An Introduction to `cxreg`

Younghoon Kim Navonil Deb Sumanta Basu

September 07, 2025

Introduction

`cxreg` is a package that fits complex-valued penalized regressions (Lasso) and Gaussian likelihoods (graphical Lasso) using an exact pathwise coordinate descent method. Similar to the well-known package `glmnet` (2010) for Lasso in real value settings, `cxreg` computes the regularization path for complex-valued linear regression with a lasso penalty, referred to as `classo`, over a grid of values for the regularization parameter lambda. Likewise, to fit complex-valued Graphical Lasso models (2008), the regularization path is computed via `cglasso`. The package includes methods for prediction, cross-validation functions, printing and plotting utilities, as well as fitting for both model types.

The authors of `cxreg` are Navonil Deb and Sumanta Basu, with contributions from Younghoon Kim. The R package is maintained by Younghoon Kim.

This vignette describes basic usage of functions related to `classo` and `cglasso` models in `cxreg` package in R. There is an original description of the algorithm that should be useful:

- "Regularized estimation of sparse spectral precision matrices"

`classo` solves the following problem: For $Y \in \mathbb{C}^n$ and $X \in \mathbb{C}^{n\times p}$,

$$\min_{\beta\in\mathbb{C}^p} \frac{1}{2n}\|Y - X\beta\|_2^2 + \lambda\|\beta\|_1,$$

over a grid of values of λ covering the entire range of possible solutions. Here, since the absolute of complex-values means a complex modulus, $\beta \in \mathbb{C}^p$ so that the ℓ_1-penalty can be viewed as

$$\|\beta\|_1 = \sum_{j=1}^{p} |\beta_j| = \sum_{j=1}^{p} \|\mathrm{Re}(\beta_j) + \mathrm{Im}(\beta_j)\|_2,$$

which is a group Lasso with p groups, each of size 2.

Similarly, `cglasso` seeks the minimizer of Whittle's approximate likelihood (1951). For a p-dimensional Gaussian time series $X_{t\,t=1}^{n}$, one can compute the p-dimensional complex-valued discrete Fourier transforms (DFT) at the Fourier frequencies $\omega_j = 2\pi j/n$, where $j \in F_n := \left\{-\left[\frac{n-1}{2}\right], \ldots, \left[\frac{n}{2}\right]\right\}$:

$$d_j := d(\omega_j) = \frac{1}{\sqrt{n}} \sum_{t=1}^{n} X_t \exp(-it\omega_j).$$

Note that $d_j \sim \mathcal{N}_{\mathbb{C}}(0, f(\omega_j))$, where $f(\omega_j)$ is the spectral density, commonly estimated by averaging the periodogram $I(\omega_j) = d(\omega_j)d^{\dagger}(\omega_j)$ over a bandwidth of $2m+1$ frequencies:

$$\hat{f}(\omega_j) = \frac{1}{2\pi(2m+1)} \sum_{|k|\leq m} I(\omega_{j+k}), \quad j \in F_n. \tag{1}$$

The approximation to the negative log-likelihood then takes the form:

$$\sum_{k=j-m}^{j+m} \log\det f^{-1}(\omega_j) - \sum_{k=j-m}^{j+m} d_k^\dagger f^{-1}(\omega_j) d_k.$$

Finally, by rearranging the components used to construct $\hat{f}(\omega_j)$ and adding an ℓ_1-penalty on the off-diagonal entries of the inverse spectral density, $\|\Theta\|_{1,\text{off}} = \sum_{k\neq\ell} |\Theta_{k,\ell}| = \sum_{k\neq\ell} \|\text{Re}(\Theta_{k,\ell}) + \text{Im}(\Theta_{k,\ell})\|_2$, one obtains the estimator:

$$\hat{\Theta}_j := \hat{\Theta}(\omega_j) = \arg\min_{\Theta\in\mathcal{H}_{++}^p} \left\{ \text{trace}(\Theta \hat{f}(\omega_j)) - \log\det\Theta + \lambda\|\Theta\|_{1,\text{off}} \right\},$$

where $\mathcal{H}_{++}^p$ represents the set of $p \times p$ symmetric positive definite matrices. Throughout the illustrative example in this document, we drop the subscript j and focus on deriving $\hat{\Theta}$ for a given $\hat{f}$.

From a numerical standpoint, there are two formulations for complex-valued graphical Lasso (see Section 5 in (2024)), where a similar argument can be made in standard graphical Lasso (e.g., (2018)).

CGLASSO-sc1: The first formulation is to solve the complex-valued graphical Lasso with scaled spectral density matrix (called spectral coherence), $\hat{R} = D^{-1}\hat{f}D^{-1}$ with $D^2 = \text{diag}(\hat{f}_{1,1}, \ldots, \hat{f}_{p,p})$, and scale back once the estimates are obtained. The corresponding optimization problem is

$$\hat{\Theta} = D^{-1}\hat{K}D^{-1}, \quad \hat{K} = \arg\min_{\Theta\in\mathcal{H}_{++}^p} \left\{ \text{trace}(\hat{R}\Theta) - \log\det\Theta + \lambda\|\Theta\|_{1,\text{off}} \right\}. \tag{CGLASSO-sc1}$$

This is equivalent to a equation (9) in (2018) for the graphical lasso with weighted penalty. The entry-wise weights determined by D as $\lambda_{k,\ell} \propto D_{k,k}D_{\ell,\ell} = \sqrt{\hat{f}_{k,k}\hat{f}_{\ell,\ell}}$.

CGLASSO-sc2: The second formulation takes the spectral density as the original input. In each iteration of `CLASSO.COV` within Algorithm 3 in (2024), we scale the partitioned W, working inverse spectral density, with the corresponding diagonal entries of the estimated spectral density. The Lasso outputs are scaled back to obtain rows and columns of W. Specifically, we implement the following update for kth row and column:

$$\begin{aligned} &W_{11}^{\text{scl}} \leftarrow D_{11}^{-1}W_{11}D_{11}^{-1}, \quad D_{11}^2 = \text{diag}(W_{11}) = \text{diag}(P_{11}), \quad \mathbf{p}_{12}^{\text{scl}} \leftarrow D_{11}^{-1}\mathbf{p}_{12}, \\ &\hat{\beta}^{\text{scl}} = \text{CLASSO.COV}(W_{11}^{\text{scl}}, \mathbf{p}_{12}^{\text{scl}}, \mathcal{B}_{\cdot,k}^{(0)}, \lambda), \\ &\mathcal{B}_{\cdot,k}^{(0)} \leftarrow \hat{\beta}^{\text{scl}}, \quad \hat{\beta} \leftarrow D_{11}^{-1}\hat{\beta}^{\text{scl}}, \quad \mathbf{w}_{12} \leftarrow W_{11}\hat{\beta}, \end{aligned} \tag{CGLASSO-sc2}$$

where the scaling matrix D_{11} is similar to D is CGLASSO-sc1, except that the last diagonal entry $D_{p,p}$ is disregarded in every update due to absence of the last row and column.

Installation

Like other R packages on Github, it can be downloaded by the command:

```
library(devtools)
devtools::install_github("yk748/cxreg")
```

Example: classo

The purpose of this section is to give users a general sense of the package regarding classo. We will briefly go over the main functions, basic operations and outputs. `cxreg` can be loaded using the `library` command:

```
library(cxreg)
```

We load a set of data created beforehand for illustration:

```
data(classo_example)
x <- classo_example$x
y <- classo_example$y
```

Note that "x" is already standardized, which makes the columns in X orthogonal. The `classo` provides the orthogonalization, and `standardization=TRUE` is the default. The necessity of the standardization is described in the original paper (2024).

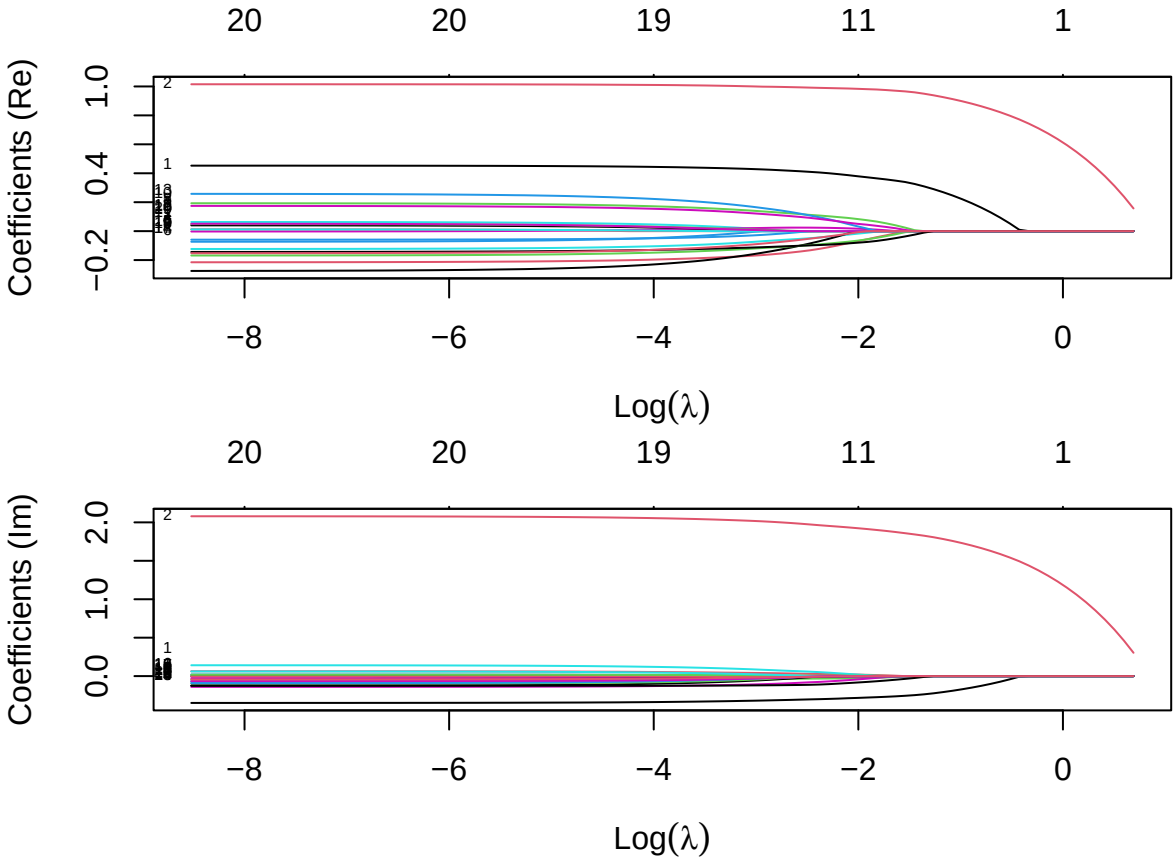
We fit the model using the most basic call to `classo`. In addition to standardization, the `intercept=FALSE` is used as a default, which means the complex-valued constant is not considered. This is another difference from the `glmnet` package.

```
fit <- classo(x,y)
```

`fit` is an object of class `classo` that contains all the relevant information of the fitted model for further use. Various methods are provided for the object such as `plot`, `coef`, and `predict`, just like `glmnet` package.

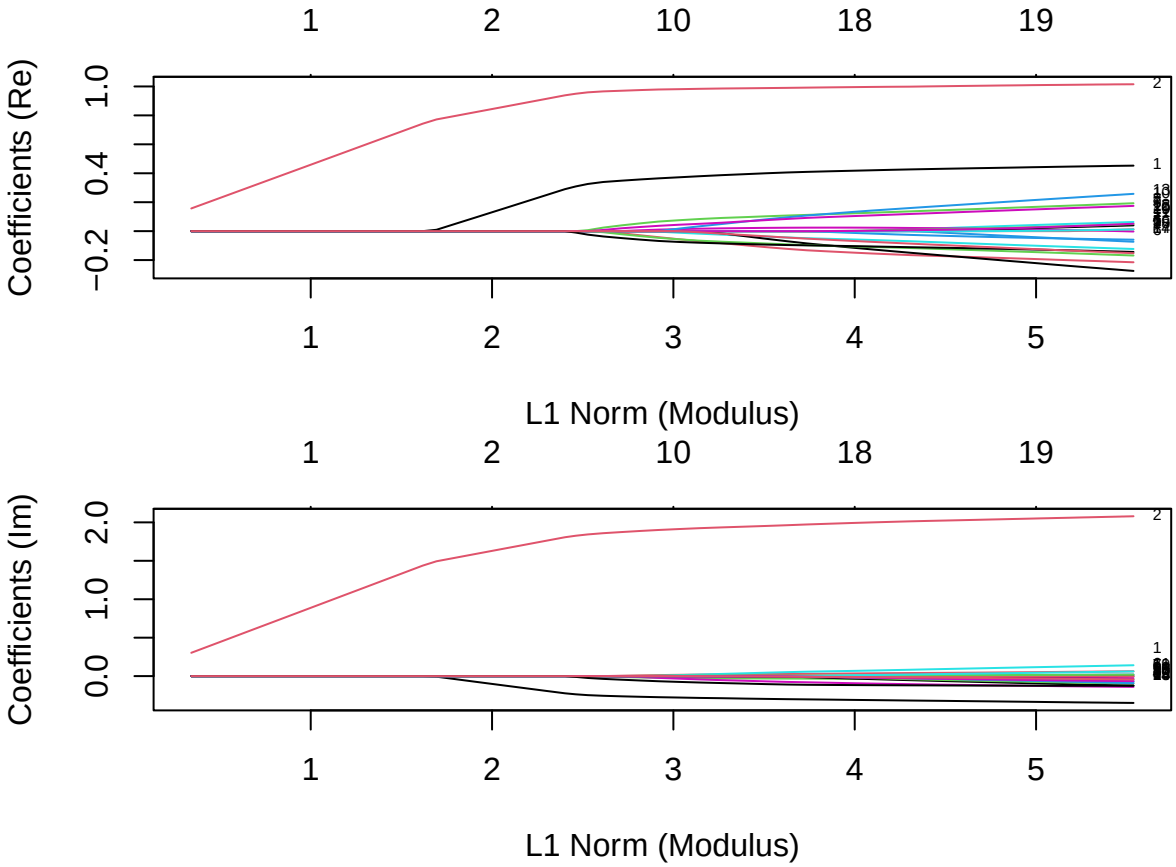
We can visualize the coefficients by executing the 'plot' method. First, we plot `fit` against the log λ values with labels:

```
plot(fit, xvar="lambda", label=TRUE)
```



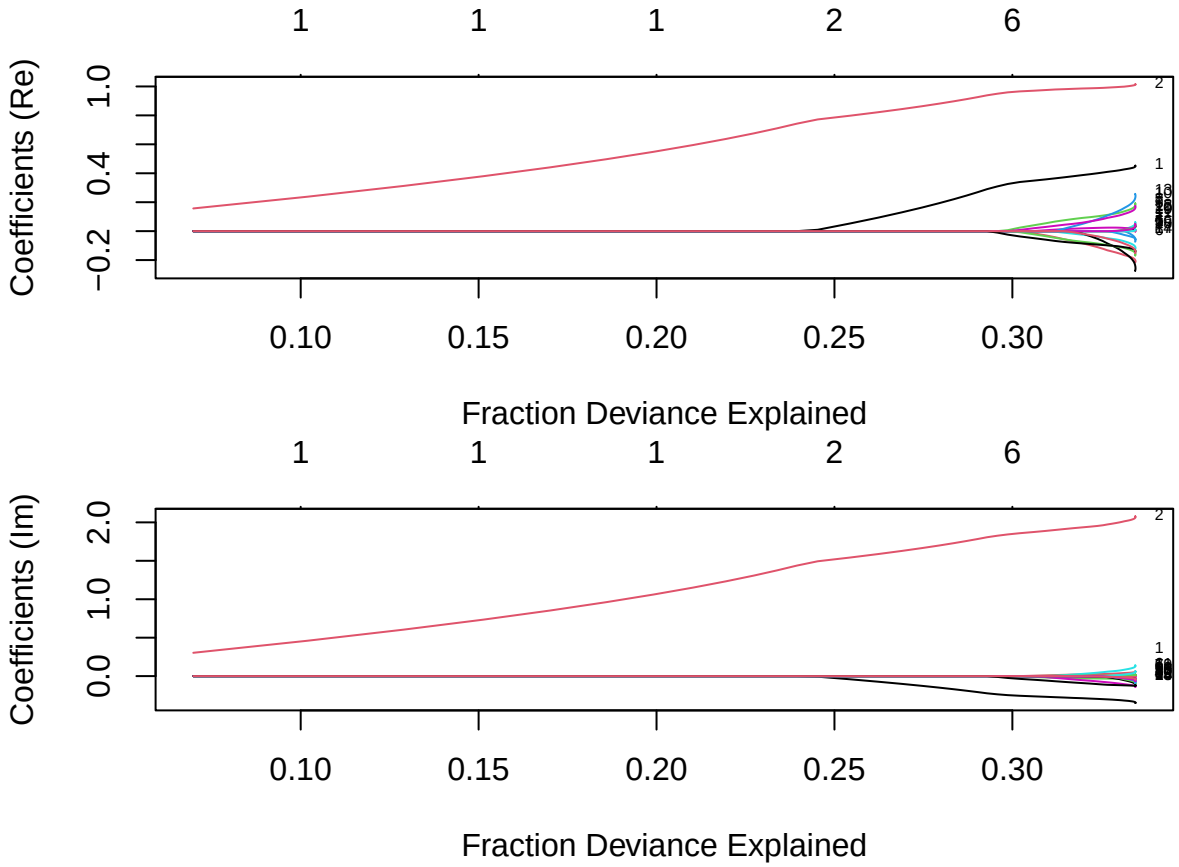
There are more types of x-axis, similar to those in `glmnet`. If "norm" is chosen, the values of the x-axis become ℓ_1 (modulus) of the coefficients:

```
plot(fit, xvar="norm", label=TRUE)
```



When "dev" is chosen, the percentage deviance explained by the model is used. Similar to those in `glmnet`, we can observe the blowing up phenomenon at the end of the curves:

```
plot(fit, xvar="dev", label=TRUE)
```



Unlike `glmnet`, the real (Re) and imaginary (Im) parts of the coefficients are displayed separately. It shows the path of its coefficients in the two parts separately against the ℓ_1-norm of the whole coefficient vector as λ varies. Users may also wish to annotate the curves: this can be done by setting "label = TRUE" in the plot command.

We can obtain the model coefficients at one or more λ's within the range of the sequence. Similar to `glmnet`, if s (lambda value) is not included in the sequence, only approximate values are provided:

```
any(fit$lambda == 0.1)
```

```
## [1] FALSE
```

```
coef(fit, s=0.1, exact=FALSE)
```

```
##                              s1
## V1   4.009883e-01-2.971462e-01i
## V2   9.890659e-01+1.954857e+00i
## V3   1.034953e-01-3.122989e-02i
## V4   0.000000e+00+0.000000e+00i
## V5   0.000000e+00+0.000000e+00i
## V6   2.280758e-02-6.980758e-02i
## V7  -5.060984e-05-7.355635e-04i
## V8  -1.128296e-01+2.628501e-02i
## V9   0.000000e+00+0.000000e+00i
## V10  8.409314e-02-2.585528e-04i
## V11 -3.891646e-02+4.569846e-02i
```

```
## V12  7.757692e-02-1.015113e-02i
## V13 -5.601959e-02-5.477926e-03i
## V14  0.000000e+00+0.000000e+00i
## V15 -8.651153e-02+6.299678e-03i
## V16  0.000000e+00+0.000000e+00i
## V17  0.000000e+00+0.000000e+00i
## V18 -1.724885e-05-4.515108e-04i
## V19 -9.358841e-02-1.049761e-01i
## V20 -3.741381e-02-5.181979e-03i
```

However, when "exact = TRUE", the data should be provided to create the original fit, as it merged to the original fit. In this case, both x and y need to be supplied as named arguments:

```
coef(fit, s=0.1, exact=TRUE, x=x, y=y)
```

```
##                          s1
## V1   0.40105458-0.29723427i
## V2   0.98901821+1.95463899i
## V3   0.10351452-0.03122183i
## V4   0.00000000+0.00000000i
## V5   0.00000000+0.00000000i
## V6   0.02282722-0.06991991i
## V7   0.00000000+0.00000000i
## V8  -0.11288850+0.02626716i
## V9   0.00000000+0.00000000i
## V10  0.08412895-0.00016548i
## V11 -0.03891355+0.04569551i
## V12  0.07753625-0.01010279i
## V13 -0.05599143-0.00552997i
## V14  0.00000000+0.00000000i
## V15 -0.08646014+0.00627438i
## V16  0.00000000+0.00000000i
## V17  0.00000000+0.00000000i
## V18  0.00000000+0.00000000i
## V19 -0.09367934-0.10513719i
## V20 -0.03736318-0.00517214i
```

Users can also make predictions at specific λ's with new input data: Note that the third line in the following chunk is about standardization, whose purpose is mentioned above. Here, 'response' means that predicted values of y.

```
set.seed(29)
nx <- array(rnorm(5*20), c(5,20)) + (1+1i) * array(rnorm(5*20), c(5,20))
for (j in 1:20) {
  nx[,j] <- nx[,j] / sqrt(mean(Mod(nx[,j])^2))
}
predict(fit, newx = nx, s = c(0.1, 0.05), type="response")
```

```
##                       s1                  s2
## [1,] -0.325392+3.646008i -0.396231+3.704370i
## [2,] -0.575134+1.796989i -0.588110+2.036627i
## [3,] -1.176727-1.152744i -1.279367-1.289659i
## [4,]  0.324128-1.768821i  0.433992-1.844658i
## [5,]  0.063003+2.285982i  0.049196+2.183301i
```

The other two output types are `coefficient` and `nonzero`, which are analogous to those in `glmnet`:

```
predict(fit, newx = nx, s = c(0.1, 0.05), type="coefficient")
```

```
##                              s1                         s2
## V1   4.009883e-01-2.971462e-01i  0.428201886-0.318938323i
## V2   9.890659e-01+1.954857e+00i  0.999698553+2.016771989i
## V3   1.034953e-01-3.122989e-02i  0.139397408-0.040215866i
## V4   0.000000e+00+0.000000e+00i -0.003497674+0.001613971i
## V5   0.000000e+00+0.000000e+00i  0.015768052-0.014274409i
## V6   2.280758e-02-6.980758e-02i  0.021275351-0.106945199i
## V7  -5.060984e-05-7.355635e-04i  0.006033702-0.056867445i
## V8  -1.128296e-01+2.628501e-02i -0.168106385+0.040922368i
## V9   0.000000e+00+0.000000e+00i  0.010268588-0.043910793i
## V10  8.409314e-02-2.585528e-04i  0.164894772-0.030224593i
## V11 -3.891646e-02+4.569846e-02i -0.072689205+0.085924440i
## V12  7.757692e-02-1.015113e-02i  0.121543797-0.015693518i
## V13 -5.601959e-02-5.477926e-03i -0.155912492-0.006879670i
## V14  0.000000e+00+0.000000e+00i  0.000000000+0.000000000i
## V15 -8.651153e-02+6.299678e-03i -0.119255771+0.011069806i
## V16  0.000000e+00+0.000000e+00i -0.028182433-0.027877388i
## V17  0.000000e+00+0.000000e+00i -0.001479872+0.023234451i
## V18 -1.724885e-05-4.515108e-04i  0.007283365-0.030805998i
## V19 -9.358841e-02-1.049761e-01i -0.112391306-0.120336184i
## V20 -3.741381e-02-5.181979e-03i -0.089882864-0.017884337i
```

```
predict(fit, newx = nx, s = c(0.1, 0.05), type="nonzero")
```

```
## $s1
##  [1]  1  2  5  6  7  9 10 11 12 14 17 18 19
##
## $s2
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 14 15 16 17 18 19
```

The function `classo` returns a sequence of models for the users to choose from. In many cases, users may prefer the software to select one of them. Cross-validation is perhaps the simplest and most widely used method for that task. `cv.classo` is the main function to do cross-validation here, along with various supporting methods such as plotting and prediction.

```
cvfit <- cv.classo(x,y,trace.it = 1)
```

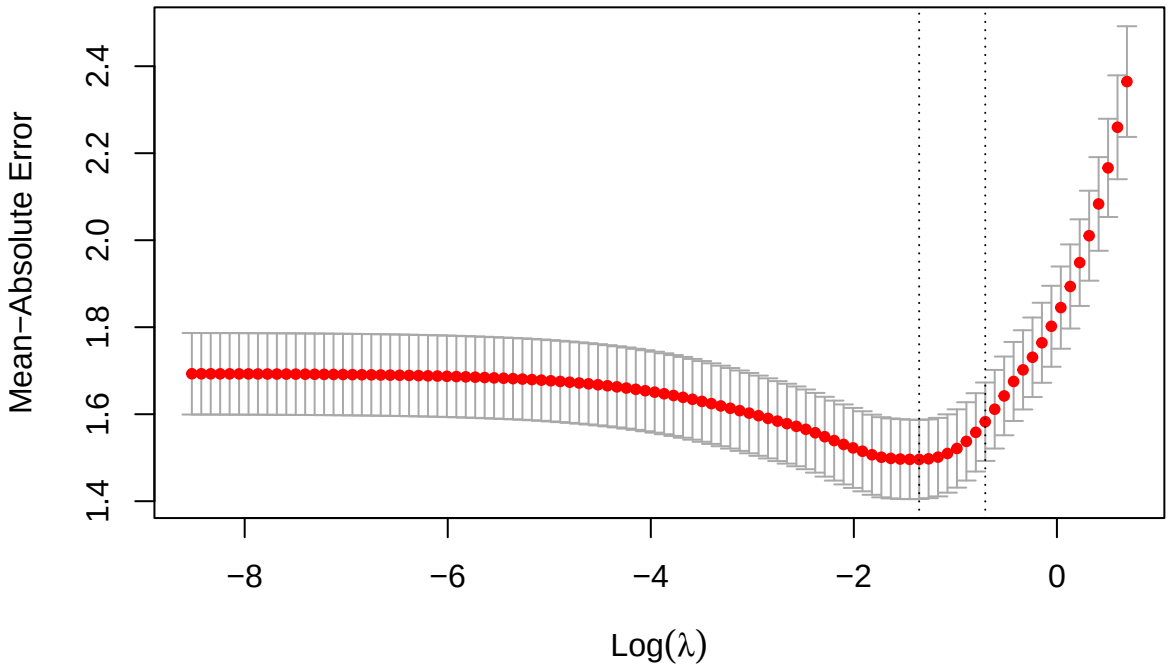
```
## Training
## Fold: 1/10
## Fold: 2/10
## Fold: 3/10
## Fold: 4/10
## Fold: 5/10
## Fold: 6/10
## Fold: 7/10
## Fold: 8/10
## Fold: 9/10
## Fold: 10/10
```

```
print(cvfit)
```

```
##
## Call:  cv.classo(x = x, y = y, trace.it = 1)
##
## Measure: Mean-Absolute Error
##
##     Lambda Index Measure      SE Nonzero
## min 0.2575    23   1.496 0.09089       2
## 1se 0.4938    16   1.583 0.09015       1
```

`cv.classo` returns a 'cv.classo' object, a list with all the ingredients of the cross-validated fit.

```
plot(cvfit)
```



This plots the cross-validation curve (red dotted line) along with upper and lower standard deviation curves along the λ sequence (error bars). Two special values along the λ sequence are indicated by the vertical dotted lines. "lambda.min" is the value of λ that gives minimum mean cross-validated error, while "lambda.lse" is the value of λ that gives the most regularized model such that the cross-validated error is within one standard error of the minimum.

We can use the following code to get the value of "lambda.min" and the model coefficients at that value of λ:

```
cvfit$lambda.min
```

```
## [1] 0.2574575
```

```
coef(cvfit, s = "lambda.min")
```

```
##                         s1
## V1   0.30900105-0.2366801i
## V2   0.94977208+1.8260644i
## V3   0.00000000+0.0000000i
## V4   0.00000000+0.0000000i
## V5   0.00000000+0.0000000i
## V6   0.00000000+0.0000000i
## V7   0.00000000+0.0000000i
## V8   0.00000000+0.0000000i
## V9   0.00000000+0.0000000i
## V10  0.00000000+0.0000000i
## V11  0.00000000+0.0000000i
## V12  0.00000000+0.0000000i
## V13  0.00000000+0.0000000i
## V14  0.00000000+0.0000000i
## V15  0.00000000+0.0000000i
## V16  0.00000000+0.0000000i
## V17  0.00000000+0.0000000i
## V18  0.00000000+0.0000000i
## V19 -0.01006015-0.0100882i
## V20  0.00000000+0.0000000i
```

To get the corresponding values at "lambda.1se", simply replace "lambda.min" with "lambda.1se" above, or omit the "s" argument, since "lambda.1se" is the default.

Note that unlike `glmnet` package, the coefficients are not represented in sparse matrix format, rather they are in the dense format. This is because of the traits of complex values in the function.

Predictions can be made based on the fitted `cv.glmnet` object as well. The code below gives predictions for the new input matrix "newx" at "lambda.min":

```
predict(cvfit, newx = x[1:5,], s = "lambda.min")
```

```
##                 lambda.min
## [1,]  0.2465621+0.0726532i
## [2,] -0.3463653-0.5161582i
## [3,]  0.5169684-1.4347468i
## [4,]  0.1364305+0.7100920i
## [5,] -0.4222057+3.4612903i
```

This concludes the basic usage of `classo`.

Example: cglasso

In this section, we illustrate our function `cglasso` for the two variants described in Section above. Again, we load a set of data created beforehand for illustration:

```
data(cglasso_example)
f_hat <- cglasso_example$f_hat
n <- cglasso_example$n
```

where the number of variables and sample size used in this example are $p = 30$ and $n = 500$, respectively, and the covariance matrix of the white noise process $\{X_t\}_{t=1}^n$ is

$$\Sigma = C^{-1} = (C_{k\ell})^{-1}, \quad C_{kk} = 0.7,\ C_{k,k-1} = C_{k-1,k} = 0.3.$$

Then the estimated spectral density $\hat{f}$ is obtained by DFT in (1).

First, consider CGLASSO-sc1: It is called by using "type="I" ".

```
fit_cglasso_I <- cglasso(S=f_hat,type="I",nobs=n)
```

```
## [1] "The algorithm was terminated at 24 th lambda"
```

Now, let's look at the example of CGLASSO-sc2:

```
fit_cglasso_II <- cglasso(S=f_hat,type="II",nobs=n, nlambda=30, stop_criterion = "AIC")
```

```
## [1] "The algorithm was terminated at 19 th lambda"
```

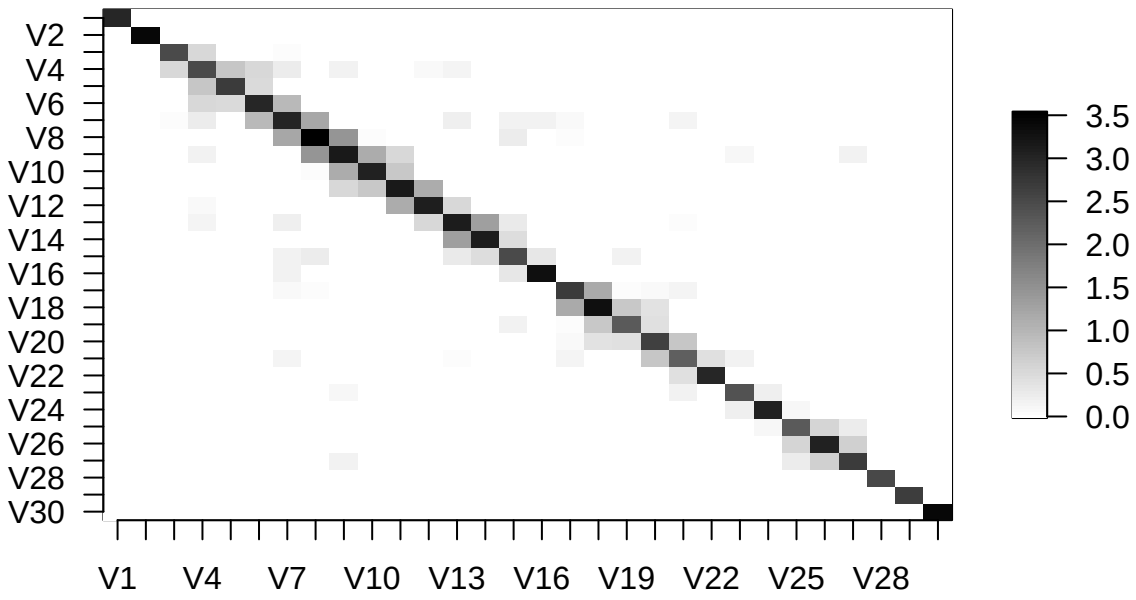
Here, we change the stopping criterion from "EBIC" to "AIC". Although the maximum number of paramters "nlambda" is set 30, due to the early termination, not all 30 lambdas created inside the function was not explored. However, if "stopping_rule=FALSE", cases of all 50 lambdas (by default) are explored:

```
fit_cglasso_another <- cglasso(S=f_hat,type="II",nobs=n, stopping_rule = FALSE)
fit_cglasso_another$lambda_grid
```

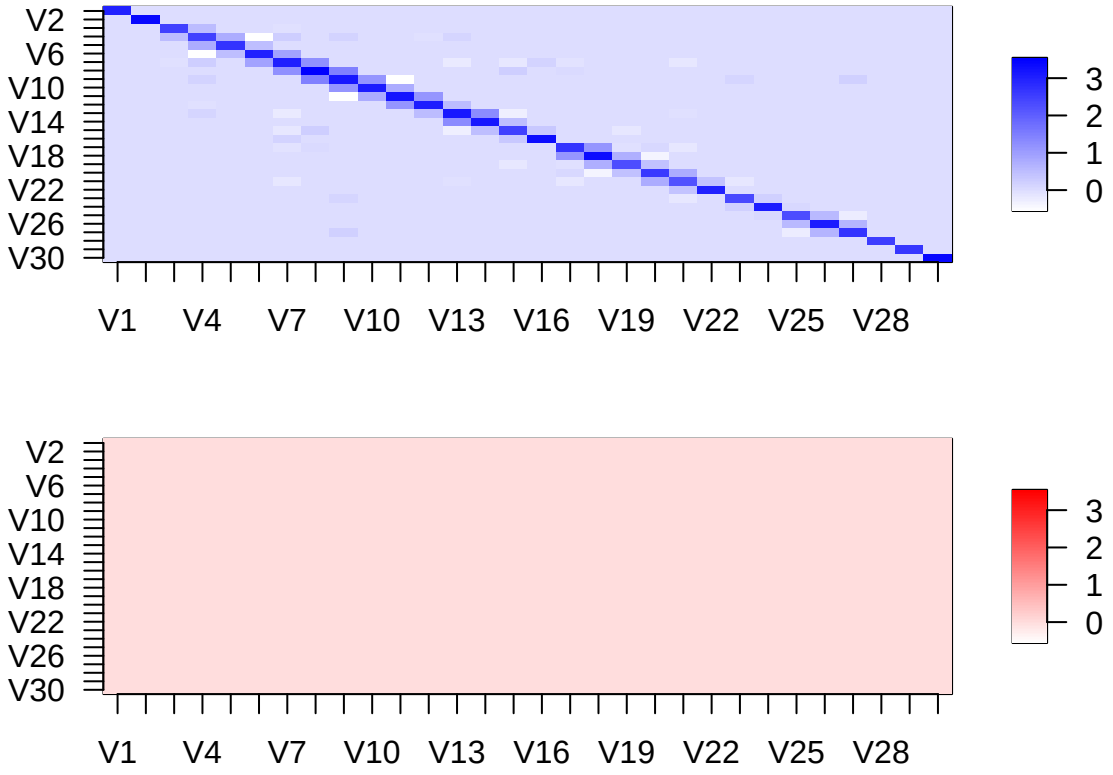
```
##  [1] 5.963815e-01 4.941872e-01 4.095047e-01 3.393331e-01 2.811859e-01
##  [6] 2.330027e-01 1.930760e-01 1.599910e-01 1.325754e-01 1.098576e-01
## [11] 9.103274e-02 7.543362e-02 6.250753e-02 5.179641e-02 4.292072e-02
## [16] 3.556595e-02 2.947146e-02 2.442132e-02 2.023655e-02 1.676887e-02
## [21] 1.389540e-02 1.151432e-02 9.541261e-03 7.906297e-03 6.551496e-03
## [26] 5.428850e-03 4.498577e-03 3.727713e-03 3.088943e-03 2.559630e-03
## [31] 2.121019e-03 1.757567e-03 1.456395e-03 1.206831e-03 1.000032e-03
## [36] 8.286694e-04 6.866709e-04 5.690049e-04 4.715018e-04 3.907065e-04
## [41] 3.237562e-04 2.682782e-04 2.223068e-04 1.842129e-04 1.526467e-04
## [46] 1.264896e-04 1.048147e-04 8.685393e-05 7.197088e-05 5.963815e-05
```

Finally, we can plot the heatmap using the list of estimated spectral precision matrices and specify the index to select a particular matrix from the list. The argument `type` determines whether the display shows the real part (`real`), imaginary part (`imaginary`), both (using `both`. in parallel), or the modulus (`mod`).

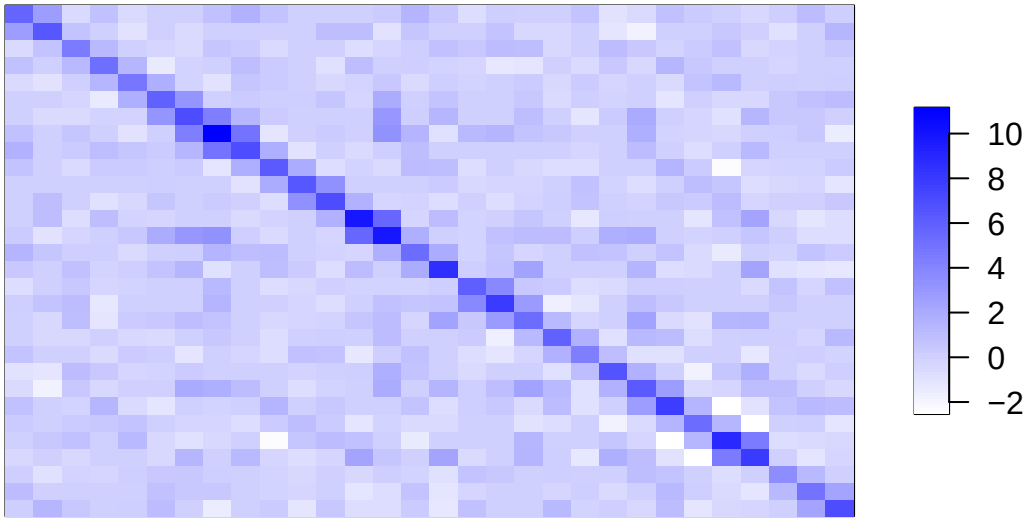
```
plot(fit_cglasso_I$Theta_list,index=fit_cglasso_I$min_index,type="mod",label=TRUE)
```



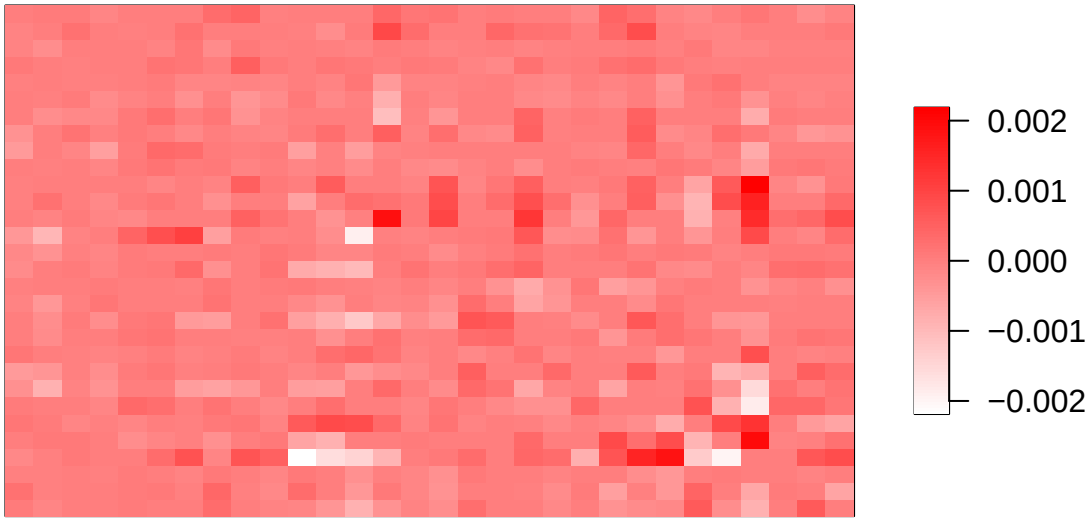

```
plot(fit_cglasso_I$Theta_list,index=fit_cglasso_I$min_index,type="both",label=TRUE)
```




```
plot(fit_cglasso_II$Theta_list,index=fit_cglasso_II$min_index,type="real",label=FALSE)
```




```
plot(fit_cglasso_II$Theta_list,index=fit_cglasso_II$min_index,type="imaginary",label=FALSE)
```



This concludes the basic usage of `cglasso`.

Deb, Navonil, Amy Kuceyeski, and Sumanta Basu. 2024. "Regularized Estimation of Sparse Spectral Precision Matrices." *arXiv Preprint arXiv:2401.11128*.

Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1): 1.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2008. "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics* 9 (3): 432–41.

Janková, Jana, and Sara van de Geer. 2018. "Inference in High-Dimensional Graphical Models." In *Handbook of Graphical Models*, 325–50. CRC Press.

Whittle, Peter. 1951. *Hypothesis Testing in Time Series Analysis*. Uppsala, Sweden: Almqvist & Wiksells.