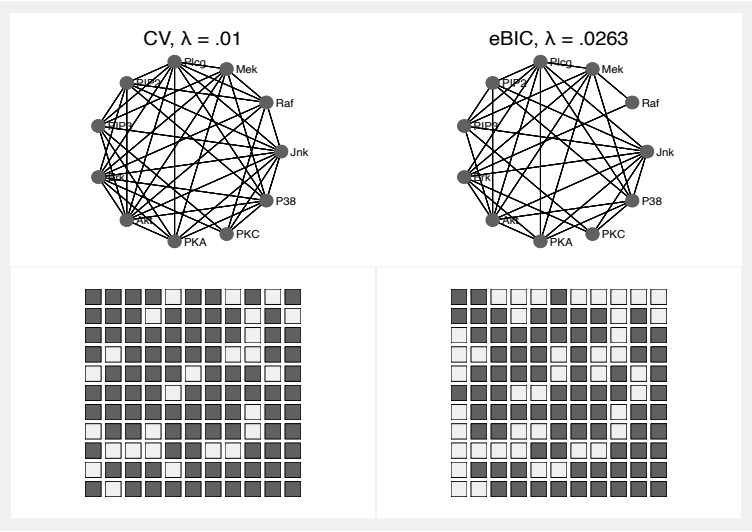


Figure 4. Glasso solutions for CV and eBIC criteria, respectively

## 4.3 Stock return volatility data

In this section, I analyze stock return volatility data. Data are borrowed from Demirer et al. (2018), where the authors estimate the global bank network connectedness. The original data contain 96 banks from 29 developed and emerging economies (countries) from September 12, 2003, to February 7, 2014. To illustrate, I select only economies where the number of banks in each economy is greater than 4; we thus have a total of 54 banks (for more details, please refer to Demirer et al. (2018).

Here I use only the AIC criterion for the tuning parameter selection. To visualize the result, I exploit a multidimensional scaling algorithm (Hastie, Tibshirani, and Friedman 2009) to calculate proximities between variables. The algorithm can be easily implemented using the `nwcommands` package. For details, see Grund (2015). Figure 5 illustrates the result. Colors in the figure (here shown in grayscale) indicate the corresponding country of the bank. As can be seen, using the estimated sparse inverse-covariance matrix as an input for the network, banks from the same country tend to compose groups, while being connected to banks from the other countries.

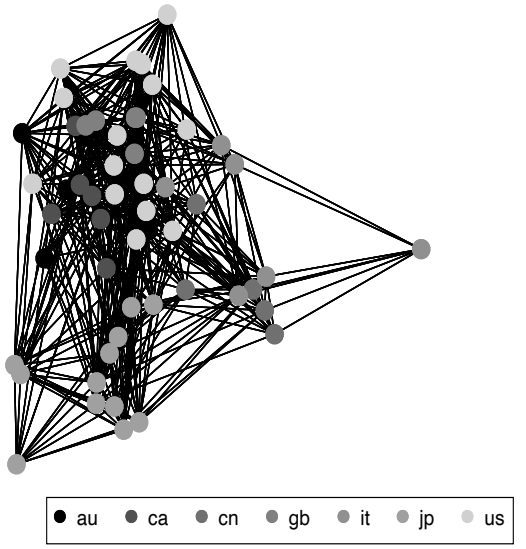


Figure 5. The Glasso solution using the AIC criterion

## 5 Conclusion

In this article, I discussed the Glasso algorithm and introduced the `graphiclasso` command for its implementation. Moreover, I provided commands for the visualization and postsimulation comparison of the estimated and the true sparse inverse-covariance matrices. I demonstrated the use of commands to analyze data when the input is a matrix or *varlist*.

For future work, I consider a possible extension of the `graphiclasso` command to implement the time-series Glasso (Dallakyan, Kim, and Pourahmadi Forthcoming) and joint Glasso (Danaher, Wang, and Witten 2014) algorithms.

## 6 Acknowledgments

I thank an anonymous referee for constructive comments and suggestions. I also thank Yulia Marchenko and Houssein Assaad from StataCorp for discussions and advice on Mata. Part of this work was done while the author was attending Texas A&M University. The views and opinions expressed in this article are those of the author. They are not endorsed by and do not necessarily reflect the position of StataCorp LLC.

# 7 Programs and supplemental materials

To install a snapshot of the corresponding software files as they existed at the time of publication of this article, type

```
. net sj 22-3
. net install st0685     (to install program files, if available)
. net get st0685         (to install ancillary files, if available)
```

# 8 References

Banerjee, O., L. El Ghaoui, and A. d'Aspremont. 2008. Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. *Journal of Machine Learning Research* 9: 485–516.

Bertsekas, D. P. 2016. *Nonlinear Programming*. 3rd ed. Belmont, MA: Athena Scientific.

Dallakyan, A., R. Kim, and M. Pourahmadi. Forthcoming. Time series graphical lasso and sparse VAR estimation. *Computational Statistics & Data Analysis*. https://doi.org/10.1016/j.csda.2022.107557.

Danaher, P., P. Wang, and D. M. Witten. 2014. The joint graphical lasso for inverse covariance estimation across multiple classes. *Journal of the Royal Statistical Society, Series B* 76: 373–397. https://doi.org/10.1111/rssb.12033.

Demirer, M., F. X. Diebold, L. Liu, and K. Yilmaz. 2018. Estimating global bank network connectedness. *Journal of Applied Econometrics* 33: 1–15. https://doi.org/10.1002/jae.2585.

Foygel, R., and M. Drton. 2010. Extended Bayesian information criteria for Gaussian graphical models. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems—Volume 1*, ed. J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, 604–612. Red Hook, NY: Curran Associates, Inc.

Friedman, J., T. Hastie, H. Höfling, and R. Tibshirani. 2007. Pathwise coordinate optimization. *Annals of Applied Statistics* 1: 302–332. https://doi.org/10.1214/07-AOAS131.

Friedman, J., T. Hastie, and R. Tibshirani. 2008. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics* 9: 432–441. https://doi.org/10.1093/biostatistics/kxm045.

Grund, T. E. 2015. nwcommands: Software tools for statistical modeling of network data in Stata. http://nwcommands.org.

Hastie, T., R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer.

Meinshausen, N., and P. Bühlmann. 2006. High-dimensional graphs and variable selection with the lasso. *Annals of Statistics* 34: 1436–1462. https://doi.org/10.1214/009053606000000281.

Pourahmadi, M. 2013. *High-Dimensional Covariance Estimation*. Hoboken, NJ: Wiley.

Rue, H. 2001. Fast sampling of Gaussian Markov random fields. *Journal of the Royal Statistical Society, Series B* 63: 325–338. https://doi.org/10.1111/1467-9868.00288.

Schwarz, G. 1978. Estimating the dimension of a model. *Annals of Statistics* 6: 461–464. https://doi.org/10.1214/aos/1176344136.

Tibshirani, R. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B* 58: 267–288. https://doi.org/10.1111/j.2517-6161.1996.tb02080.x.

Whittaker, J. 1990. *Graphical Models in Applied Multivariate Statistics*. Chichester, U.K.: Wiley.

**About the author**

Aramayis Dallakyan is Senior Statistician and Software Developer at StataCorp LLC. His research interests include graphical models, statistical and machine learning, high-dimensional time series, and computational statistics.

# A Appendix

## A.1 Additional commands

Commands `datafromicov` and `compareicov` generate multivariate Gaussian data from the random inverse-covariance matrix with a given sparsity level and compare the estimated and the true matrices using the TPR, FPR, and TDR metrics. Below, I show the syntax of those commands.

`datafromicov, n(#) p(#)` [ *options* ]

| *options* | Description |
|---|---|
| *`n(#)` | number of observations for generated data |
| *`p(#)` | number of variables (dimension) for generated data |
| `prob(#)` | probability that any off-diagonal element of inverse-covariance matrix is nonzero; default is `prob(3/`$p$`)` |
| `v(#)` | off-diagonal elements of the precision matrix, controlling the magnitude of partial correlations with `u()`; default is `v(0.3)` |
| `u(#)` | positive number being added to the diagonal elements of the precision matrix to control the magnitude of partial correlations; default is `u(0.1)` |

* `n()` and `p()` are required.

`datafromicov` stores the following in `r()`:

Scalars

| | |
|---|---|
| `r(sparsity)` | sparsity level in the inverse-covariance matrix |

Matrices

| | |
|---|---|
| `r(data)` | generated data matrix |
| `r(Omega)` | inverse-covariance matrix for the generated data |
| `r(Sigma)` | covariance matrix for the generated data |
| `r(S)` | empirical covariance matrix for the generated data |

The required inputs for the command are the number of observations $n$ and the number of variables $p$ of the generated data. General steps for data generation are the following: We start with the $p \times p$ zero matrix $\mathbf{\Omega}$ and assign its off-diagonal elements to value 1 with probability `prob()`. The default is `prob(3/`$p$`)`. The value of `prob()` controls the sparseness of the inverse-covariance matrix; that is, the lower the `prob()`, the sparser the matrix, and vice versa. Then, in the next step, nonzero elements are filled with values generated from the Uniform$[0, 0.5]$ distribution. To control the magnitude of partial correlations, we multiply the $\mathbf{\Omega}$ matrix with the positive number $v$. The default value for $v$ is 0.3. The diagonal elements of the $\mathbf{\Omega}$ are filled with $\lambda_{\min}(\mathbf{\Omega})+u$, where $\lambda_{\min}$ is the minimum eigenvalue and $u$ is a positive number added to the diagonal elements. Finally, multivariate Gaussian data can be sampled from the inverse-covariance matrix through the Cholesky factor. For a fast sampling technique, see, for example, Rue (2001).

Next I discuss the `compareicov` syntax.

`compareicov` *matname1*`, true(`*matname2*`)`

`compareicov` stores the following in `r()`:

Scalars

| | |
|---|---|
| `r(TPR)` | TPR |
| `r(FPR)` | FPR |
| `r(TDR)` | TDR |

`compareicov` takes two inputs: the $p \times p$ dimensional inverse-covariance matrix, which can be output from the `graphiclasso` command, and the true matrix. It reports the TPR, FPR, and TDR such that

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$
$$\text{TDR} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

where TP = number of estimated edges that are also present in the true graph, FN = number of estimated gaps that are present in the true graph, FP = number of estimated edges that are not present in the true graph, and TN = number of estimated gaps that are not present in the true graph.