# Estimation of covariance matrix via factor models

*Rui Zhou and Daniel P. Palomar*

*2018-08-18*

## Contents



---

This vignette illustrates the estimation of covariance matrix via factor models with the package `covFactorModel` and gives a description of the algorithms used.

## 1 Comparison with other packages

We compare the provided package `covFactorModel` with the existing package `FinCovRegularization` and function `stats::factanal()`. First, we compare the estimation results with package `FinCovRegularization`, which also estimates the covariance matrix of asset returns via three types of factor models as our package. We start by loading built-in data from package `FinCovRegularization`:

```
library(FinCovRegularization)
library(xts)

# load raw data
data(m.excess.c10sp9003)
assets <- m.excess.c10sp9003[, 1:10]
factor <- m.excess.c10sp9003[, 11]
T <- nrow(assets)
# convert data into xts object
assets_xts <- as.xts(assets, order.by = as.Date("1995-03-15")+1:T)
factor_xts <- as.xts(factor, order.by = as.Date("1995-03-15")+1:T)

# sector information for BARRA Industry factor model
# from help page of function FinCovRegularization::FundamentalFactor.Cov
beta <- matrix(0, 10, 3)
dimnames(beta) <- list(colnames(assets), c("Drug", "Auto", "Oil"))
beta[c("ABT", "LLY", "MRK", "PFE"), "Drug"] <- 1
beta[c("F", "GM"), "Auto"] <- 1
```

```
beta[c("BP", "CVX", "RD", "XOM"), "Oil"] <- 1
sector_info <- c(rep(1, 4),
                 rep(2, 2),
                 rep(3, 4))
```

Then, we use the two packages to compute the covariance matrix estimation (the comparison of execution time is ignored here because their implementation are almost same) via three factor models and compare the results:

```
library(covFactorModel)
# compare cov by macroeconomic factor model
my_cov_macro <- covFactorModel(assets_xts, type = "Macro", econ_fact = factor_xts)
cov_macro <- MacroFactor.Cov(assets, factor)
norm(cov_macro - my_cov_macro, "F")
#> [1] 8.396794e-18

# compare cov by BARRA Industry factor model
my_cov_BARRA <- covFactorModel(assets_xts, type = "Barra", stock_sector_info = sector_info)
cov_BARRA <- FundamentalFactor.Cov(assets, exposure = beta, method = "OLS")
norm(cov_BARRA - my_cov_BARRA, "F")
#> [1] 8.673617e-19

# compare cov by statistical factor model
my_cov_stat <- covFactorModel(assets_xts, type = "Stat-PCA", K = 3)
cov_stat <- StatFactor.Cov(assets, 3)
norm(cov_stat - my_cov_stat, "F")
#> [1] 4.970124e-17
```

It is clear that the covariance matrix estimation results from the packages `covFactorModel` and `FinCovRegularization` are exactly the same. Note that `covFactorModel` allows the user to choose different structures on residual covariance matrix (i.e., scaled identity, diagonal, block diagonal, and full), while `FinCovRegularization` assumes it to be diagonal only. (When use the BARRA Industry factor model, `covFactorModel` requires sector information in vector form or nothing if column names of data matrix is contained in the in-built database `data(stock_sector_database)`, while `FinCovRegularization` forces user to pass the sector information in matrix form.)

Next, we compare the performance of `covFactorModel()` and `stats::factanal()` in covariance matrix estimation. From the description of `factanal()` (use `?factanal` for details), it performs a maximum-likelihood factor analysis on a covariance matrix or data matrix and is in essence a model for the correlation matrix. We compare the correlation matrix estimation in terms of PRIAL (see next section for details) and running time. Since `covFactorModel()` returns the covariance matrix, we use `cov2cor()` to obtain the correlation matrix. As shown in Figure 1, `covFactorModel()` can achieve a similar estimation performance but is much faster compared with `factanal()`.

Summarizing, our package `covFactorModel` performs the same as the package `FinCovRegularization` in terms of covariance matrix estimation and computational speed for the three cases of Macroeconomic factor model, BARRA Industry factor model, and statistical PCA factor model (although `covFactorModel` allows for more types of structure in the residual covariance matrix). In addtition, our package `covFactorModel` can implement the statistical ML factor model with the same performance as the function `stats::factanal()` but with a much faster computational speed (one order of magnitude faster).

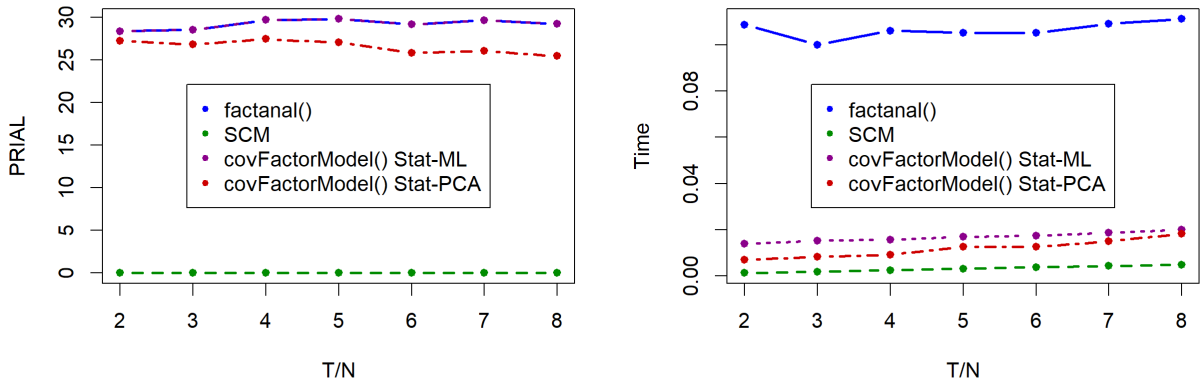



Figure 1: Average PRIAL and running time.

# 2 Usage of the package

## 2.1 Usage of `factorModel()`

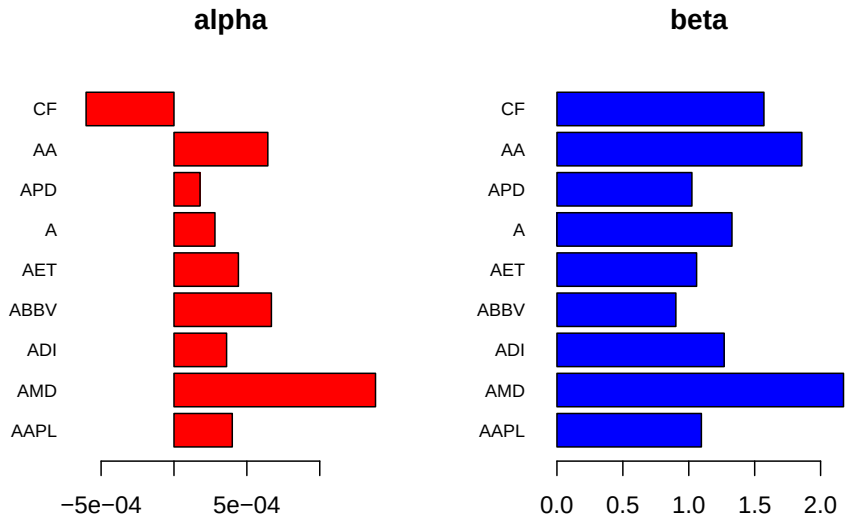
The function `factorModel()` builds a factor model for the data, i.e., it decomposes the asset returns into a factor component and a residual component. The user can choose different types of factor models, namely, macroeconomic, BARRA, or statistical. We start by loading some real market data using package `quantmod`:

```
library(xts)
library(quantmod)

# set begin-end date and stock namelist
begin_date <- "2016-01-01"
end_date <- "2017-12-31"
stock_namelist <- c("AAPL", "AMD", "ADI",  "ABBV", "AET", "A",  "APD", "AA","CF")

# download stock data from YahooFinance
data_set <- xts()
for (stock_index in 1:length(stock_namelist))
  data_set <- cbind(data_set, Ad(getSymbols(stock_namelist[stock_index],
                                            from = begin_date, to = end_date,
                                            auto.assign = FALSE)))
colnames(data_set) <- stock_namelist
indexClass(data_set) <- "Date"
# check stock data
head(data_set)
#>                AAPL  AMD      ADI     ABBV      AET        A      APD       AA       CF
#> 2016-01-04 100.62618 2.77 51.29987 52.05566 106.0680 39.70486 110.5926 23.00764 36.30703
#> 2016-01-05  98.10455 2.75 50.92294 51.83879 107.5435 39.56824 108.6151 21.96506 35.16852
#> 2016-01-06  96.18465 2.51 48.75560 51.84784 106.9999 39.74389 105.9699 20.40121 32.12948
#> 2016-01-07  92.12524 2.28 47.51174 51.69422 107.0484 38.05577 102.4173 19.59558 30.60548
#> 2016-01-08  92.61236 2.14 47.09712 50.28463 103.9419 37.65570 101.8866 19.12169 30.31861
#> 2016-01-11  94.11198 2.34 48.21848 48.68528 102.2431 37.02144 102.2889 18.95583 29.09045
tail(data_set)
#>                AAPL   AMD      ADI     ABBV      AET        A      APD    AA       CF
```

```
#> 2017-12-21 173.6298 10.89 87.77656 95.24335 179.2424 67.05882 159.9547 48.99 40.62705
#> 2017-12-22 173.6298 10.54 87.97459 95.53518 178.4787 66.88999 160.2095 49.99 41.08011
#> 2017-12-26 169.2248 10.46 87.75676 95.08771 178.9349 66.79068 159.7978 50.38 42.20289
#> 2017-12-27 169.2546 10.53 88.22212 95.41844 179.3614 66.84033 160.4547 51.84 42.41957
#> 2017-12-28 169.7308 10.55 88.49937 95.12663 179.7383 66.98931 161.3860 54.14 41.96652
#> 2017-12-29 167.8954 10.28 88.15281 94.07604 178.9052 66.65984 161.7903 53.87 41.89757

# download SP500 Index data from YahooFinance
SP500_index <- Ad(getSymbols("^GSPC", from = begin_date, to = end_date, auto.assign = FALSE))
colnames(SP500_index) <- "index"
# check SP500 index data
head(SP500_index)
#>              index
#> 2016-01-04 2012.66
#> 2016-01-05 2016.71
#> 2016-01-06 1990.26
#> 2016-01-07 1943.09
#> 2016-01-08 1922.03
#> 2016-01-11 1923.67
```

We first build a *macroeconomic factor model*, where `SP500_index` is used as one macroeconomic factor:

```
library(covFactorModel)
# compute log-returns
X <- diff(log(data_set), na.pad = FALSE)
f <- diff(log(SP500_index), na.pad = FALSE)
N <- ncol(X)  # number of stocks
T <- nrow(X)  # number of days

# use package to build macroeconomic factor model
macro_econ_model <- factorModel(X, type = "Macro", econ_fact = f)

# sanity check
X_ <- with(macro_econ_model,
           matrix(alpha, T, N, byrow = TRUE) + f %*% t(beta) + residual)
norm(X - X_, "F")
#> [1] 1.038217e-16

par(mfrow = c(1, 2))
barplot(macro_econ_model$alpha, horiz = TRUE,
        main = "alpha", col = "red", cex.names = 0.75, las = 1)
barplot(t(macro_econ_model$beta), horiz = TRUE,
        main = "beta", col = "blue", cex.names = 0.75, las = 1)
```




In finance, this is also known as capital asset pricing model (CAPM) assuming the risk free rate is zero. The term `alpha` is the stock's abnormal return and `beta` is the stock's responsiveness to the market return.

Next, we build a *BARRA industry factor model:*

```
barra_model <- factorModel(X, type = "Barra")
print(barra_model$beta)
#>      factor1 factor2 factor3
#> AAPL       1       0       0
#> AMD        1       0       0
#> ADI        1       0       0
#> ABBV       0       1       0
#> AET        0       1       0
#> A          0       1       0
#> APD        0       0       1
#> AA         0       0       1
#> CF         0       0       1

# sanity check
X_ <- with(barra_model,
           matrix(alpha, T, N, byrow = TRUE) + factors %*% t(beta) + residual)
norm(X - X_, "F")
#> [1] 2.050556e-17
```

Finally, we build a *statistical factor model*, which is based on principal component analysis (PCA):

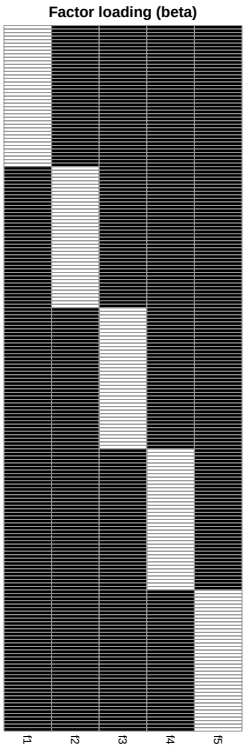
```
# set factor dimension as K=2
stat_model <- factorModel(X, K = 2)

# sanity check
X_ <- with(stat_model,
           matrix(alpha, T, N, byrow = TRUE) + factors %*% t(beta) + residual)
norm(X - X_, "F")
#> [1] 4.955954e-17
```

## 2.2 Usage of `covFactorModel()`

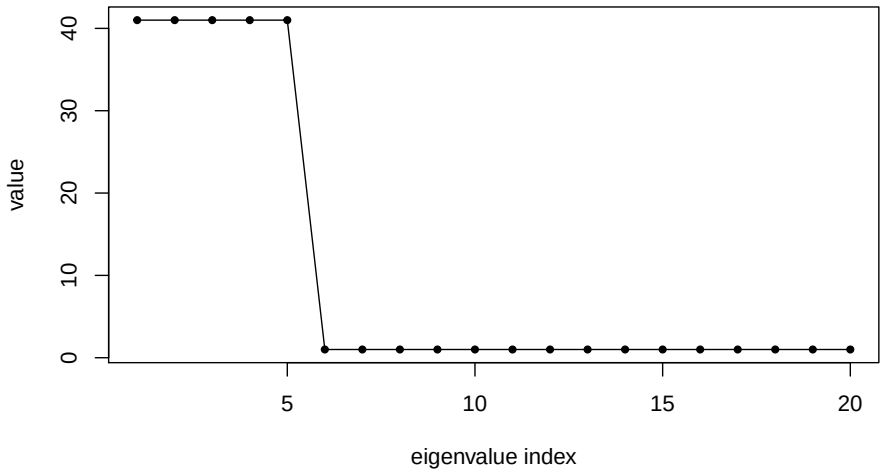
The function `covFactorModel()` estimates the covariance matrix of the data based on factor models. The user can choose not only the type of factor model (i.e., macroeconomic, BARRA, or statistical) but also the structure of the residual covariance matrix (i.e., scaled identity, diagonal, block diagonal, and full).

Firstly, we compare covariance matrix estimation based on different factor model decompositions. Let's start by preparing some parameters for the synthetic data generation:

```
library(covFactorModel)
library(xts)
library(MASS)
library(pheatmap)

# create parameters for generation of synthetic data
N <- 200  # number of stocks
mu <- rep(0, N)
num_sectors <- 5 # num of sectors
stock_sector_info <- rep(1:num_sectors, each = N/num_sectors)
# generate beta following BARRA model
beta <- matrix(0, N, num_sectors)
for (i in 1:num_sectors) {
  mask <- stock_sector_info == i
  beta[mask, i] <- 1
}
# show beta
colnames(beta) <- paste0("f", 1:num_sectors)
pheatmap(beta, cluster_rows = FALSE, cluster_cols = FALSE, color = c(1, 0), legend = FALSE,
         main = "Factor loading (beta)")
```



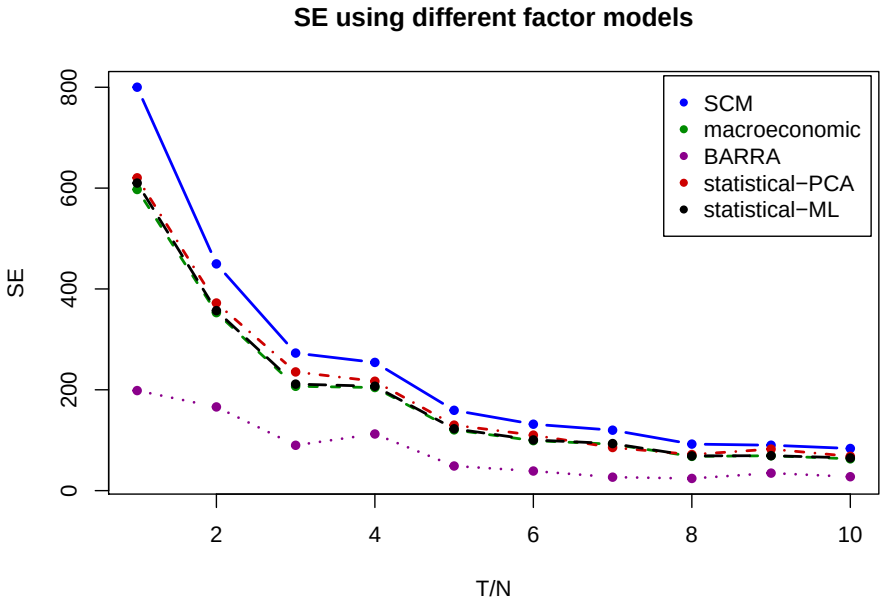
```r
Psi <- diag(N)
Sigma_f <- diag(num_sectors)
Sigma <- beta %*% Sigma_f %*% t(beta) + Psi

# plot first 20 eigenvalues of Sigma
plot(eigen(Sigma)$values[1:20], type = "o", pch = 20, xlab = "eigenvalue index", ylab = "value")
```




Then, we simply use function `covFactorModel()` (by default it uses a diagonal structure for the residual covariance matrix). We show the square error (SE) $\|\hat{\mathbf{\Sigma}} - \mathbf{\Sigma}_{\text{true}}\|_F^2$ w.r.t. number of observations:

```r
# generate synthetic data
set.seed(234)
err_scm <- err_macroecon <- err_barra <- err_statPCA <- err_statML <- c()
index_T <- N*seq(10)
for (T in index_T) {
  # generate factors and observed data matrix
  factors <- xts(mvrnorm(T, rep(0, num_sectors), Sigma_f),
                 order.by = as.Date('1995-03-15') + 1:T)
  X <- factors %*% t(beta) + xts(mvrnorm(T, mu, Psi), order.by = index(factors))

  # sample covariance matrix
  err_scm <- c(err_scm, norm(Sigma - cov(X), "F")^2)

  # macroeconomic factor model
  cov_macroecon <- covFactorModel(X, type = "Macro", econ_fact = factors)
  err_macroecon <- c(err_macroecon, norm(Sigma - cov_macroecon, "F")^2)

  # BARRA factor model
  cov_barra <- covFactorModel(X, type = "Barra", stock_sector_info = stock_sector_info)
  err_barra <- c(err_barra, norm(Sigma - cov_barra, "F")^2)

  # statistical factor model by PCA with diagonal Psi (default)
  cov_statPCA <- covFactorModel(X, type = "Stat-PCA", K = num_sectors)
  err_statPCA <- c(err_statPCA, norm(Sigma - cov_statPCA, "F")^2)
```

```r
  # statistical factor model by ML with diagonal Psi (default)
  cov_statML <- covFactorModel(X, type = "Stat-ML", K = num_sectors)
  err_statML <- c(err_statML, norm(Sigma - cov_statML, "F")^2)
}
res <- cbind("SCM"               = err_scm,
             "macroeconomic"     = err_macroecon,
             "BARRA"             = err_barra,
             "statistical-PCA"   = err_statPCA,
             "statistical-ML"    = err_statML)
colors <- c("blue", "green4", "darkmagenta", "red3", "gray0")
matplot(index_T/N, res,
        xlab = "T/N", ylab = "SE",
        main = "SE using different factor models",
        type = "b", pch = 20, lwd = 2, col = colors)
legend("topright", inset = 0.01, legend = colnames(res), pch = 20, col = colors)
```



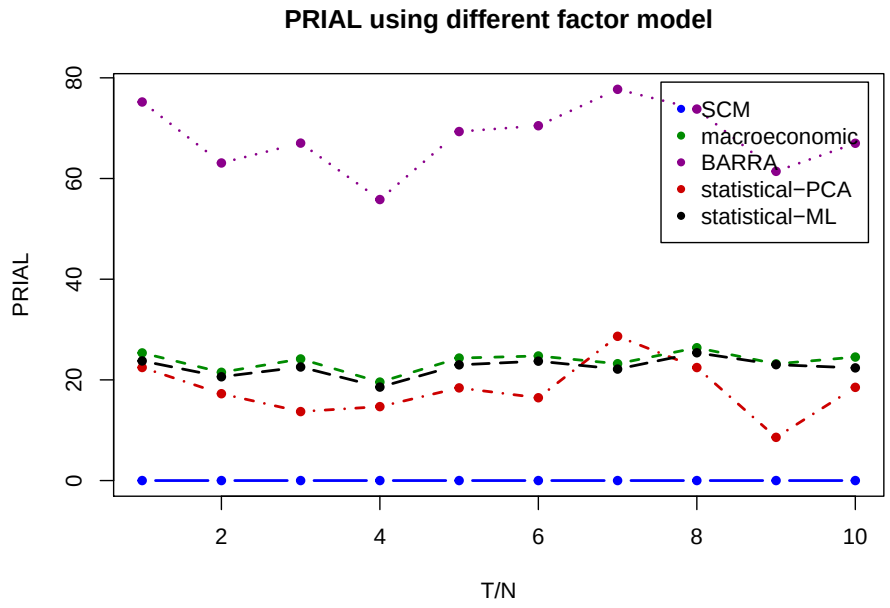

Obviously, using factor models for covariance matrix estimation definitely helps (note that BARRA is obviously the best simply because the synthetic data was generated according to the BARRA model). In order to show how well the estimated covariance matrices do compared to the sample covariance matrix (benchmark), the estimation error can also be evaluated in terms of PRIAL (PeRcentage Improvement in Average Loss):

$$\text{PRIAL} = 100 \times \frac{\|\hat{\mathbf{S}} - \boldsymbol{\Sigma}_{\text{true}}\|_F^2 - \|\hat{\boldsymbol{\Sigma}} - \boldsymbol{\Sigma}_{\text{true}}\|_F^2}{\|\hat{\mathbf{S}} - \boldsymbol{\Sigma}_{\text{true}}\|_F^2}$$

which goes to 0 when the estimation $\hat{\boldsymbol{\Sigma}}$ tends to the sample covariance matrix $\hat{\mathbf{S}}$ and goes to 100 when the estimation $\hat{\boldsymbol{\Sigma}}$ tends to the true covariance matrix $\boldsymbol{\Sigma}_{\text{true}}$.

```r
PRIAL <- 100*(1 - apply(res, 2, "/", res[, 1]))
matplot(index_T/N, PRIAL,
        xlab = "T/N", ylab = "PRIAL",
        main = "PRIAL using different factor model",
```

```
        type = "b", pch = 20, lwd = 2, col = colors)
legend("topright", inset=0.02, legend = colnames(res), pch = 20, col = colors)
```




The performance of BARRA Industry and macroeconomic factor models seems better than that of the statistical factor model, but this is just because the synthetic data has been generated according to the BARRA model and because the macroeconomic factor model has been fed with the exact factors. The reality of market data may be different with other results (e.g., the industry information might be missing or wrong because it changes over time, and so are the factors). The statistical factor model is always easier to implement and more robust to the aforementioned practical issues.

In Figure 2, we generate synthetic data using $\mathbf{\Psi}$ with different structures, namely, diagonal, block diagonal, scaled identity, and full. Then we estimate the covariance matrix using the statistical factor model (imposing different structures on $\mathbf{\Psi}$) and show the performance. The estimation based on the statistical factor model can beat the sample covariance matrix mostly except when $\mathbf{\Psi}$ has a full structure (i.e., no structure at all).

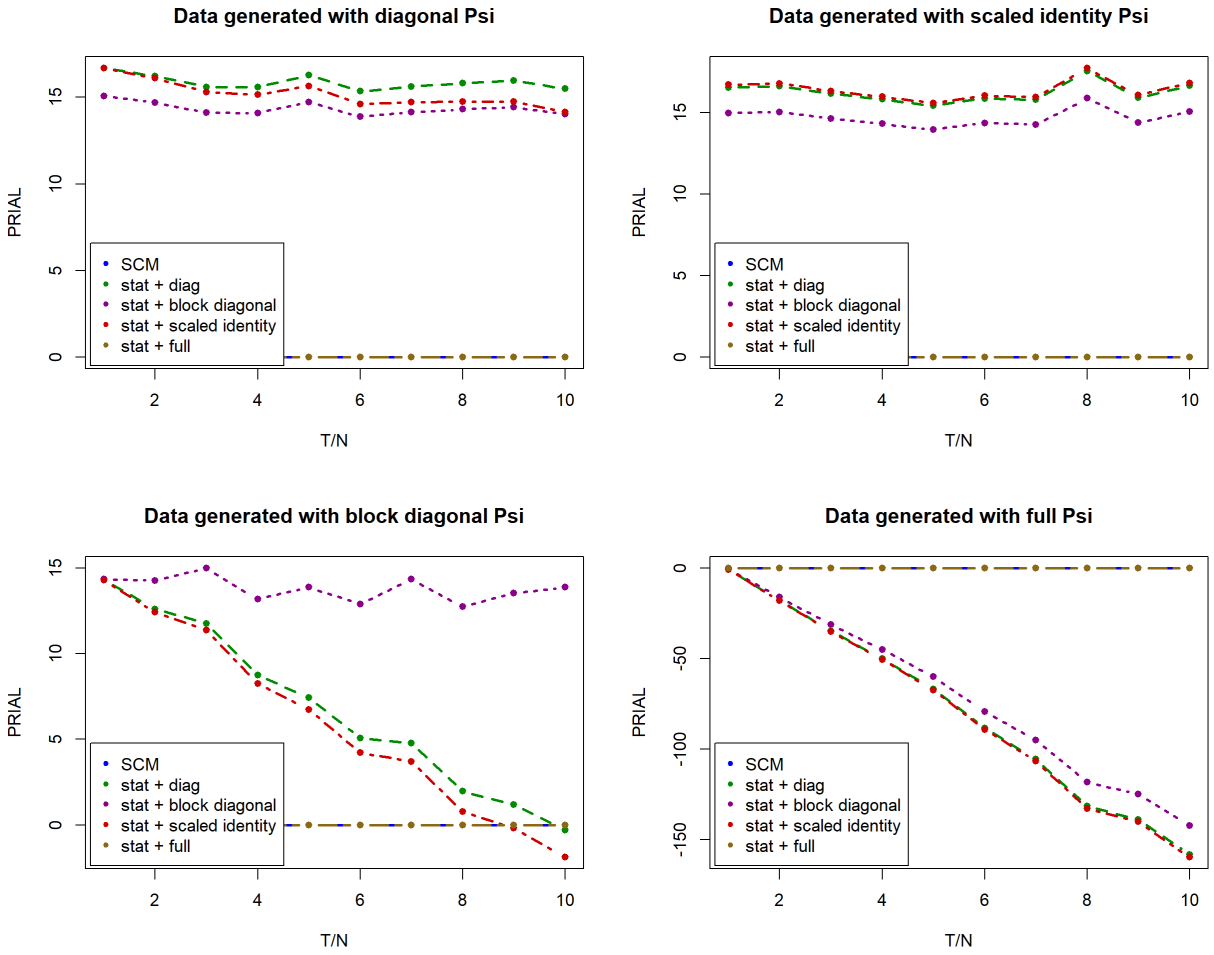



Figure 2: Performance of estimation under different Psi structures.

### 2.3 Usage of `getSectorInfo()`

The function `getSectorInfo()` provides sector information for a given set of stock symbols. The usage is very simple:

```
library(covFactorModel)

mystocks <- c("AAPL",  "ABBV", "AET", "AMD", "APD", "AA","CF", "A", "ADI", "IBM")
getSectorInfo(mystocks)
#> $stock_sector_info
#> AAPL ABBV  AET  AMD  APD   AA   CF    A  ADI  IBM
#>    1    2    2    1    3    3    3    2    1    1
#>
#> $sectors
#>                         1                         2                         3
#> "Information Technology"            "Health Care"               "Materials"
```

The built-in sector database can be overidden by providing a stock-sector pairing:

```
my_stock_sector_database <- cbind(mystocks, c(rep("sector1", 3),
                                              rep("sector2", 4),
                                              rep("sector3", 3)))
getSectorInfo(mystocks, my_stock_sector_database)
#> $stock_sector_info
#> AAPL ABBV  AET  AMD  APD   AA   CF    A  ADI  IBM
#>    1    1    1    2    2    2    2    3    3    3
#>
#> $sectors
#>         1         2         3
#> "sector1" "sector2" "sector3"
```

# 3 Explanation of the algorithms

The factor model decomposes the stock returns into two parts: low-dimensional factors and idiosyncratic residual noise. There are three basic types of factor models [1], namely, macroeconomic, fundamental, and statistical. Suppose there are $N$ stocks in market and we have $T$ observations, then factor models can be expressed in linear form:

$$x_{i,t} = \alpha_i + \beta_{1,i} f_{1,t} + \cdots + \beta_{K,i} f_{K,t} + \epsilon_{i,t},\ t = 1, \ldots, T$$

where $i$ is the stock index, $K$ is the number of factors, $\alpha_i$ is the intercept of the $i$-th stock, $\mathbf{f}_k = [f_{k,1}, \ldots, f_{k,T}]^T$ is the common $k$-th factor, $\boldsymbol{\beta}_i = [\beta_{1,i}, \ldots, \beta_{K,i}]^T$ is the factor loading of the $i$-th stock and $\epsilon_{i,t}$ is residual term for the $i$-th stock at time $t$. With the compact notation $\mathbf{F} = \begin{bmatrix} \mathbf{f}_1 & \cdots & \mathbf{f}_K \end{bmatrix}$, $\mathbf{x}_i = [x_{i,1}, \ldots, x_{i,T}]^T$, and $\boldsymbol{\epsilon}_i = [\epsilon_{i,1}, \ldots, \epsilon_{i,T}]^T$ it can also be written into vector form:

$$\mathbf{x}_i = \mathbf{1}\alpha_i + \mathbf{F}\boldsymbol{\beta}_i + \boldsymbol{\epsilon}_i,\ i = 1, \ldots, N$$

### 3.1 `factorModel()`: Build factor model for given data

The goal of `factorModel()` is the decomposition of a $T \times N$ data matrix $\mathbf{X}$ into factors and residual idiosyncratic component. User can choose different types of factor models, namely, macroeconomic, BARRA (a special case of fundamental factor model), or statistical.

### 3.1.1 Macroeconomic factor model (aka explicit factor model)

In this model, the factors are observed economic/financial time series. The macroeconomic factor model can be estimated through Least-Squares (LS) regression:

$$\underset{\boldsymbol{\gamma}_i}{\text{minimize}} \quad \|\mathbf{x}_i - \tilde{\mathbf{F}}\boldsymbol{\gamma}_i\|^2$$

where $\tilde{\mathbf{F}} = \begin{bmatrix} \mathbf{1}_T & \mathbf{F} \end{bmatrix}$ and $\boldsymbol{\gamma}_i = \begin{bmatrix} \alpha_i \\ \boldsymbol{\beta}_i \end{bmatrix}$. The closed-form solution is: $\hat{\boldsymbol{\gamma}}_i = \left(\tilde{\mathbf{F}}^T\tilde{\mathbf{F}}\right)^{-1}\tilde{\mathbf{F}}^T\mathbf{x}_i$. Then simply use the factor model decomposition to get the residual $\boldsymbol{\epsilon}_i = \mathbf{x}_i - \tilde{\mathbf{F}}\hat{\boldsymbol{\gamma}}_i$.

### 3.1.2 BARRA Industry factor model (specific case of fundamental factor models)

Normally, fundamental factor model use observable asset specific characteristics (fundamentals) like industry classification, market capitalization, style classification (value, growth), etc., to determine the common risk factors $\mathbf{F}$. In this function, we only consider one of the cases: BARRA Industry factor model, which assumes that there are $K$ factors corresponding to $K$ mutually exclusive industries (aka, sectors). Apart from that, the loadings $\beta_{i,k}$ are directly defined as

$$\beta_{i,k} = \begin{cases} 1 & \text{if stock } i \text{ is in industry } k \\ 0 & \text{otherwise.} \end{cases}$$

Using compact combination $\mathbf{B} = \begin{bmatrix} \boldsymbol{\beta}_1 & \cdots & \boldsymbol{\beta}_N \end{bmatrix}^T$, the industry factor model is (note that $\boldsymbol{\alpha} = \mathbf{0}$):

$$\mathbf{x}_t = \mathbf{B}\mathbf{f}_t + \boldsymbol{\epsilon}_t,\ t = 1, \ldots, T$$

where $\mathbf{x}_t = [x_{1,t}, \ldots, x_{N,t}]^T$ and $\mathbf{f}_t = [f_{1,t}, \ldots, f_{K,t}]^T$. Here the LS regression can also be applied to recover the factors (instead of the loadings as before) as

$$\underset{\mathbf{f}_t}{\text{minimize}} \quad \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{x}_t - \mathbf{B}\mathbf{f}_t\|_2^2$$

The solution is $\hat{\mathbf{f}}_t = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{x}_t,\ t = 1, \ldots, T$ and the residual can be simply calculated as $[\hat{\epsilon}_{1,t}, \ldots, \hat{\epsilon}_{N,t}]^T = \mathbf{x}_t - \mathbf{B}\hat{\mathbf{f}}_t$.

### 3.1.3 Statistical factor model via PCA (aka implicit factor model)

The statistical factor model via Principal Component Analysis (PCA) assumes that $\mathbf{f}_t$ is an affine transformation of $\mathbf{x}_t$, i.e., $\mathbf{f}_t = \mathbf{d} + \mathbf{C}^T\mathbf{x}_t$, where $\mathbf{d} \in \mathbb{R}^K$ and $\mathbf{C} \in \mathbb{R}^{N\times K}$ are parameters to be estimated. Therefore, we can formulate the problem as

$$\underset{\boldsymbol{\alpha},\mathbf{B},\mathbf{C},\mathbf{d}}{\text{minimize}} \quad \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{x}_t - \boldsymbol{\alpha} - \mathbf{B}\left(\mathbf{C}^T\mathbf{x}_t + \mathbf{d}\right)\|_2^2$$

which is proven equivalent to PCA and thus solved by

$$\hat{\boldsymbol{\alpha}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{x}_t, \quad \hat{\mathbf{B}} = \hat{\mathbf{C}} = \hat{\boldsymbol{\Gamma}}^{(K)}, \quad \hat{\mathbf{d}} = -\hat{\mathbf{C}}^T\hat{\boldsymbol{\alpha}}$$

where $\hat{\boldsymbol{\Gamma}}^{(K)} \in \mathbb{R}^{N\times K}$ with $k$-th column being the $k$-th largest eigenvector of sample covariance matrix $\hat{\mathbf{S}}$. The desired structure on the residual covariance matrix is then enforced.

A further refinement of the above PCA method is based on an iterative approach where each iteration performs an improved PCA (the method above corresponds to just one iteration) [see [2] for details]:

**Algorithm 1**

1. Calculate sample covariance matrix $\hat{\mathbf{S}}$ and eigen-decomposition (EVD): $\hat{\mathbf{\Gamma}}_1\hat{\mathbf{\Lambda}}_1\hat{\mathbf{\Gamma}}_1^T$
2. Set index $s = 1$
3. Compute $\hat{\mathbf{B}}_{(s)} = \hat{\mathbf{\Gamma}}_{(s)}^{(K)}\hat{\mathbf{\Lambda}}_{(s)}^{(K)\frac{1}{2}}$, $\hat{\mathbf{\Psi}}_{(s)} = \text{struct}(\hat{\mathbf{\Sigma}} - \hat{\mathbf{B}}_{(s)}\hat{\mathbf{B}}_{(s)}^T)$, $\hat{\mathbf{\Sigma}}_{(s)} = \hat{\mathbf{B}}_{(s)}\hat{\mathbf{B}}_{(s)}^T + \hat{\mathbf{\Psi}}_{(s)}$
4. Update EVD: $\hat{\mathbf{\Sigma}} - \hat{\mathbf{\Psi}}_{(s)} = \hat{\mathbf{\Gamma}}_{(s+1)}\hat{\mathbf{\Lambda}}_{(s+1)}\hat{\mathbf{\Gamma}}_{(s+1)}^T$ and $s \leftarrow s + 1$
5. Repeat Steps 3-4 until convergence.
6. Return $(\hat{\mathbf{B}}_{(s)}, \hat{\mathbf{\Psi}}_{(s)}, \hat{\mathbf{\Sigma}}_{(s)})$

where struct() is to impose certain structure on $\hat{\mathbf{\Psi}}_{(s)}$, one typical option is diagonal. After the algorithm is done, we can calculate $\hat{\boldsymbol{\alpha}} = \frac{1}{T}\sum_{t=1}^T \mathbf{x}_t$ and build the statistical factor model using the algorithm output:

$$\hat{\mathbf{B}} = \hat{\mathbf{\Gamma}}^{(K)}\hat{\mathbf{\Lambda}}^{(K)^{\frac{1}{2}}}, \quad \hat{\mathbf{f}}_t = \hat{\mathbf{\Lambda}}^{(K)^{-\frac{1}{2}}}\hat{\mathbf{\Gamma}}^{(K)^T}(\mathbf{x}_t - \hat{\boldsymbol{\alpha}}), \quad \hat{\boldsymbol{\epsilon}}_t = \mathbf{x}_t - \hat{\boldsymbol{\alpha}} - \hat{\mathbf{B}}\hat{\mathbf{f}}_t$$

(Note that the Algorithm 1 is equivalent to previous method when only one iteration is performed.)

## 3.2 `covFactorModel()`: Covariance matrix estimation via factor models

### 3.2.1 Through the factor decomposition obtained by the function `factorModel()`

The first approach is based on the factor model decomposition. As mentioned above, the factor model can be expressed as:

$$\mathbf{x}_t = \boldsymbol{\alpha} + \mathbf{B}\mathbf{f}_t + \boldsymbol{\epsilon}_t, \ t = 1, \ldots, T$$

Assuming $\{\mathbf{f}_t\}$ and $\{\boldsymbol{\epsilon}_t\}$ are uncorrelated, the covariance matrix $\mathbf{\Sigma}$ can be written as

$$\mathbf{\Sigma} = \mathbf{B}\mathbf{\Sigma}_\mathbf{f}\mathbf{B}^T + \mathbf{\Psi}$$

where $\mathbf{\Sigma}_\mathbf{f} = \mathsf{Cov}[\mathbf{x}_t]$ and $\mathbf{\Psi} = \mathsf{Cov}[\boldsymbol{\epsilon}_t]$. Therefore, we can simply use result from function `factorModel()` to estimate covariance matrix $\mathbf{\Sigma}$ as:

$$\hat{\mathbf{\Sigma}} = \hat{\mathbf{B}}\hat{\mathbf{\Sigma}}_\mathbf{f}\hat{\mathbf{B}}^T + \hat{\mathbf{\Psi}}$$

where $\hat{\mathbf{\Sigma}}_\mathbf{f}$ and $\hat{\mathbf{\Psi}}$ are the sample covariance matrix of $\{\mathbf{f}_t\}$ and $\{\boldsymbol{\epsilon}_t\}$. Besides, the $\mathbf{\Psi}$ is expected to follow a special structure, i.e.,

$$\hat{\mathbf{\Sigma}} = \hat{\mathbf{B}}\hat{\mathbf{\Sigma}}_\mathbf{f}\hat{\mathbf{B}}^T + \text{struct}\{\hat{\mathbf{\Psi}}\}.$$

In the statistical factor model by PCA of function `factorModel()`, the estimate $\hat{\mathbf{\Sigma}}$ is actually available when building the model. Therefore the algorithm output $\hat{\mathbf{\Sigma}}_{(s)}$ is directly extracted as the covariance matrix estimation.

### 3.2.2 Through maximum likelihood (ML) estimation under the Gaussian assumption

Another popular statistical factor model covariance matrix estimation is based on maximum likelihood (ML) estimation. Under the Gaussian assumption on the returns, it can be formulated as

$$\begin{aligned}
\underset{\mathbf{B},\mathbf{\Psi}}{\mathsf{minimize}} \quad & -\log|\mathbf{\Sigma}^{-1}| + \text{Tr}\left(\mathbf{\Sigma}^{-1}\hat{\mathbf{S}}\right) \\
\mathsf{subject\ to} \quad & \mathbf{\Sigma} = \mathbf{B}\mathbf{B}^T + \mathbf{\Psi} \\
& \mathbf{B} \in \mathbb{R}^{N\times K} \\
& \mathbf{\Psi} = \text{diag}\,(\psi_1, \ldots, \psi_p) \geq \epsilon\mathbf{I}
\end{aligned}$$

To solve this problem, we implement a very efficient algorithm from [3], which is much faster than the one used in `stats::factanal()`.

# 4 Macroeconomic factor model with sparse loading matrix

We impose no structure constraint on loading matrix $\mathbf{B}$ till now, which would weaken its interpretability or incur the over-fitting problem. One convenient approach is to assume $\mathbf{B}$ is sparse, i.e., add regularization term to loss function. Recall the macroeconomic factor model estimation problem in compact form is

$$\underset{\boldsymbol{\alpha},\mathbf{B}}{\text{minimize}} \quad \frac{1}{2T}\|\mathbf{X}-\mathbf{1}\boldsymbol{\alpha}^T-\mathbf{F}\mathbf{B}^T\|_F^2$$

In this Section, we show how to estimate loading matrix $\mathbf{B}$ of macroeconomic factor model with sparsity assumption.

## 4.1 Element-wise sparse

One of the most popular and convenient method is to add penalty element-wisely to $\mathbf{B}$, i.e.,

$$\underset{\boldsymbol{\alpha},\mathbf{B}}{\text{minimize}} \quad \frac{1}{2T}\|\mathbf{X}-\mathbf{1}\boldsymbol{\alpha}^T-\mathbf{F}\mathbf{B}^T\|_F^2 \quad +\sum_{(i,j)} p\left(B_{i,j}\right)$$

where $p(\theta)$ is a penalty function with some typical choices, e.g., minimax concave penalty (MCP), smoothly clipped absolute deviation (SCAD) and lasso. The three penalty functions are defined by:

$$p_{MCP}(\theta)=\begin{cases}\lambda|\theta|-\frac{\theta^2}{2\gamma} & |\theta|\le\gamma\lambda\\ \frac{1}{2}\gamma\lambda^2 & |\theta|\le\gamma\lambda\end{cases} \quad P_{SCAD}(\theta)=\begin{cases}\lambda|\theta| & |\theta|\le\lambda\\ \frac{\gamma\lambda|\theta|-0.5\left(\theta^2+\lambda^2\right)}{\gamma-1} & \lambda<|\theta|\le\gamma\lambda\\ \frac{\lambda^2\left(\gamma^2-1\right)}{2(\gamma-1)} & |\theta|>\gamma\lambda\end{cases} \quad p_{lasso}(\theta)=\lambda|\theta|$$

As each row of $\mathbf{B}$ is decoupled, the above problem can be decomposed into $N$ sub-problems with each be follows

$$\underset{\alpha_i,\boldsymbol{\beta}_i}{\text{minimize}} \quad \frac{1}{2T}\|\mathbf{x}_i-\mathbf{1}\alpha_i-\mathbf{F}\boldsymbol{\beta}_i\|_2^2 \quad +\sum_{j} p\left(B_{i,j}\right)$$

We found a package `ncvreg` which is very powerful to solve above problem with three mentioned penalty function. For matrix case, we can simply call it $N$ times and combine the final results. The example codes are given below as

```
library(ncvreg)
library(MASS)
set.seed(123)

# implement the function using package ncvreg
linreg_ele_sparse <- function(Factor, X, penalty = "lasso", lambda = 1, gamma = 4) {
  N <- ncol(X)
  K <- ncol(Factor)
  B <- matrix(0, N, K)
  alpha <- matrix(0, N, 1)
  for (i in 1:N) {
    tmp <- ncvreg(Factor, X[, i], family = "gaussian",
                  penalty = penalty, lambda = lambda, gamma = gamma)$beta
    B[i, ] <- as.vector(tmp[-1])
    alpha[i] <- tmp[1]
  }
  return(list(
    "B" = B,
    "alpha" = alpha
  ))
```

```
}

# compare the real B with estimated version by three type of penalty function
# generate factor structure data with element-sparse B
T <- 50
N <- 10
K <- 3
lambda <- 0.01
Factor <- matrix(rnorm(T*K), T, K)
Factor <- mvrnorm(n = T, mu = rep(0, K), Sigma = diag(K))
B_real <- matrix(rnorm(K*N), N, K)
B_real[abs(B_real) < 0.5] <- 0
X <- Factor %*% t(B_real) + 0.01 * matrix(rnorm(T*N), T, N)
```

Let's do the least-square estimation without penalty term and see the results

```
fit_nopenalty <- linreg_ele_sparse(Factor, X, penalty = "lasso", lambda = 0)

# show real B and its non-penalty estimation
print(B_real)
#>              [,1]       [,2]       [,3]
#>  [1,] -0.7152422  1.3011760  0.0000000
#>  [2,] -0.7526890  0.7567748  1.2181086
#>  [3,] -0.9385387 -1.7267304 -1.3387743
#>  [4,] -1.0525133 -0.6015067  0.6608203
#>  [5,]  0.0000000  0.0000000 -0.5229124
#>  [6,]  0.0000000  0.7035239  0.6837455
#>  [7,] -2.0142105  0.0000000  0.0000000
#>  [8,]  0.0000000 -1.2586486  0.6329607
#>  [9,]  1.2366750  1.6844357  1.3355176
#> [10,]  2.0375740  0.9113913  0.0000000
print(fit_nopenalty$B)
#>               [,1]          [,2]          [,3]
#>  [1,] -0.717357402  1.2997206989 -0.0007982666
#>  [2,] -0.752010522  0.7571250192  1.2182790308
#>  [3,] -0.937257140 -1.7268490154 -1.3389380506
#>  [4,] -1.053288562 -0.6002033616  0.6603603979
#>  [5,]  0.001789329 -0.0004867880 -0.5226743705
#>  [6,]  0.002997378  0.7054429240  0.6812290779
#>  [7,] -2.013026521  0.0003653609 -0.0004651422
#>  [8,] -0.000805525 -1.2549932732  0.6320091507
#>  [9,]  1.234909284  1.6851224016  1.3370964453
#> [10,]  2.038388189  0.9104624027  0.0026302068
```

It is significant that all the elements in estimated $\mathbf{B}$ are nonzero. Now let us estimate it again with penalty

```
fit_lasso <- linreg_ele_sparse(Factor, X, penalty = "lasso", lambda = 0.01)
fit_MCP <-   linreg_ele_sparse(Factor, X, penalty = "MCP", lambda = 0.01, gamma = 3)
fit_SCAD <-  linreg_ele_sparse(Factor, X, penalty = "SCAD", lambda = 0.01, gamma = 3.7)

# show result
print(fit_lasso$B)
#>              [,1]       [,2]       [,3]
#>  [1,] -0.7072337  1.2894224  0.0000000
```

```
#>  [2,] -0.7411290  0.7489424  1.2083948
#>  [3,] -0.9265676 -1.7172452 -1.3306948
#>  [4,] -1.0409810 -0.5860961  0.6458932
#>  [5,]  0.0000000  0.0000000 -0.5118097
#>  [6,]  0.0000000  0.6963452  0.6724008
#>  [7,] -2.0023690  0.0000000  0.0000000
#>  [8,]  0.0000000 -1.2416446  0.6184172
#>  [9,]  1.2242198  1.6755186  1.3288532
#> [10,]  2.0273006  0.8997473  0.0000000
print(fit_MCP$B)
#>             [,1]       [,2]       [,3]
#>  [1,] -0.7174110  1.2995651  0.0000000
#>  [2,] -0.7520105  0.7571250  1.2182790
#>  [3,] -0.9372571 -1.7268490 -1.3389381
#>  [4,] -1.0532886 -0.6002034  0.6603604
#>  [5,]  0.0000000  0.0000000 -0.5226601
#>  [6,]  0.0000000  0.7052447  0.6814573
#>  [7,] -2.0130679  0.0000000  0.0000000
#>  [8,]  0.0000000 -1.2549410  0.6319480
#>  [9,]  1.2349093  1.6851224  1.3370964
#> [10,]  2.0385774  0.9109857  0.0000000
print(fit_SCAD$B)
#>             [,1]       [,2]       [,3]
#>  [1,] -0.7174110  1.2995651  0.0000000
#>  [2,] -0.7520105  0.7571250  1.2182790
#>  [3,] -0.9372571 -1.7268490 -1.3389381
#>  [4,] -1.0532886 -0.6002034  0.6603604
#>  [5,]  0.0000000  0.0000000 -0.5226601
#>  [6,]  0.0000000  0.7052447  0.6814573
#>  [7,] -2.0130679  0.0000000  0.0000000
#>  [8,]  0.0000000 -1.2549410  0.6319480
#>  [9,]  1.2349093  1.6851224  1.3370964
#> [10,]  2.0385774  0.9109857  0.0000000
```

Obviously, the three estimations all give us a element-wise sparse estimation to $\mathbf{B}$.

## 4.2 Column-wise sparsity

Sometimes, we may want the $\mathbf{B}$ be not only element-wise sparse, but also column-wise sparse. As the macroeconomic factors can be collected from many different sources like some public national statistical data or bought from the agencies, we hope some useless or low-impact factors could be identified via the building of factor model. If a column of estimated $\mathbf{B}$ is all zero, then the corresponding factor can be intuitively seen as useless or low-impact. We introduce here two methods to help get a column-wise sparse estimation for $\mathbf{B}$.

### 4.2.1 Group Lasso

Basically, the group lasso problem is

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \frac{1}{2}\left\|\mathbf{x}-\sum_{l}^{m}\mathbf{F}^{(l)}\boldsymbol{\beta}^{(l)}\right\| + \lambda\sum_{l}^{m}\sqrt{p_l}\left\|\boldsymbol{\beta}^{(l)}\right\|_2$$

where $\mathbf{F}^{(l)}$ is the submatrix of $\mathbf{F}$ with columns corresponding to the factors in group $l$, $\boldsymbol{\beta}^{(l)}$ the coefficient vector of that group and $p_l$ is the length of $\boldsymbol{\beta}^{(l)}$. We can reformulate our macroeconomic factor model

estimation problem by using group lasso

$$\underset{\text{vec}(\mathbf{B}^T)}{\text{minimize}} \quad \frac{1}{2}\left\|\text{vec}\left(\mathbf{X}\right) - \left(\mathbf{I}\otimes\mathbf{F}\right)\text{vec}\left(\mathbf{B}^T\right)\right\| + \lambda\sum_{l}^{m}\sqrt{p_l}\left\|\text{vec}\left(\mathbf{B}^T\right)^{(l)}\right\|_2$$

As our expectation, that $\mathbf{B}$ should be column-wise sparse, we cam simply pass the information that $i$-th and $(i+K)$-th factors are of same group. We found a package `SGL` which fits a linear regression model of lasso and group lasso regression, i.e.,

$$\underset{\text{vec}(\mathbf{B}^T)}{\text{minimize}} \quad \frac{1}{2}\left\|\text{vec}\left(\mathbf{X}\right) - \left(\mathbf{I}\otimes\mathbf{F}\right)\text{vec}\left(\mathbf{B}^T\right)\right\| + (1-\alpha)\,\lambda\sum_{l}^{m}\sqrt{p_l}\left\|\text{vec}\left(\mathbf{B}^T\right)^{(l)}\right\|_2 + \alpha\lambda\left\|\text{vec}\left(\mathbf{B}^T\right)^{(l)}\right\|_1$$

where $\alpha$ is the turning parameter for a convex combination of the lasso and group lasso penalties. In our case, we realize what we want by the following R codes

```
set.seed(123)
library(SGL)

# implement the function with package SGL
linreg_row_sparse <- function(Factor, X, lambda = 0.01, alpha = 0.85) {
  N <- ncol(X)
  K <- ncol(Factor)
  index <- rep(1:K, N)
  data <- list(x = diag(N) %x% Factor, y = as.vector(X))
  beta <- SGL(data, index, type = "linear", lambdas = lambda / N, alpha = alpha,
              thresh = 1e-5, standardize = FALSE)$beta
  B <- t(matrix(beta, K, N, byrow = FALSE))
  return(B)
}
```

Then, we generate data with some factors only influencing one among $N$ stocks, i.e., these factors are low-impact.

```
n_lowimp <- 4
F_ <- mvrnorm(n = T, mu = rep(0, n_lowimp+K), Sigma = diag(n_lowimp+K))

B_ <- cbind(B_real, matrix(0, N, n_lowimp))
for (i in 1:n_lowimp) {
  B_[i, K+i] <- 0.5
}
print(B_)
#>             [,1]       [,2]       [,3] [,4] [,5] [,6] [,7]
#>  [1,] -0.7152422  1.3011760  0.0000000  0.5  0.0  0.0  0.0
#>  [2,] -0.7526890  0.7567748  1.2181086  0.0  0.5  0.0  0.0
#>  [3,] -0.9385387 -1.7267304 -1.3387743  0.0  0.0  0.5  0.0
#>  [4,] -1.0525133 -0.6015067  0.6608203  0.0  0.0  0.0  0.5
#>  [5,]  0.0000000  0.0000000 -0.5229124  0.0  0.0  0.0  0.0
#>  [6,]  0.0000000  0.7035239  0.6837455  0.0  0.0  0.0  0.0
#>  [7,] -2.0142105  0.0000000  0.0000000  0.0  0.0  0.0  0.0
#>  [8,]  0.0000000 -1.2586486  0.6329607  0.0  0.0  0.0  0.0
#>  [9,]  1.2366750  1.6844357  1.3355176  0.0  0.0  0.0  0.0
#> [10,]  2.0375740  0.9113913  0.0000000  0.0  0.0  0.0  0.0
X_ <- F_ %*% t(B_) + 0.01 * matrix(rnorm(T*N), T, N)
```

We then compare the differences between element-wise sparse and row-wise sparse regression.

```
B_elesparse <- linreg_ele_sparse(F_, X_, penalty = "lasso", lambda = 0.3)$B
B_rowsparse <- linreg_row_sparse(F_, X_, lambda = 0.3, alpha = 0.2)

print(B_)
#>             [,1]       [,2]       [,3] [,4] [,5] [,6] [,7]
#>  [1,] -0.7152422  1.3011760  0.0000000  0.5  0.0  0.0  0.0
#>  [2,] -0.7526890  0.7567748  1.2181086  0.0  0.5  0.0  0.0
#>  [3,] -0.9385387 -1.7267304 -1.3387743  0.0  0.0  0.5  0.0
#>  [4,] -1.0525133 -0.6015067  0.6608203  0.0  0.0  0.0  0.5
#>  [5,]  0.0000000  0.0000000 -0.5229124  0.0  0.0  0.0  0.0
#>  [6,]  0.0000000  0.7035239  0.6837455  0.0  0.0  0.0  0.0
#>  [7,] -2.0142105  0.0000000  0.0000000  0.0  0.0  0.0  0.0
#>  [8,]  0.0000000 -1.2586486  0.6329607  0.0  0.0  0.0  0.0
#>  [9,]  1.2366750  1.6844357  1.3355176  0.0  0.0  0.0  0.0
#> [10,]  2.0375740  0.9113913  0.0000000  0.0  0.0  0.0  0.0
print(B_elesparse)
#>             [,1]       [,2]       [,3]      [,4]      [,5]      [,6]       [,7]
#>  [1,] -0.4672228  1.0303872  0.0000000 0.2074709 0.0000000 0.0000000 0.00000000
#>  [2,] -0.4967683  0.3758835  0.8502721 0.0000000 0.1161491 0.0000000 0.00000000
#>  [3,] -0.5907305 -1.3630490 -1.0550118 0.0000000 0.0000000 0.1904491 0.00000000
#>  [4,] -0.7142322 -0.2500819  0.2563827 0.0000000 0.0000000 0.0000000 0.08978659
#>  [5,]  0.0000000  0.0000000 -0.2029512 0.0000000 0.0000000 0.0000000 0.00000000
#>  [6,]  0.0000000  0.3654966  0.3474925 0.0000000 0.0000000 0.0000000 0.00000000
#>  [7,] -1.7212507  0.0000000  0.0000000 0.0000000 0.0000000 0.0000000 0.00000000
#>  [8,]  0.0000000 -0.9518036  0.3266415 0.0000000 0.0000000 0.0000000 0.00000000
#>  [9,]  0.9110410  1.3026263  1.0140192 0.0000000 0.0000000 0.0000000 0.00000000
#> [10,]  1.7007091  0.5442624  0.0000000 0.0000000 0.0000000 0.0000000 0.00000000
print(B_rowsparse)
#>             [,1]       [,2]        [,3] [,4] [,5] [,6] [,7]
#>  [1,] -0.5672868  0.9521714  0.00164297    0    0    0    0
#>  [2,] -0.5698816  0.4169104  0.70405147    0    0    0    0
#>  [3,] -0.6291273 -1.1877945 -0.84138534    0    0    0    0
#>  [4,] -0.7616124 -0.3750200  0.30986008    0    0    0    0
#>  [5,]  0.0000000  0.0000000 -0.29007523    0    0    0    0
#>  [6,]  0.0000000  0.4614279  0.38507470    0    0    0    0
#>  [7,] -1.5377723  0.0000000  0.00000000    0    0    0    0
#>  [8,]  0.0000000 -0.8912436  0.37153231    0    0    0    0
#>  [9,]  0.8944246  1.1460135  0.80219627    0    0    0    0
#> [10,]  1.5247966  0.5781963  0.00000000    0    0    0    0
```

Obviously, we can obtain the row-sparse $\mathbf{B}$ using sparse-group lasso by properly choosing penalty coefficient $\lambda$ and $\alpha$.

### 4.2.2 Subset selection

In machine learning, there exists a classical method called subset selection. We found a function `best.r.sq()` from a recently released R package `mvabund`, which implements a forward selection in a multivariate linear model.

```
library(mvabund)
best.r.sq( X_~F_ )
#> $xs
#> [1] 1 2 3
#>
```

```
#> $r2Step
#>   F_[, 1]  +F_[, 2]  +F_[, 3]
#> 0.3379401 0.6656976 0.9675958
#>
#> $r2Matrix
#>              Step 1    Step 2    Step 3
#> F_[, 1] 0.33794014        NA        NA
#> F_[, 2] 0.31615795 0.6656976        NA
#> F_[, 3] 0.29150681 0.6309482 0.9675958
#> F_[, 4] 0.02937063 0.3650031 0.6868668
#> F_[, 5] 0.01793328 0.3543163 0.6734623
#> F_[, 6] 0.02076671 0.3561918 0.6799328
#> F_[, 7] 0.01915951 0.3556588 0.6843262
```

Then, we can perform the trivial linear factor model regression with chosen factors.

# References


[1] R. S. Tsay, *Analysis of financial time series*. John Wiley & Sons, 2005.

[2] P. Kempthorne, "Topics in mathematics with applications in finance," *(Massachusetts Institute of Technology: MIT OpenCouseWare), http://ocw.mit.edu (Accessed 18 Aug, 2018). License: Creative Commons BY-NC-SA*.

[3] K. Khamaru and R. Mazumder, "Computation of the maximum likelihood estimator in low-rank factor analysis," *arXiv preprint arXiv:1801.05935*, 2018.