Estimation of covariance matrix via factor models

Rui Zhou and Daniel P. Palomar

2018-05-17

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 results of covariance matrix estimation by using our package `covFactorModel` and another package `FinCovRegularization`, which also estimate the covariance matrix of asset returns via three types of factor models as we do. We start by loading built-in data from package `FinCovRegularization`:

```r
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
Indicator <- matrix(0, 10, 3)
dimnames(Indicator) <- list(colnames(assets), c("Drug", "Auto", "Oil"))
Indicator[c("ABT", "LLY", "MRK", "PFE"), "Drug"] <- 1
Indicator[c("F", "GM"), "Auto"] <- 1
Indicator[c("BP", "CVX", "RD", "XOM"), "Oil"] <- 1
sector_info <- c(rep(1, 4),
```

```r
                 rep(2, 2),
                 rep(3, 4))
```

Then, we use the two packages to compute covariance matrix estimation via three factor models and compare the results:

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

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

# compare cov by statistical factor model
cov_stat <- covFactorModel(assets_xts, type = "S", K = 3)
my_cov_stat <- StatFactor.Cov(assets, 3)
norm(cov_stat - my_cov_stat, "F")
#> [1] 3.842331e-17
```

It is clear that the results from `covFactorModel` and `FinCovRegularization` are exactly the same. Note that different from the three individual functions in `FinCovRegularization`, the package `covFactorModel` provides one uniform function that with the argument `type` to choose the type of factor model. 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. Besides, `covFactorModel` allows user to choose different structures on residual covariance matrix, while `FinCovRegularization` assumes it to be diagonal only.

Next, we compare the performance of `covFactorModel()` and `factanal()` in covariance matrix estimation. By description (use `?factanal` for details) of `factanal()`, 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 proximal estimation performance but be much quicker compared with `factanal()`.

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`:

```r
library(xts)
library(quantmod)

# set begin-end date and stock namelist
```

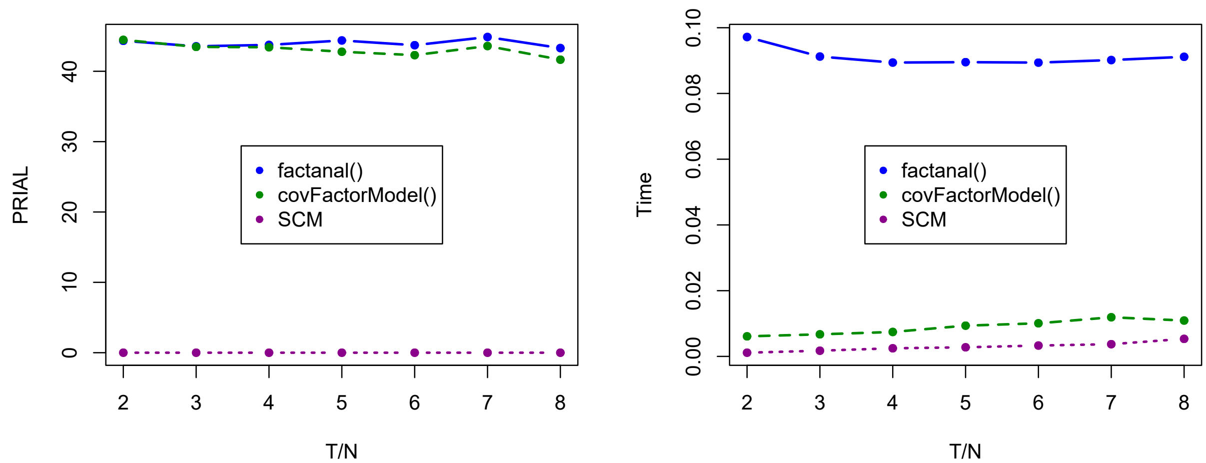


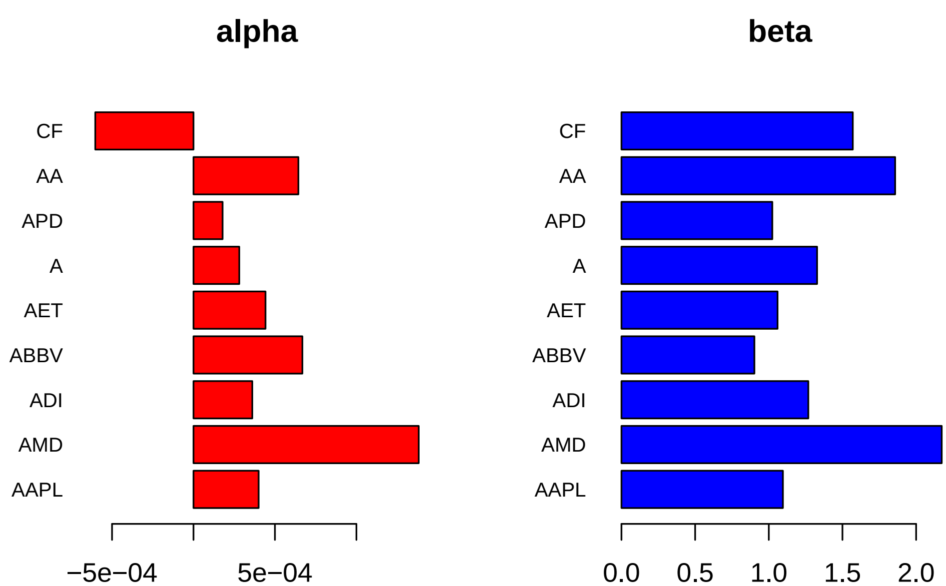
Figure 1: Average PRIAL and running time.

```
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 51.54218 52.58730 106.3503 39.80076 111.3778 23.00764 36.30703
#> 2016-01-05  98.10455 2.75 51.16347 52.36822 107.8298 39.66381 109.3862 21.96506 35.16852
#> 2016-01-06  96.18465 2.51 48.98590 52.37734 107.2848 39.83988 106.7223 20.40121 32.12948
#> 2016-01-07  92.12524 2.28 47.73616 52.22216 107.3334 38.14769 103.1445 19.59558 30.60548
#> 2016-01-08  92.61236 2.14 47.31958 50.79818 104.2186 37.74665 102.6100 19.12169 30.31861
#> 2016-01-11  94.11198 2.34 48.44624 49.18250 102.5152 37.11086 103.0152 18.95583 29.09045
tail(data_set)
#>                AAPL   AMD      ADI     ABBV      AET        A      APD    AA       CF
#> 2017-12-21 173.6298 10.89 88.19117 96.21605 179.7195 67.22079 161.0903 48.99 40.62705
#> 2017-12-22 173.6298 10.54 88.39013 96.51086 178.9538 67.05154 161.3470 49.99 41.08011
#> 2017-12-26 169.2248 10.46 88.17127 96.05882 179.4112 66.95200 160.9324 50.38 42.20289
#> 2017-12-27 169.2546 10.53 88.63884 96.39293 179.8388 67.00177 161.5939 51.84 42.41957
#> 2017-12-28 169.7308 10.55 88.91739 96.09813 180.2167 67.15110 162.5318 54.14 41.96652
#> 2017-12-29 167.8954 10.28 88.56920 95.03680 179.3814 66.82085 162.9390 53.87 41.89757

# download SP500 Index data from YahooFinance
SP500_index <- Ad(getSymbols("^GSPC", from = begin_date, to = end_date, auto.assign = FALSE))
```

```r
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:

```r
library(covFactorModel)
# compute log-return
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 = "M", 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] 1.000912e-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 = "B")
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] 2.162987e-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] 4.754236e-17
```

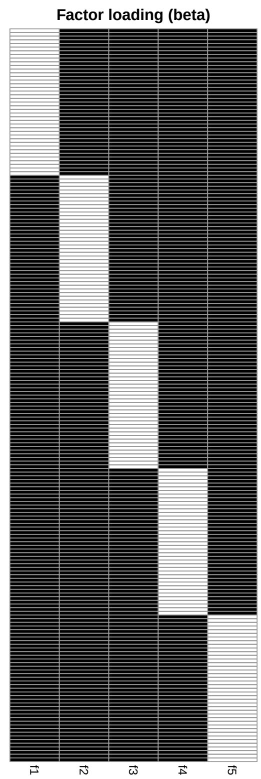
2.2 Usage of `covFactorModel()`

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., diagonal, block diagonal, scaled identity, and full).

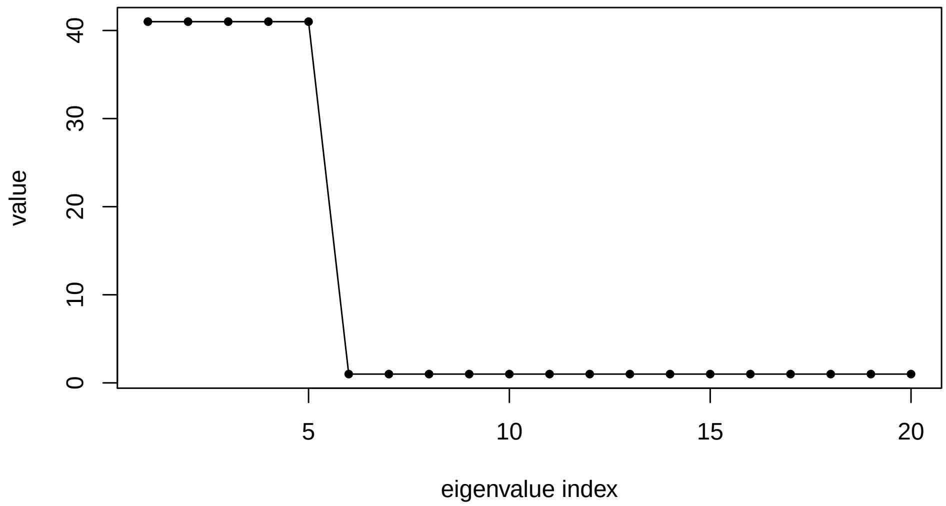
Firstly, we compare covariance matrix estimation based on different factor model decomposition. Let's start by preparing some parameters for the synthetic data generation:

```
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_sector <- 5 # num of sectors
stock_sector_info <- rep(1:num_sector, each = N/num_sector)
# generate beta following BARRA model
beta <- matrix(0, N, num_sector)
for (i in 1:num_sector) {
```

```r
  mask <- stock_sector_info == i
  beta[mask, i] <- 1
}
# show beta
colnames(beta) <- paste("f", 1:num_sector, sep = "")
pheatmap(beta, cluster_rows = FALSE, cluster_cols = FALSE, color = c(1, 0), legend = FALSE,
         main = "Factor loading (beta)")
```



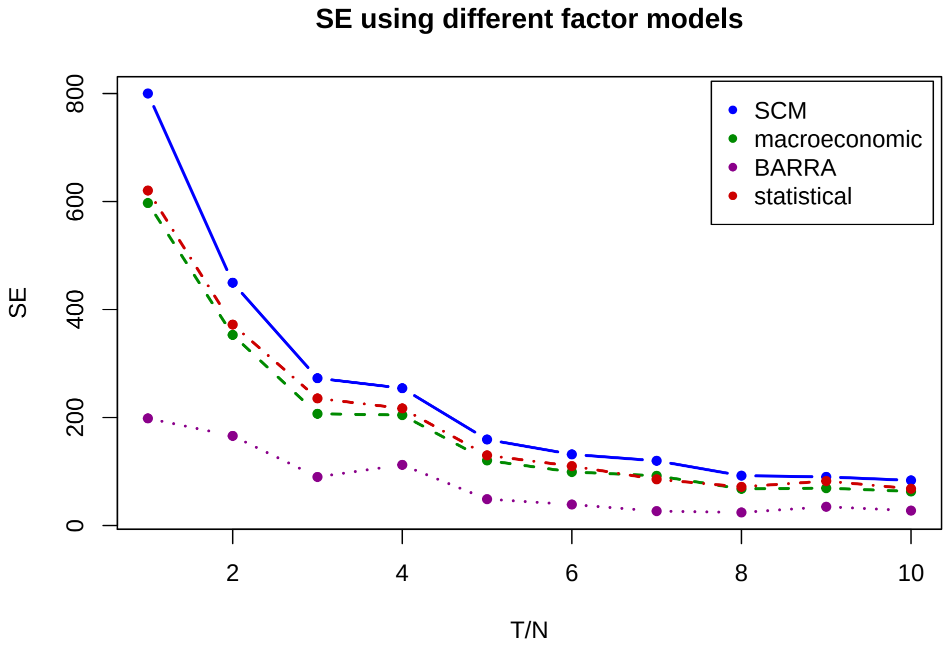

```r
Psi <- diag(N)
Sigma_f <- diag(num_sector)
Sigma <- beta %*% Sigma_f %*% t(beta) + Psi

# 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) $\|\hat{\boldsymbol{\Sigma}} - \boldsymbol{\Sigma}_{\mathsf{true}}\|_F^2$ w.r.t. number of observations:

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

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

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

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

  # use statistical factor model with diagonal Psi (default)
  cov_stat <- covFactorModel(X, K = num_sector)
  err_stat <- c(err_stat, norm(Sigma - cov_stat, "F")^2)
}
res <- cbind("SCM"           = err_scm,
             "macroeconomic" = err_macroecon,
             "BARRA"         = err_barra,
             "statistical"   = err_stat)
colors <- c("blue", "green4", "darkmagenta", "red3")
matplot(index_T/N, res,
        xlab = "T/N", ylab = "SE",
        main = "SE using different factor models",
```

```r
        type = "b", pch = 20, lwd = 2, col = colors)
legend("topright", inset = 0.01, legend = colnames(res), pch = 20, col = colors)
```

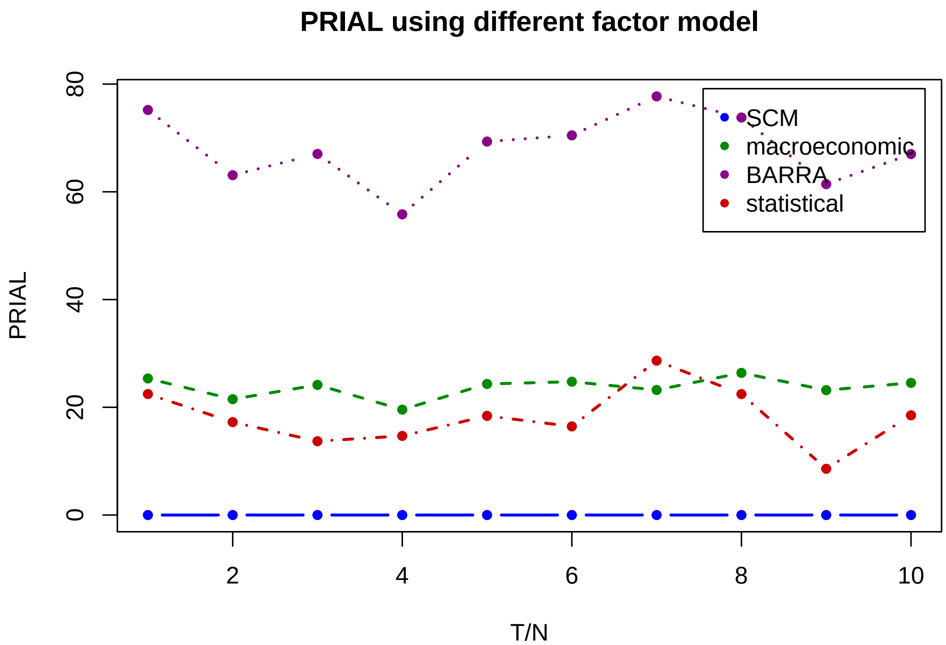


Obviously, using factor models for covariance matrix estimation definitely helps (note that BARRA is definitely the best simply because the synthetic data was generated according to the BARRA model). In order to show how well the estimated covariance matrices do compared to the sample covariance matrix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

$$\mathsf{PRIAL} = 100 \times \frac{\|\boldsymbol{\Sigma}_{\mathsf{scm}} - \boldsymbol{\Sigma}_{\mathsf{true}}\|_F^2 - \|\hat{\boldsymbol{\Sigma}} - \boldsymbol{\Sigma}_{\mathsf{true}}\|_F^2}{\|\boldsymbol{\Sigma}_{\mathsf{scm}} - \boldsymbol{\Sigma}_{\mathsf{true}}\|_F^2}$$

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

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

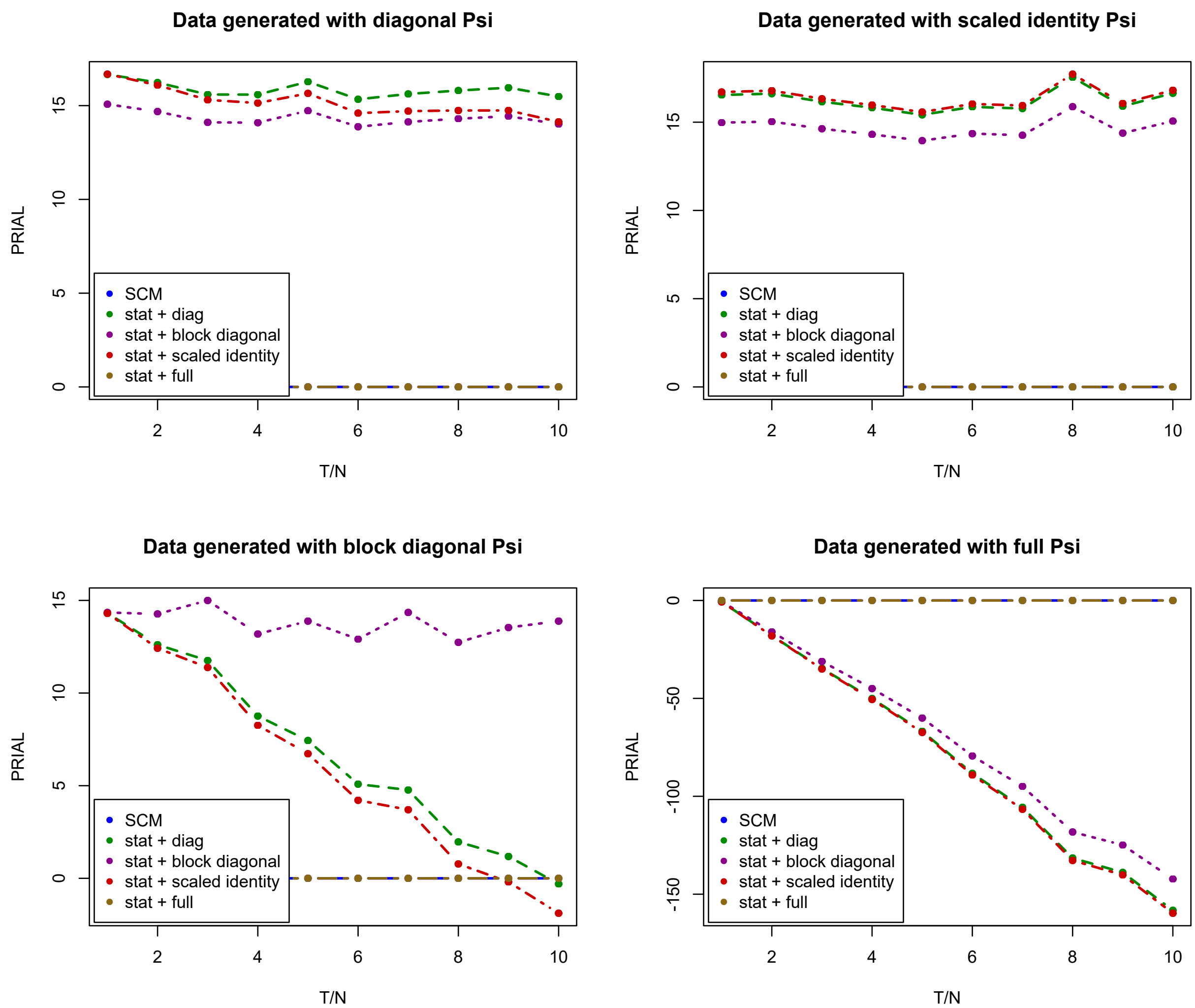


Figure 2: Performance of estimation under different Psi structures.

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, α_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 combination $\mathbf{F} = \begin{bmatrix} \mathbf{f}_1 & \cdots & \mathbf{f}_K \end{bmatrix}$, $\mathbf{x}_i = [x_{i,1}, \ldots, x_{i,T}]^T$, 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:

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

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 $\boldsymbol{\alpha} = \mathbf{0}$):

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

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

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

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

3.1.3 Statistical factor model (aka implicit factor model)

The statistical factor model holds the assumption 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. We use the following iterative method [see [1] for details] to estimate the parameters:

Algorithm 1

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

where struct() is to impose certain structure on $\hat{\boldsymbol{\Psi}}_{(s)}$, 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 statistical factor model use algorithm output:

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

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

The function `covFactorModel()` estimates a covariance matrix 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 $\boldsymbol{\Sigma}$ can be written as

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

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

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

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

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

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

References

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