Using Robust Covariance Maxtrix Estimators in PA

Yifu Kang

2022-07-24

This vignette aims at showing users how to import robust covariance matrix estimators to mitigate the influence of outliers when performing optimal portfolio construction in the package **PortfolioAnalytics**. It is well-known that outliers can adversely influence all classical estimates, including especially sample covariance matrix estimates, which are a foundational element of mean-variance optimal (MVO) portfolios.

It is very common to find two types of outliers in assets returns. The first one is Cross-Section Outliers (CSO), which refers to outliers that show up for most of the assets in portfolio at a specific time. For example, in 1987, a global severe stock market crash, known as Black Monday, drastically reduced the returns of most of the assets, which can thus be viewed as CSO's. The second type is the Independent Outliers in Assets (IOA), in which outliers are mutually uncorrelated among assets. This type of outliers is also called the 'cell-wise contamination' in the robust statistic literature. Details of concerning these two types of outliers, and their applications to MVO portfolios are discussed in Martin (2013).

The existence of CSO and IOA often adversely influences the accuracy of covariance matrix estimates, thereby adversely influencing mean-variance optimization results. We integrate the functions *covRob* from **RobStatTM**, *covMcd* from **robustbase** and *TSGS* from **GSE** packages into **PortfolioAnalytics**. Incorporating robust covariance matrix estimates from the above functions ensures users a more reliable mean-variance optimal(MVO) portfolio that is not much influenced by outliers. This vignette will guide users on how to use newly created moment functions that have been incorporated into PortfolioAnalytics and can be accessed using parameters in the call to the *optimize.portfolio()* function or via the already available approach of writing a moment generating function from scratch.

1. Import Packages and Data

We first import necessary packages.

```
suppressMessages(library(PortfolioAnalytics))
suppressMessages(library(CVXR))
suppressMessages(library(PCRA))
suppressMessages(library(data.table))
```

In this vignette, we will use the weekly return data of 30 small capitalization stocks in package **PCRA** to construct portfolios. The time interval for the stock returns is the seven year period from January 1, 2006 to December 31, 2012.

```
stockItems <- c("Date","TickerLast","CapGroupLast","Return","MktIndexCRSP",
                "Ret13WkBill")
dateRange    <- c("2006-01-01","2012-12-31")
stocksDat    <- selectCRSPandSPGMI("weekly",dateRange = dateRange, stockItems =
                                   stockItems, factorItems = NULL)
stocksDat    <- stocksDat[CapGroupLast == "SmallCap"]
returnsAll   <- returnsCRSPxts(stocksDat) # dim = (168,108)
tickers      <- unique(stocksDat[,TickerLast])
```

```
tickers30    <- tickers[1:30]
colnames     <- c(tickers30,"Market")
returns30Mkt <- returnsAll[,colnames]

returns <- returns30Mkt[ ,1:30]
MARKET <- returns30Mkt$Market
```

2. Set Up Portfolio

Using PortfolioAnalytics, we compute the minimum variance portfolio with full-investment and long-only constraints. The mathematical formulation is

$$\begin{aligned} \min_{w} \quad & w^T Q w \\ \text{s.t.} \quad & e^T w = 1 \\ & w \geq 0 \end{aligned} \tag{1}$$

where w is the sample mean and Q is the sample covariance matrix of the return data.
In *PortfolioAnalytics*, we can set up portfolio with the following code. The optimized portfolio is called "GmvLo", which is short for Global Minimum Variance Long-only Portfolio .

```
funds <- colnames(returns)
pspec <- portfolio.spec(assets=tickers30)
pspec <- add.constraint(pspec, type="full_investment")
pspec <- add.constraint(pspec, type="long_only")
pspec <- add.objective(pspec, type="risk", name="var")
```

3. Use Robust Estimators in Optimal Portfolio Construction

In this section we show how to use the three newly created functions for estimating a robust covariance matrix in the process of portfolio optimization in **PortfolioAnalytics**. The way to optimize the previous portfolio with conventional sample mean and sample covariance is using function *optimize.portfolio*.

```
optimize.portfolio(returns, pspec, optimize_method="CVXR")
```

```
## ***********************************
## PortfolioAnalytics Optimization
## ***********************************
##
## Call:
## optimize.portfolio(R = returns, portfolio = pspec, optimize_method = "CVXR")
##
## Optimal Weights:
##    AAN    ABM   AEGN    AIN    AIT   ALOG    AMD   AMWD   APOG   ASNA   ASTE
## 0.0346 0.0536 0.0000 0.0000 0.0000 0.0471 0.0000 0.0000 0.0000 0.0000 0.0000
##   ATRI    ATW    AVP    AXE   AXLL    AZZ      B     BC   BCPC    BGG    BIG
## 0.1028 0.0000 0.0826 0.0000 0.0000 0.0000 0.0000 0.0000 0.0627 0.0000 0.0093
##    BMI    BMS   BOBE    BRC    CAL   CASY   CATO    CBB
## 0.0000 0.4676 0.0000 0.0000 0.0000 0.1305 0.0000 0.0092
##
## Objective Measures:
##  StdDev
```

```
## 0.02942
```

The optimization method choice *CVXR* provides us the access to the solvers in package **CVXR**.

3.1 CovRob Estimators

The first method to compute robust estimators is based on *covRob* function from package **RobStatTM**. There are two ways for users to apply this method, either using the function we provide in the PortfolioAnalytics or custom a function themselves.

In **PortfolioAnalytics**, we build a function called *custom.covRob* based on method *covRob* from package **RobStatTM**. In this way, users can apply **covRob** by setting the parameter **momentFUN** equal to "custom.covRob" in method **optimize.portfolio()**. Meanwhile, users can set other parameters of **covRob** in **optimize.portfolio()**. See example below.

```
# This is the function we set up for users in advance in PortfolioAnalytics
custom.covRob <- function(R, ...){
  out <- list()
  if(hasArg(type)) type=match.call(expand.dots=TRUE)$type else type="auto"
  if(hasArg(tol)) tol=match.call(expand.dots=TRUE)$tol else tol=1e-4
  if(hasArg(maxit)) maxit=match.call(expand.dots=TRUE)$maxit else maxit=50

  robustCov <- RobStatTM::covRob(X=R, type=type, tol=tol, maxit=maxit)

  out$sigma <- robustCov$cov
  out$mu <- robustCov$center
  return(out)
}

# One can modify the parameter in this way
optimize.portfolio(returns, pspec,
                   optimize_method="CVXR",
                   momentFUN="custom.covRob",
                   type="MM", maxit=100, tol=1e-5)
```

```
## **********************************
## PortfolioAnalytics Optimization
## **********************************
##
## Call:
## optimize.portfolio(R = returns, portfolio = pspec, optimize_method = "CVXR",
##     type = "MM", maxit = 100, tol = 1e-05, momentFUN = "custom.covRob")
##
## Optimal Weights:
##    AAN    ABM   AEGN    AIN    AIT   ALOG    AMD   AMWD   APOG   ASNA   ASTE
## 0.0415 0.0736 0.0000 0.0000 0.0000 0.0565 0.0000 0.0000 0.0000 0.0000 0.0000
##   ATRI    ATW    AVP    AXE   AXLL    AZZ      B     BC   BCPC    BGG    BIG
## 0.1670 0.0000 0.1096 0.0000 0.0000 0.0108 0.0000 0.0000 0.0000 0.0000 0.0000
##    BMI    BMS   BOBE    BRC    CAL   CASY   CATO    CBB
## 0.0000 0.3678 0.0000 0.0000 0.0000 0.1634 0.0000 0.0098
##
## Objective Measures:
##  StdDev
## 0.02509
```

Alternatively, users can use existing functionality to create a function themselves. In this way, users can have more possibilities to explore while customing the covariance matrix. If they want to make some adjustments to the outcome matrix of **covRob**, or if they want to add some code based on original algorithms, this would be a better choice. The key here is the return of the function must be a list containing **mu** and **sigma**. See example below:

```
users.covRob <- function(R){
  out <- list()

  robustCov <- RobStatTM::covRob(X=R, type="MM", maxit=100, tol=1e-5)

  out$sigma <- robustCov$cov
  out$mu <- robustCov$center
  return(out)
}

optimize.portfolio(returns, pspec,
                   optimize_method="CVXR",
                   momentFUN="users.covRob")
```

```
## **********************************
## PortfolioAnalytics Optimization
## **********************************
##
## Call:
## optimize.portfolio(R = returns, portfolio = pspec, optimize_method = "CVXR",
##     momentFUN = "users.covRob")
##
## Optimal Weights:
##    AAN    ABM   AEGN    AIN    AIT   ALOG    AMD   AMWD   APOG   ASNA   ASTE
## 0.0415 0.0736 0.0000 0.0000 0.0000 0.0565 0.0000 0.0000 0.0000 0.0000 0.0000
##   ATRI    ATW    AVP    AXE   AXLL    AZZ      B     BC   BCPC    BGG    BIG
## 0.1670 0.0000 0.1096 0.0000 0.0000 0.0108 0.0000 0.0000 0.0000 0.0000 0.0000
##    BMI    BMS   BOBE    BRC    CAL   CASY   CATO    CBB
## 0.0000 0.3678 0.0000 0.0000 0.0000 0.1634 0.0000 0.0098
##
## Objective Measures:
##  StdDev
## 0.02509
```

The above code uses default settings to compute robust estimators with the *covRob* function.. The available parameters are
1. **type**: which estimator to use. Possible options are **MM**, **Rocke** and **auto**. Default is **auto**.
2. **maxiter**: maximum number of iterations. Default is 50.
3. **tol**: tolerance for covergence. Default is 1e-4.

3.2 covMcd Robust Estimators

The second method to compute robust statistics is based on *covMcd* from package **robustbase**. This approach is similar to *covRob* but has more parameters. *Mcd* method looks for $\boldsymbol{h}$ observations (out of n) whose classical covariance matrix has the lowest possible determinant. Then the raw MCD estimates of location and scatter are computed based on the average and covariance matrix of these h points.

Similarly, **PortfolioAnalytics** provides two ways for users to apply *Mcd* robust estimators. For the first approach, we add two functions in **PortfolioAnalytics**. The first one, *custom.covMcd*, takes advantage

of *covMcd* function from package **robustbase** to compute covariance matrix.

```
custom.covMcd <- function(R, ...){

  if(hasArg(control)) control=match.call(expand.dots=TRUE)$control else control=MycovMcd()
  if(hasArg(alpha)) alpha=match.call(expand.dots=TRUE)$alpha else alpha=control$alpha
  if(hasArg(nsamp)) nsamp=match.call(expand.dots=TRUE)$nsamp else nsamp=control$nsamp
  if(hasArg(nmini)) nmini=match.call(expand.dots=TRUE)$nmini else nmini=control$nmini
  if(hasArg(kmini)) kmini=match.call(expand.dots=TRUE)$kmini else kmini=control$kmini
  if(hasArg(scalefn)) scalefn=match.call(expand.dots=TRUE)$scalefn else scalefn=control$scalefn
  if(hasArg(maxcsteps)) maxcsteps=match.call(expand.dots=TRUE)$maxcsteps
  else maxcsteps=control$maxcsteps

  if(hasArg(initHsets)) initHsets=match.call(expand.dots=TRUE)$initHsets
  else initHsets=control$initHsets

  if(hasArg(seed)) seed=match.call(expand.dots=TRUE)$seed else seed=control$seed
  if(hasArg(tolSolve)) tolSolve=match.call(expand.dots=TRUE)$tolSolve else tolSolve=control$tolSolve
  if(hasArg(wgtFUN)) wgtFUN=match.call(expand.dots=TRUE)$wgtFUN else wgtFUN=control$wgtFUN

  if(hasArg(use.correction)) use.correction=match.call(expand.dots=TRUE)$use.correction
  else use.correction=control$use.correction


  robustMCD <- robustbase::covMcd(x=R, alpha=alpha,
                                  nsamp=nsamp, nmini=nmini,
                                  kmini=kmini, seed=seed,
                                  tolSolve=tolSolve, scalefn=scalefn,
                                  maxcsteps=maxcsteps,
                                  initHsets=initHsets,
                                  wgtFUN=wgtFUN, use.correction=use.correction)

  return(list(mu = robustMCD$center, sigma = robustMCD$cov))
}
```

And the second one *MycovMcd* helps with parameter setting. This function returns a list containing possible parameters for *covMcd*.

```
MycovMcd <- function(alpha = 1/2,
           nsamp = 500, nmini = 300, kmini = 5,
           scalefn = "hrv2012", maxcsteps = 200,
           seed = NULL, tolSolve = 1e-14,
           wgtFUN = "01.original", beta,
           use.correction=TRUE
  ){
    if(missing(beta) || !is.numeric(beta))
      beta <- 0.975

    return(list(alpha=alpha, nsamp=nsamp,
                nmini=as.integer(nmini), kmini=as.integer(kmini),
                seed = as.integer(seed),
                tolSolve=tolSolve, scalefn=scalefn,
                maxcsteps=as.integer(maxcsteps),
                wgtFUN=wgtFUN, beta=beta,
```

```
                use.correction=use.correction))
  }
```

Users can apply following code to optimize portfolio with *covMcd* estimators.

```
# pass parameter in optimize.portfolio
optimize.portfolio(returns, pspec,
                   optimize_method="CVXR",
                   momentFUN="custom.covMcd",
                   alpha=0.75, nsamp=600)
```

```
## **********************************
## PortfolioAnalytics Optimization
## **********************************
##
## Call:
## optimize.portfolio(R = returns, portfolio = pspec, optimize_method = "CVXR",
##     alpha = 0.75, nsamp = 600, momentFUN = "custom.covMcd")
##
## Optimal Weights:
##    AAN    ABM   AEGN    AIN    AIT   ALOG    AMD   AMWD   APOG   ASNA   ASTE
## 0.0397 0.0553 0.0000 0.0000 0.0000 0.0813 0.0000 0.0000 0.0000 0.0000 0.0000
##   ATRI    ATW    AVP    AXE   AXLL    AZZ      B     BC   BCPC    BGG    BIG
## 0.1445 0.0000 0.0971 0.0000 0.0000 0.0193 0.0000 0.0000 0.0000 0.0000 0.0000
##    BMI    BMS   BOBE    BRC    CAL   CASY   CATO    CBB
## 0.0000 0.3918 0.0000 0.0000 0.0000 0.1578 0.0000 0.0131
##
## Objective Measures:
##  StdDev
## 0.02527
```

Also, users can first create a parameter list using *MycovMcd*. The pass the list into *optimize.portfolio*.

```
# use MycovMcd
covMcd.params <- MycovMcd(alpha=0.75, nsamp=600)
optimize.portfolio(returns, pspec,
                   optimize_method="CVXR",
                   momentFUN="custom.covMcd",
                   control=covMcd.params)
```

```
## **********************************
## PortfolioAnalytics Optimization
## **********************************
##
## Call:
## optimize.portfolio(R = returns, portfolio = pspec, optimize_method = "CVXR",
##     control = covMcd.params, momentFUN = "custom.covMcd")
##
## Optimal Weights:
##    AAN    ABM   AEGN    AIN    AIT   ALOG    AMD   AMWD   APOG   ASNA   ASTE
## 0.0380 0.0582 0.0000 0.0000 0.0000 0.0635 0.0000 0.0000 0.0000 0.0000 0.0000
##   ATRI    ATW    AVP    AXE   AXLL    AZZ      B     BC   BCPC    BGG    BIG
## 0.1527 0.0000 0.1053 0.0000 0.0000 0.0190 0.0000 0.0000 0.0000 0.0000 0.0000
##    BMI    BMS   BOBE    BRC    CAL   CASY   CATO    CBB
## 0.0000 0.3850 0.0000 0.0000 0.0000 0.1679 0.0000 0.0103
```

```
##
## Objective Measures:
##  StdDev
## 0.02535
```

Users already have the option to incorporate the *covMcd* estimation function by creating their own function to pass into optimize.portfolio through the *momentFUN* parameter. The following example illustrates how this could be accomplished.

```
users.covMcd <- function(R){

  robustMCD <- robustbase::covMcd(x=R, alpha=0.75, nsamp=600)

  return(list(mu = robustMCD$center, sigma = robustMCD$cov))
}

optimize.portfolio(returns, pspec,
                   optimize_method="CVXR",
                   momentFUN="users.covMcd")
```

```
## **********************************
## PortfolioAnalytics Optimization
## **********************************
##
## Call:
## optimize.portfolio(R = returns, portfolio = pspec, optimize_method = "CVXR",
##     momentFUN = "users.covMcd")
##
## Optimal Weights:
##    AAN    ABM   AEGN    AIN    AIT   ALOG    AMD   AMWD   APOG   ASNA   ASTE
## 0.0484 0.0638 0.0000 0.0000 0.0000 0.0652 0.0000 0.0000 0.0000 0.0000 0.0000
##   ATRI    ATW    AVP    AXE   AXLL    AZZ      B     BC   BCPC    BGG    BIG
## 0.1431 0.0000 0.0983 0.0000 0.0000 0.0104 0.0000 0.0000 0.0000 0.0000 0.0000
##    BMI    BMS   BOBE    BRC    CAL   CASY   CATO    CBB
## 0.0000 0.3902 0.0000 0.0000 0.0000 0.1637 0.0000 0.0169
##
## Objective Measures:
##  StdDev
## 0.02576
```

3.3 TSGS Estimators

The third method for creating a robust covariance matrix is the use of the 2-step Generalized S-estimators(TSGS) from package **GSE**. TSGS computes robust estimation of multivariate location and scatter matrix in the presence of outliers. The first step of TSGS is to filter out the large cellwise outliers and replace them with NA's. The second step is using GSE to deal with high-dimensional casewise outliers that are undetected under step 1. **GSE** is a specifically designed estimators to find outliers in data with NA's. For more details refer to Agostinelli (2015).

We proceed in the same fashion as with the *covMcd* method. We add a function *custom.TSGS* which takes advantage of *TSGS* function from package **GSE** to compute covariance matrix.

```
custom.TSGS <- function(R, ...){
 if(hasArg(control)) control=match.call(expand.dots=TRUE)$control else control=MyTSGS()
 if(hasArg(filter)) filter=match.call(expand.dots=TRUE)$filter else filter=control$filter
```

```
 if(hasArg(partial.impute)) partial.impute=match.call(expand.dots=TRUE)$partial.impute
 else partial.impute=control$partial.impute

 if(hasArg(tol)) tol=match.call(expand.dots=TRUE)$tol else tol=control$tol
 if(hasArg(maxiter)) maxiter=match.call(expand.dots=TRUE)$maxiter else maxiter=control$maxiter
 if(hasArg(loss)) loss=match.call(expand.dots=TRUE)$loss else loss=control$loss
 if(hasArg(init)) init=match.call(expand.dots=TRUE)$init else init=control$init

  tsgsRob <- GSE::TSGS(x=R, filter=filter,
                       partial.impute=partial.impute, tol=tol,
                       maxiter=maxiter, method=loss,
                       init=init)

  return(list(mu = tsgsRob@mu, sigma = tsgsRob@S))

}
```

We also add an auxiliary function *MyTSGS* that helps with parameter setting.

```
MyTSGS <- function(filter=c("UBF-DDC","UBF","DDC","UF"),
                         partial.impute=FALSE, tol=1e-4, maxiter=150,
                         loss=c("bisquare","rocke"),
                         init=c("emve","qc","huber","imputed","emve_c")){

  filter <- match.arg(filter)
  loss <- match.arg(loss)
  init <- match.arg(init)

  return(list(filter=filter, partial.impute=partial.impute,
              tol=tol, maxiter=as.integer(maxiter),
              loss=loss,init))
}
```

```
optimize.portfolio(returns, pspec,
                   optimize_method="CVXR",
                   momentFUN="custom.TSGS")
```

```
## **********************************
## PortfolioAnalytics Optimization
## **********************************
##
## Call:
## optimize.portfolio(R = returns, portfolio = pspec, optimize_method = "CVXR",
##     momentFUN = "custom.TSGS")
##
## Optimal Weights:
##    AAN    ABM   AEGN    AIN    AIT   ALOG    AMD   AMWD   APOG   ASNA   ASTE
## 0.0288 0.0827 0.0000 0.0000 0.0000 0.0754 0.0000 0.0000 0.0000 0.0000 0.0000
##   ATRI    ATW    AVP    AXE   AXLL    AZZ      B     BC   BCPC    BGG    BIG
## 0.1720 0.0000 0.1919 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##    BMI    BMS   BOBE    BRC    CAL   CASY   CATO    CBB
## 0.0000 0.2722 0.0000 0.0000 0.0000 0.1771 0.0000 0.0000
##
```

```
## Objective Measures:
##  StdDev
## 0.02211
```

The second way to incorporate the *TSGS* estimators is for users to create a custom *TSGS* function. See examples below:

```
users.TSGS <- function(R){

  tsgsRob <- GSE::TSGS(x=R, tol=1e-5, maxiter=150)

  return(list(mu = tsgsRob@mu, sigma = tsgsRob@S))
}

optimize.portfolio(returns, pspec,
                   optimize_method="CVXR",
                   momentFUN="users.TSGS"
                  )
```

```
## **********************************
## PortfolioAnalytics Optimization
## **********************************
##
## Call:
## optimize.portfolio(R = returns, portfolio = pspec, optimize_method = "CVXR",
##     momentFUN = "users.TSGS")
##
## Optimal Weights:
##    AAN    ABM   AEGN    AIN    AIT   ALOG    AMD   AMWD   APOG   ASNA   ASTE
## 0.0286 0.0828 0.0000 0.0000 0.0000 0.0757 0.0000 0.0000 0.0000 0.0000 0.0000
##   ATRI    ATW    AVP    AXE   AXLL    AZZ      B     BC   BCPC    BGG    BIG
## 0.1720 0.0000 0.1916 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##    BMI    BMS   BOBE    BRC    CAL   CASY   CATO    CBB
## 0.0000 0.2723 0.0000 0.0000 0.0000 0.1770 0.0000 0.0000
##
## Objective Measures:
##  StdDev
## 0.02212
```

The *TSGS* method inlcudes a parameter to choose the filter in step 1. Possible options are "UBF-DDC","UBF","DDC","UF". The default choice is "UBF-DDC". The *TSGS* method also provides the parameter *init* for users to choose the type of initial estimator. Options include "emve", "qc", "huber","imputed","emve_c".
Similar to *covMCD*, users can either directly pass in value or use a control function *MyTSGS*. For more details of above parameters and other parameters please refer to function *MyTSGS* in PA manual.

4.Robust Covariance Matrix Estimtors Backtests

In this section we conduct backtests for each of the three robust estimators using function *optimize.portfolio.rebalancing* in **PortfolioAnalytics**. The training period for the backtests is 100 weeks, and the portfolio is rebalanced weekly. To show the backtest results in a clear way, we create the function shown below to plot the results. The function has a parameter called *plot* to control which robsut estimator is used in backtest: 1 is default choice for *covRov*, 2 is for *covMcd* and 3 is for *TSGS*.

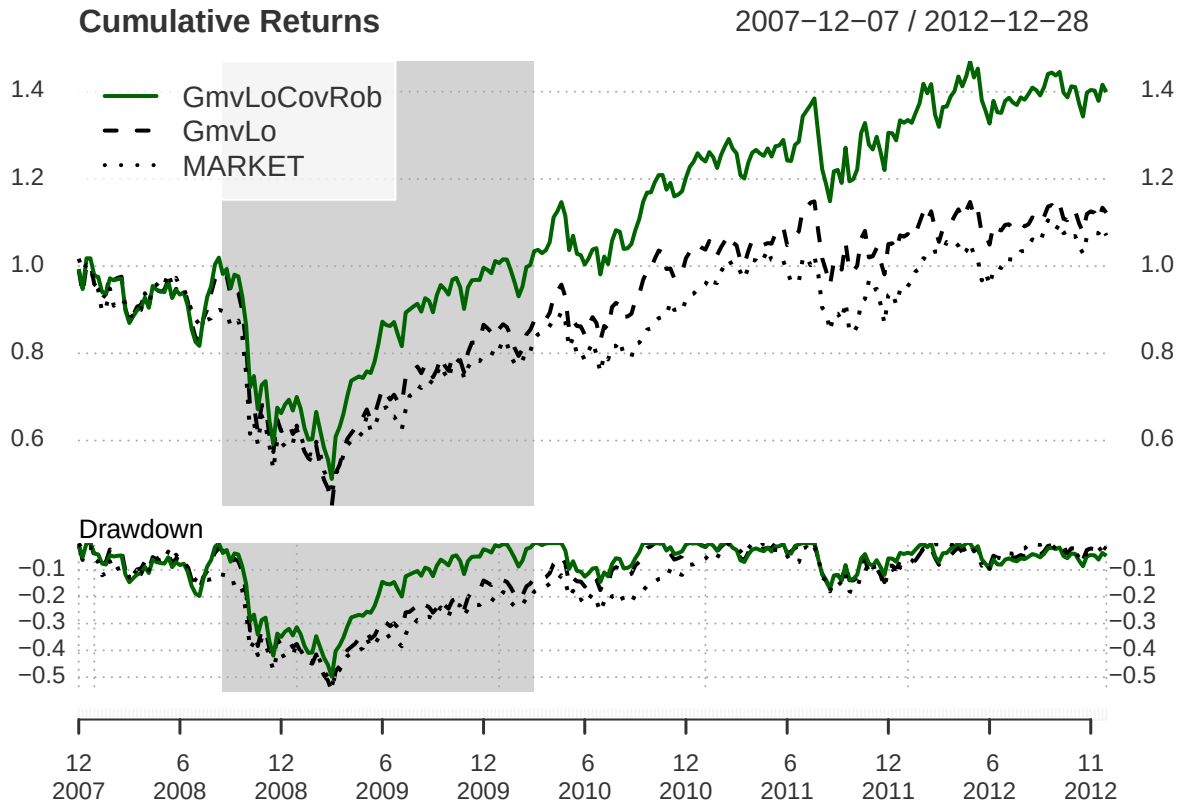
```r
# Plot function
robPlot <- function(GMV, MAERKET, plot=1){
  # Optimize Portfolio at Monthly Rebalancing and 5-Year Training
  if(plot == 1){
    momentEstFun = 'custom.covRob'
    name = "GmvLoCovRob"
  }else if(plot == 2){
    momentEstFun = 'custom.covMcd'
    name = "GmvLoCovMcd"
  }else if(plot == 3){
    momentEstFun = 'custom.TSGS'
    name = "GmvLoTSGS"
  }else{
    print("plot should be 1, 2 or 3")
    return()
  }

  bt.gmv.rob <- optimize.portfolio.rebalancing(returns, pspec,
                                               optimize_method="CVXR",
                                               rebalance_on="weeks",
                                               training_period=100,
                                               momentFUN=momentEstFun)

  # Extract time series of portfolio weights
  wts.gmv.rob = extractWeights(bt.gmv.rob)
  # Compute cumulative returns of portfolio
  GMV.rob = Return.rebalancing(returns, wts.gmv.rob)

  # Combine GMV.LO and MARKET cumulative returns
  ret.comb<- na.omit(merge(GMV, GMV.rob, MARKET, all=F))
  names(ret.comb) <- c(name, "GmvLo", "MARKET")

  R <- ret.comb
  geometric = TRUE
  c.xts <- if ( geometric ) {
    cumprod(1+R)
  } else {
    1 + cumsum(R)
  }

  p <- xts::plot.xts(c.xts[,1], col="black", main = "Cumulative Returns",
                     grid.ticks.lwd=1, grid.ticks.lty = "dotted",
                     grid.ticks.on = "years",
                     labels.col="grey20", cex.axis=0.8,
                     format.labels = "%b\n%Y",
                     ylim = c(min(c.xts), max(c.xts)), lty="dashed")
  p <- xts::addSeries(c.xts[,2], on=1, lwd=2, col="darkgreen")
  p <- xts::addSeries(c.xts[,3], on=1, lwd=2, col="black", lty="dotted")
  p <- xts::addLegend("topleft", on = 1,
                      legend.names = names(c.xts),
                      lty = c(1,2,3), lwd = rep(2, NCOL(c.xts)),
                      col = c("darkgreen", "black", "black"),
```

```
                      bty = "o", box.col = "white",
                      bg=rgb(t(col2rgb("white")), alpha = 200,
                             maxColorValue = 255) )
  ## Drawdowns panel(Peter Carl)
  d.xts <- PerformanceAnalytics::Drawdowns(R)
  p <- xts::addSeries(d.xts[,1], col="black", lwd=2, main="Drawdown",
                      ylim = c(min(d.xts), 0), lty="dashed")
  p <- xts::addSeries(d.xts[,2], on=2, lwd=2, col="darkgreen")
  p <- xts::addSeries(d.xts[,3], on=2, lwd=2, col="black", lty="dotted")
  # panel 1 and 2 ylim
  ylim1 <- c(p$Env$ylim[[2]][1], p$Env$ylim[[2]][2])
  ylim2 <- c(p$Env$ylim[[4]][1], p$Env$ylim[[4]][2])
  ylim <- c(ylim1, ylim2)
  # get longest drawdown dates for xts object
  dt <- table.Drawdowns(GMV.rob, top = 1) # just want to find the worst drawdown

  if(is.na(dt$To) == TRUE){
    dt$To = index(R)[dim(R)[1]]
  }
  dt2 <- t(dt[,c("From", "To")])
  x <- as.vector(dt2[,NCOL(dt2)])
  y <- as.xts(matrix(rep(ylim, length(x)),ncol=length(ylim), byrow=TRUE), order.by=as.Date(x))
  i=1
  p <- xts::addPolygon(y[i:(i+1),1:2], on=-1, col="lightgrey") # top panel
  p <- xts::addPolygon(y[i:(i+1),3:4], on=-2, col="lightgrey") # lower panel

  return(list(plot = p, ret = R[,2]))
}
```

4.1 CovRob Estimators Backtest

We use default settings of the covRob function for the backtest. From the plot, we can see that during year 2008, when the well-known finance crisis occurred, optimized portfolios with the standard sample covariance matrix estimator and the *covRob* estimators have similar performance. However,the portfolio based on *covRob* outperforms the one based on the standard sample covariance matrix after the crisis in both return and drawdown. This is due to the fact that the *covRob* method rejects influential outliers,, which helps the portfolio optimization avoid being adversely influenced by the outilers that occur during the financial crisis, but not before or after the crisis.

```
bt.gmv <- optimize.portfolio.rebalancing(returns, pspec,
                                         optimize_method="CVXR",
                                         rebalance_on="weeks",
                                         training_period=100)
# Extract time series of portfolio weights
wts.gmv = extractWeights(bt.gmv)
# Compute cumulative returns of portfolio
GMV = Return.rebalancing(returns, wts.gmv)

res.covRob = robPlot(GMV=GMV, MAERKET=MARKET, plot=1)
res.covRob$plot
```



covRob also outperforms conventional portfolio optimization in the worst drawdown. The duration of worst drawdown for **covRob** based portfolio is 73 weeks while the other has a duration of 110 weeks.

```
# longest drawdown for robust based portfolio
table.Drawdowns(GMV, top=1)
```

```
##         From     Trough         To   Depth Length To Trough Recovery
## 1 2008-01-04 2009-03-06 2010-10-08 -0.5529    145        62       83
```

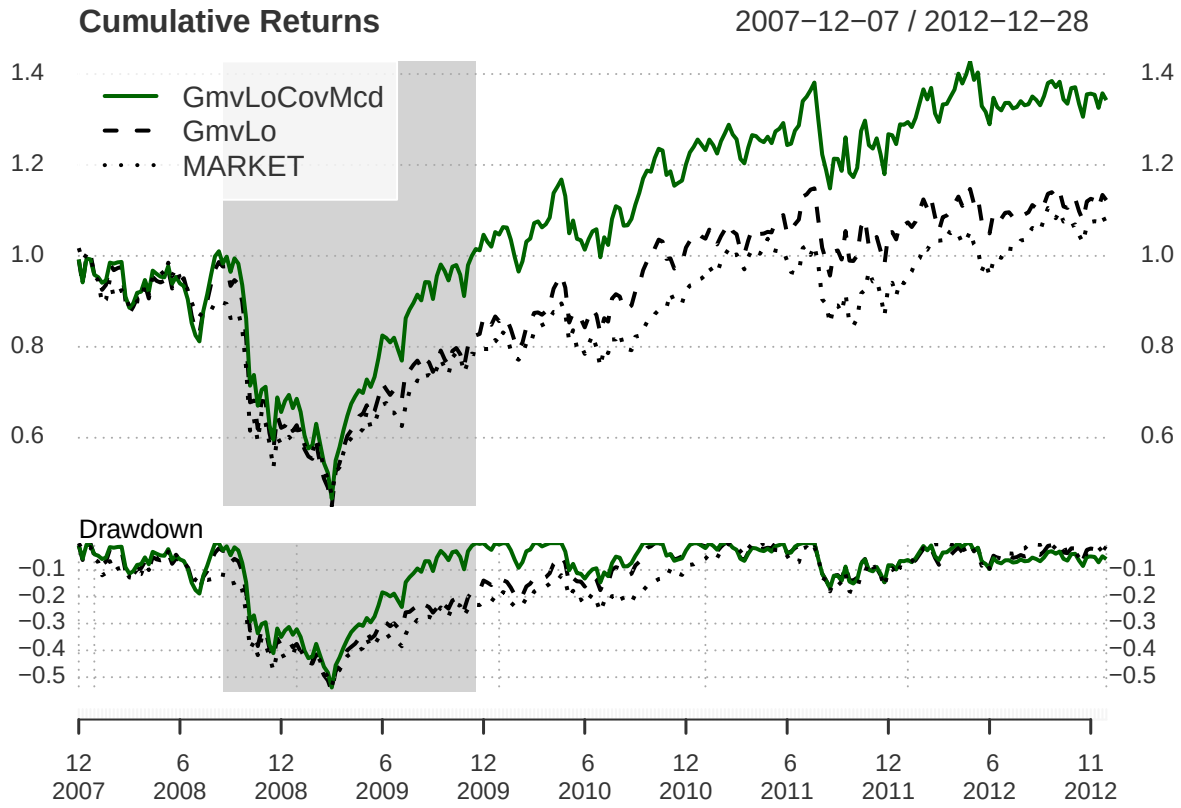
```
# longest drawdown for conventional optimized portfolio
table.Drawdowns(res.covRob$ret, top=1)
```

```
##         From     Trough         To   Depth Length To Trough Recovery
## 1 2008-08-22 2009-03-06 2010-03-05 -0.4979     81        29       52
```

4.2 CovMCD Estimators Backtest

For *covMCD*, we set *alpha* equal to 0.5 during the process of backtest. From the plots below, we can see it outperforms the portfolio with conventional estimators after the crisis even more than *covRob*, which is a really inspiring result.

```
set.seed(1234)
res.covMcd = robPlot(GMV=GMV, MAERKET=MARKET, plot=2)
res.covMcd$plot
```



The worst drawdown for *covMcd* based portfolio has a duration of 71 weeks, better than *covRob* based portfolio and the Markovitz optimized portfolio.

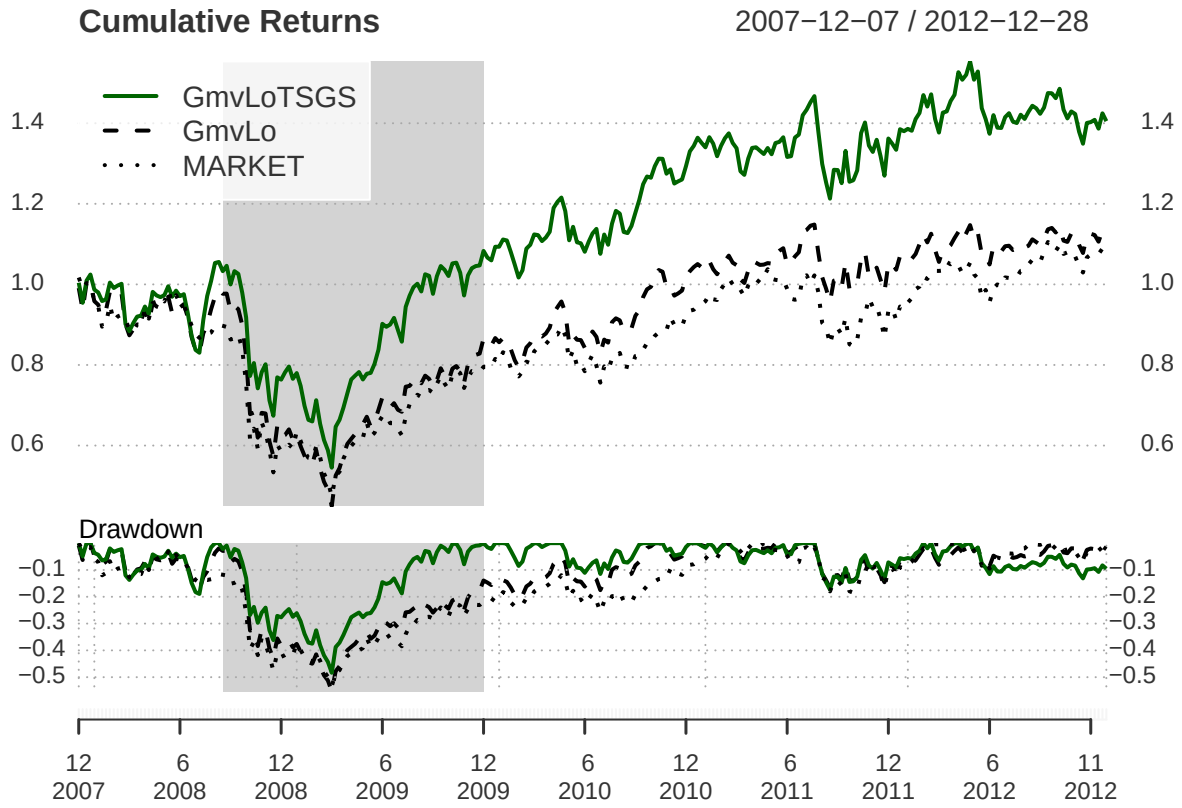
```
# longest drawdown for robust based portfolio
table.Drawdowns(res.covMcd$ret, top=1)
```

```
##         From     Trough         To   Depth Length To Trough Recovery
## 1 2008-08-22 2009-03-06 2009-11-20 -0.5385     66        29       37
```

4.3 TSGS Estimators Backtest

For **TSGS**, we use the default settings. Similiar to *covRob*, optimized portfolios with standard sample covariance matrix and *TSGS* estimators have similar performance before the end of 2008. Furthermore, the *TSGS* based portfolio shows a better cumulative return after year 2008.

```
res.TSGS = robPlot(GMV=GMV, MAERKET=MARKET, plot=3)
res.TSGS$plot
```



The longest drawdown for **TSGS** based portfolio lasts 68 weeks, which is the best among all the robust based portfolios.

```
# longest drawdown for robust based portfolio
table.Drawdowns(res.TSGS$ret, top=1)
```

```
##         From     Trough         To   Depth Length To Trough Recovery
## 1 2008-08-22 2009-03-06 2009-12-04 -0.4837     68        29       39
```

Reference

Agostinelli, Yohai, Claudio. 2015. "Robust Estimation of Multivariate Location and Scatter in the Presence of Cellwise and Casewise Contamination." *Test: An Official Journal of the Spanish Society of Statistics and Operations Research.*

Martin, R. Douglas. 2013. "Robust Covariances: Common Risk Versus Specific Risk Outliers." R-finance Conference 2013.