# CVXR for PortfolioAnalytics

Xinran Zhao and Doug Martin

November 3, 2024

## Contents

# 1 Introduction

CVXR is an R package that provides an object-oriented modeling language for convex optimization, including the Second-Order Cone (SOCP) Optimization required to minimize Coherent Second Moment(CSM) problem, which is not supported by other solvers in PortfolioAnalytics. Hence, CVXR is a significant extension of PortfolioAnalytics.

The purpose of this vignette is to demonstrate examples of optimization problems that can be solved in PortfolioAnalytics with CVXR and its many supported solvers. The problem types covered include not only Linear Programming(LP), Quadratic Programming(QP) but also Second-Order Cone Programming(SOCP). Multiple solvers supported by CVXR can be selected according to optimization types. For example, SCS and ECOS can completely cover the types of problems that ROI can deal with, such as mean-variance and ES problem. In order to better understand the functionality and use of PortfolioAnalytics, users are recommended to read the Vignette *Introduction to PortfolioAnalytics* first.

The R code `demo_cvxrPortfolioAnalytics.R` in the PortfolioAnalytics `demo` folder, reproduces the results in this Vignette.

# 2 Getting Started

## 2.1 Load Packages

Load the necessary packages.

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

## 2.2 Solvers

The website https://cvxr.rbind.io/ shows that CVXR currently supports the use of 9 solvers, some of which are commercial (CBC, CPLEX, GUROBI, MOSEK)[1] and the others are open source(GLPK, GLPK_MI, OSQP, SCS, ECOS).

The PortfolioAnalytics package provides the following two main wrapper functions for constrained optimization of portfolios using a wide variety of methods:

```
optimize.portfolio(R =, portfolio =, optimize_method =)
```

```
optimize.portfolio.rebalancing(R =, portfolio =, optimize_method =, rebalance_on =,
training_period =, rolling_window =)
```

Different solvers support different types of portfolio optimization problems, which should be specified by the argument `optimize_method`. The `optimize_method=c("CVXR", {CVXRsolver})` argument of the function `optimize.portfolio` and `optimize.portfolio.rebalancing` allows the user to specify the solver to use with CVXR. If the argument is `optimize_method="CVXR"`, the default solver for QP type portfolio optimization problems, such as minimum variance portfolio optimization is OSQP, and the default solver for LP and SOCP type portfolio optimizations, such as maximum mean return, "robust portfolio optimization" to control for alpha uncertainty, and Coherent Second Moment (CSM) portfolio optimization, is SCS.

[1] MOSEK ,GUROBI, and CPLEX require licenses to use, but free academic licenses are available for all three.

| Solver | LP | QP | SOCP |
|---|---|---|---|
| CBC | ✓ | | |
| GLPK | ✓ | | |
| GLPK_MI | ✓ | | |
| OSQP | ✓ | ✓ | |
| SCS | ✓ | ✓ | ✓ |
| ECOS | ✓ | ✓ | ✓ |
| CPLEX | ✓ | ✓ | ✓ |
| GUROBI | ✓ | ✓ | ✓ |
| MOSEK | ✓ | ✓ | ✓ |

## 2.3 Data

The examples in this Vignette make use of the following three data sets in the Sections listed for each:

1. `ret_edhec` For the remainder of this Section 2, through Section 8.
2. `retD_CRSP` In Sections 9.1 and 9.2
3. `retM_CRSP_5` In Sections 9.3 and 9.4

The `ret_edhecc` data set consists of the monthly returns of hedge fund style indexes from January 2011 to December 2015, extracted from the the longer `edhec` returns data set in the `PerformanceAnalytics` package, as described just below.

The `retD_CRSP` data set contains daily returns of 30 smallcap stocks from 1993 to 2015, extracted from the `stocksCRSP` data set in the `PCRA` package, as described at the beginning of Section 9.

The `retM_CRSP_5` data set contains monthly returns from January 2011 to December 2015, extracted from the above `retD_CRSP` daily returns, as described at the beginning of Section 9.3.

The `ret_edhec` data set is created by the code chunk below.

```
data(edhec)
class(edhec)
#> [1] "xts" "zoo"
ret_edhec <- tail(edhec, 60) # Extract the last 5 years
range(index(edhec)) # Start and end dates of `edhec`
#> [1] "1997-01-31" "2021-05-31"
range(index(ret_edhec)) # Start and end dates of ret_edhec
#> [1] "2016-06-30" "2021-05-31"
# names(edhec) # Names of `edhec` long, so use shorter names
colnames(ret_edhec) <- c("CA", "CTAG", "DS", "EM", "EMN", "ED",
                         "FIA", "GM", "LSE", "MA", "RV", "SS", "FF")
print(head(ret_edhec, 5))
#>                 CA    CTAG     DS     EM     EMN      ED     FIA      GM     LSE      MA     RV      
#> 2016-06-30 0.0016  0.0352 0.0082 0.0149 -0.0037 -0.0012 -0.0021  0.0107 -0.0093 -0.0018 0.0027  0.01
#> 2016-07-31 0.0154  0.0067 0.0192 0.0233  0.0080  0.0186  0.0114  0.0064  0.0203  0.0070 0.0125 -0.05
#> 2016-08-31 0.0102 -0.0225 0.0209 0.0169 -0.0019  0.0149  0.0076 -0.0061  0.0062  0.0082 0.0064 -0.03
#> 2016-09-30 0.0070 -0.0066 0.0097 0.0097  0.0040  0.0039  0.0062 -0.0028  0.0064  0.0041 0.0056 -0.01
#> 2016-10-31 0.0027 -0.0257 0.0203 0.0064  0.0024 -0.0015  0.0063  0.0014 -0.0072 -0.0062 0.0052  0.02
```

We see from `class(edhec)` that edhec is an `xts` type time series object, and we see from the two lines above that use the `range` function that, whereas `edhec` starts in December 1997, `ret_edhec1` does not start until December 2014. But both `edhec` and `ret_edhec` end in November 2019. (The function `index` extracts the dates of an xts object, and `range` extracts the first and last dates of a `Dates` object).

`tsPlotMP` is a function in the R package `PCRA` that is convenient for plotting `xts` class time series objects.

```
tsPlotMP(ret_edhec, layout = c(2, 7))
```

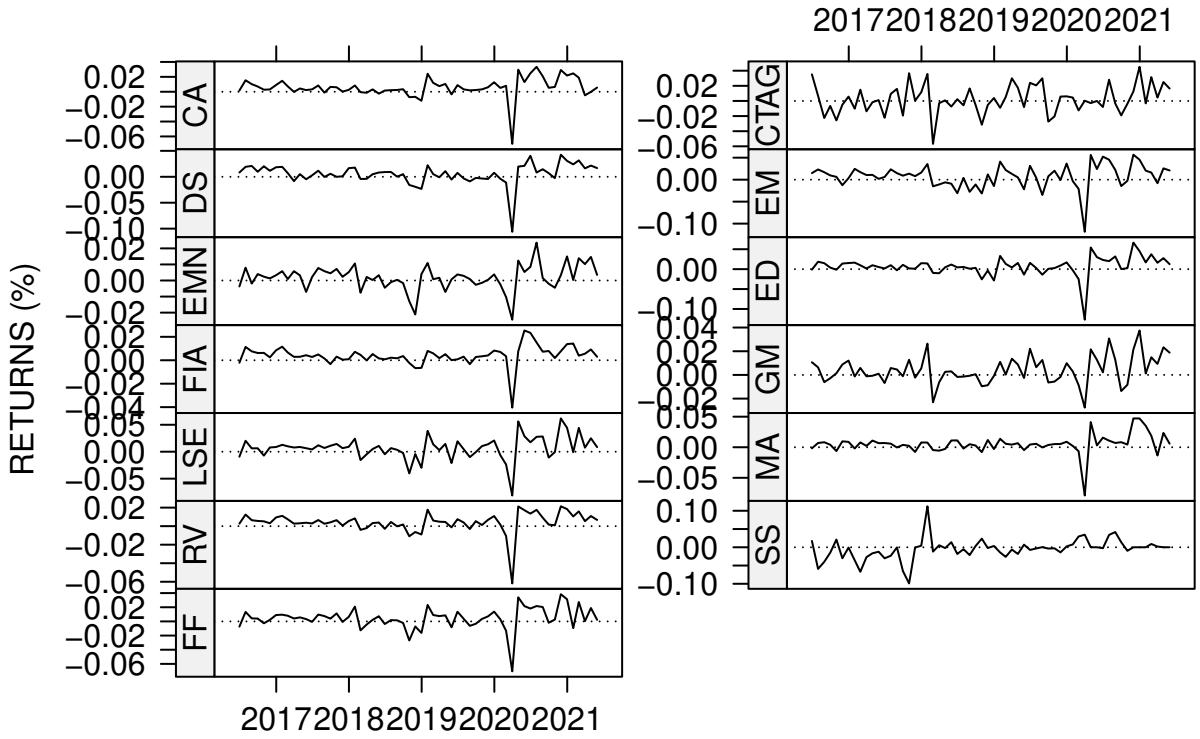


Fig 2.1 EDHEC hedge fund style indexes returns from November 2014 to November 2019.

### 2.4 Optimization Problems

In this Vignette, all mean vectors and covariance matrices in the optimization formula will use standard sample based estimates. All optimization problems treated will use linear constraints unless stated otherwise. There will be one equality constraint, i.e., the full-investment constraint, and one or more inequality constraints such as the long-only and box constraints. More comprehensive constraint types can be found in Ross Bennett (2018) *Introduction to PortfolioAnalytics*.

## 3 Maximizing Mean Return

The objective to maximize mean return is a linear problem of the form:

$$\max_{w} \quad \boldsymbol{\mu}'\boldsymbol{w}$$

$$s.t. \quad A\boldsymbol{w} \geq b$$
$$B\boldsymbol{w} = c$$

Where $\boldsymbol{\mu}$ is the estimated asset returns mean vector and $\boldsymbol{w}$ is the vector of portfolio weights.

### 3.1 Portfolio Object

The first step in setting up a model is to create the portfolio object, which contains the form of the constraints, and the objective specifications. In the following we create full-investment and box constraints specifications, and a maximum return objective specification.

```
# Create portfolio object
fund_edhec <- colnames(ret_edhec)
pspec_maxret <- portfolio.spec(assets = fund_edhec)
# Add constraints to the portfolio object
pspec_maxret <- add.constraint(pspec_maxret, type = "full_investment")
pspec_maxret <- add.constraint(portfolio = pspec_maxret, type = "box",
                               min = rep(0.02, 13),
                               max = c(rep(0.15, 8), rep(0.1, 5)))
# Add objective to the portfolio object
pspec_maxret <- add.objective(portfolio = pspec_maxret,
                              type = "return", name = "mean")
pspec_maxret
#> ***********************************************
#> PortfolioAnalytics Portfolio Specification
#> ***********************************************
#>
#> Call:
#> portfolio.spec(assets = fund_edhec)
#>
#> Number of assets: 13
#> Asset Names
#>  [1] "CA"   "CTAG" "DS"   "EM"   "EMN"  "ED"   "FIA"  "GM"   "LSE"  "MA"
#> More than 10 assets, only printing the first 10
#>
#> Constraints
#> Enabled constraint types
#>      - full_investment
#>      - box
#>
#> Objectives:
#> Enabled objective names
#>      - mean
```

### 3.2 Optimization

The next step is to run the optimization. Note that `optimize_method = c("CVXR", {CVXRsolver})` should be specified in the function `optimize.portfolio` to use CVXR solvers for the optimization, or use the default solver by giving `optimize_method = "CVXR"`. For maximizing mean return, which is a linear programming optimization solver, the default solver is `OSQP`.

```
# Run the optimization with default solver
opt_maxret <- optimize.portfolio(R = ret_edhec, portfolio = pspec_maxret, optimize_method = "CVXR")
opt_maxret
#> ***********************************
#> PortfolioAnalytics Optimization
#> ***********************************
#>
#> Call:
#> optimize.portfolio(R = ret_edhec, portfolio = pspec_maxret, optimize_method = "CVXR")
#>
#> Optimal Weights:
#>     CA   CTAG     DS     EM    EMN     ED    FIA     GM    LSE     MA     RV     SS     FF
#> 0.1501 0.0201 0.1501 0.1501 0.0201 0.1501 0.0201 0.0201 0.1001 0.1001 0.0201 0.0201 0.0788
```

```
#>
#> Objective Measures:
#>     mean
#> 0.005805
opt_maxret$solver
#> [1] "SCS"

# Run the optimization with a different solver
opt_maxret_glpk <- optimize.portfolio(R = ret_edhec, portfolio = pspec_maxret,
                                      optimize_method = c("CVXR", "GLPK"))
opt_maxret_glpk$solver
#> [1] "GLPK"
```

You can find the class of the object opt_maxret, and the names of its components as follows:

```
class(opt_maxret)
#> [1] "optimize.portfolio.CVXR" "optimize.portfolio"
```

```
names(opt_maxret)
#>  [1] "weights"            "objective_measures" "opt_values"         "out"                "call"
#> [11] "end_t"
```

If you just want to see the values of optimal weights, use:

```
opt_maxret$weights
#>         CA       CTAG         DS         EM        EMN         ED        FIA         GM        LSE
#> 0.15009683 0.02009675 0.15009683 0.15009685 0.02009675 0.15009684 0.02009677 0.02009603 0.10009682 0
```

However, if in addition to finding out what the optimal weights are, you also want to know the optimal portfolio's mean return, standard deviation, and Sharpe ratio, use the convenience function `opt.outputMvo`, whose second argument is the assets returns time series, in this case `ret_edhec`:

```
opt.outputMvo(opt_maxret, ret_edhec, digits =3)
#> $Wgts
#>    CA  CTAG    DS    EM   EMN    ED   FIA    GM   LSE    MA    RV    SS    FF
#> 0.150 0.020 0.150 0.150 0.020 0.150 0.020 0.020 0.100 0.100 0.020 0.020 0.079
#>
#> $Mean
#> [1] 0.07
#>
#> $StdDev
#> [1] 0.06
#>
#> $SR
#> [1] 1.157
```

### 3.3 Backtesting

Out of sample back-testing based that consists of computing portfolio cumulative gross returns (CGR), can be done using the function `optimize.portfolio.rebalancing`. In this regard there are two distinctly different ways of doing the computation with regard to use of the asset returns histories:

1. Growing data window method
2. Moving data window method.

For the growing data window method, the portfolio manager specifies an initial `training_period` numeric value, and a `rebalance_on` character value. The `training_period` value is the number of periods from

inception of the returns data to be used to compute an initial optimal portfolio. For example in the case of a portfolio of assets with monthly returns: (1) use of `training_period = 36` specifies that the first 36 months, i.e., the first three years, of returns are used for the initial portfolio optimization, and (2) use of `rebalance_on = "quarters` specifies that the portfolio is rebalanced every quarter, i.e., every 3 months, starting the month at the end of the training period.

For the moving data window method, the portfolio manager specifies a `moving_window` value and a `rebalance_on` value. The first of these specifies the number of periods of asset returns to use for each portfolio optimization, and second has the same meaning as for the above growing window method. So for a portfolio of daily returns, `moving_window = 500` means that approximately two years of daily returns are used for each portfolio optimization, with the first window position starting at the inception of the data, and `rebalance_on = monthly` specifies that the moving window will increase its position by one month for each optimization.

The following `ret_edhec` data code example uses the growing window method, and Sections 9.1 and 9.2 use the moving window method.

```
bt_maxret <- optimize.portfolio.rebalancing(R = ret_edhec, portfolio = pspec_maxret,
                                            optimize_method = "CVXR",
                                            rebalance_on = "quarters", training_period = 36)
```

The class of the `bt_maxret` object is `optimize.portfolio.rebalance`, which is essentially a list object with the following components names:

```
names(bt_maxret)
#> [1] "portfolio"       "R"               "call"            "elapsed_time"    "opt_rebalancing"
```

The most important is `opt_rebalancing`, which itself is a list with

```
names(bt_maxret$opt_rebalancing)
#> [1] "2019-06-30" "2019-09-30" "2019-12-31" "2020-03-31" "2020-06-30" "2020-09-30" "2020-12-31" "2021
```

We see that the names of `opt_rebalancing` are the dates on which the portfolio is rebalanced, and for each date the contents of `opt_rebalancing` are the same as a single portfolio optimization in Section 3.2. Note that the first rebalance date 2017-12-31 is month 60 after the initial `ret_edhec` date of 2014-12-31. Then each subsequent rebalance date is 3 months later, except the last date 2019-11-30 is only 2 months later because it is the final date.

# 4 Minimizing Variance

The objective to minimize variance is a quadratic problem of the form:

$$\min_{w} \quad \boldsymbol{w}'\Sigma\boldsymbol{w}$$

subject to portfolio managers' desired constraints, where $\Sigma$ is the estimated covariance matrix of asset returns and $\boldsymbol{w}$ is the vector of portfolio weights. This is a quadratic optimization problem.

## 4.1 Global Minimum Variance Portfolio

### 4.1.1 Portfolio Object

In this example, the only constraint specified is the full investment constraint, and we will compute the weights of a global minimum variance (GMV) portfolio.

```
# Create portfolio object
pspec_gmv <- portfolio.spec(assets = fund_edhec)
# Add full-investment constraint
pspec_gmv <- add.constraint(pspec_gmv, type = "full_investment")
```

```
# Add objective of minimizing variance
pspec_gmv <- add.objective(portfolio = pspec_gmv, type = "risk", name = "var")
```

#### 4.1.2 Optimization

```
opt_gmv <- optimize.portfolio(ret_edhec, pspec_gmv, optimize_method = "CVXR")
opt.outputMvo(opt_gmv, ret_edhec, digits =3)
#> $Wgts
#>     CA   CTAG     DS     EM    EMN     ED    FIA     GM    LSE     MA     RV     SS     FF
#> -0.146 -0.034 -0.052 -0.161  0.004 -0.640  0.358  0.099  0.322  0.242  1.121  0.032 -0.146
#>
#> $Mean
#> [1] 0.029
#>
#> $StdDev
#> [1] 0.013
#>
#> $SR
#> [1] 2.224
```

As this example illustrates, a global minimum variance portfolio with only a full-investment constraint can have short positions.

### 4.2 Long-Only and Group Constrained Minimum Variance Portfolio

Various linear inequality constraint, such as box constraints, group constraints and a target mean return constraint, can be used with GMV portfolio construction. Here we demonstrate the case of a linearly constrained groups weights minimum variance portfolio, where the constraints are on the sum of the weights in each group.

```
# portfolio object
pspec_gmv <- add.constraint(pspec_gmv, type = "long_only")
pspec_gmvGroup <- add.constraint(pspec_gmv, type = "group",
                            groups = list(groupA=1,
                                          groupB=c(2:12),
                                          groupC=13),
                            group_min = c(0, 0.05, 0.05),
                            group_max = c(0.4, 0.8, 0.5))
pspec_gmvGroup <- add.constraint(pspec_gmvGroup, type = "return", return_target = 0.003)
pspec_gmvGroup
#> ***************************************************
#> PortfolioAnalytics Portfolio Specification
#> ***************************************************
#>
#> Call:
#> portfolio.spec(assets = fund_edhec)
#>
#> Number of assets: 13
#> Asset Names
#>  [1] "CA"   "CTAG" "DS"   "EM"   "EMN"  "ED"   "FIA"  "GM"   "LSE"  "MA"
#> More than 10 assets, only printing the first 10
#>
#> Constraints
#> Enabled constraint types
```

```
#>       - full_investment
#>       - long_only
#>       - group
#>       - return
#>
#> Objectives:
#> Enabled objective names
#>       - var

# optimization
opt_gmvGroup <- optimize.portfolio(ret_edhec, pspec_gmvGroup,    optimize_method = "CVXR")
opt.outputMvo(opt_gmvGroup, ret_edhec, digits =3)
#> $Wgts
#>    CA  CTAG    DS    EM   EMN    ED   FIA    GM   LSE    MA    RV    SS    FF
#> 0.150 0.039 0.000 0.000 0.327 0.000 0.368 0.000 0.000 0.000 0.000 0.066 0.050
#>
#> $Mean
#> [1] 0.036
#>
#> $StdDev
#> [1] 0.027
#>
#> $SR
#> [1] 1.333
```

The optimal weights show that the first group constraint is not binding, but the second one is binding with FIA plus MA at the upper bound of 0.8, and the third group constraint is binding at the lower bound with FF.

The use of an alternative to the CVXR default `OSQP` solver will typically result in the same minimum variance weights to many significant digits. For example, if we use `optimize_method = c("CVXR", "ECOS")`, we result in the same optimal weights to 4 significant digits.

```
opt_gmvGroup_ecos <- optimize.portfolio(ret_edhec, pspec_gmvGroup, optimize_method = c("CVXR", "ECOS"))
opt.outputMvo(opt_gmvGroup_ecos, ret_edhec, digits =3)
#> $Wgts
#>    CA  CTAG    DS    EM   EMN    ED   FIA    GM   LSE    MA    RV    SS    FF
#> 0.150 0.039 0.000 0.000 0.327 0.000 0.368 0.000 0.000 0.000 0.000 0.066 0.050
#>
#> $Mean
#> [1] 0.036
#>
#> $StdDev
#> [1] 0.027
#>
#> $SR
#> [1] 1.333

opt_gmvGroup$solver
#> [1] "OSQP"
opt_gmvGroup_ecos$solver
#> [1] "ECOS"
```

# 5 Maximizing Quadratic Utility

Here we will compute a maximum quadratic utility portfolio, which is of course a minimum variance portfolios. The quadratic utility function is $\text{QU}(\boldsymbol{w}) = \mu_\text{p} - \lambda\sigma_\text{p}^2 = \boldsymbol{\mu}'\boldsymbol{w} - \lambda\boldsymbol{w}'\Sigma\boldsymbol{w}$:

$$\max_w \quad \boldsymbol{\mu}'\boldsymbol{w} - \lambda\boldsymbol{w}'\Sigma\boldsymbol{w}$$

$$s.t. \quad A\boldsymbol{w} \geq b$$

Here $\boldsymbol{\mu}$ is the vector of estimated mean asset returns, $0 \leq \lambda < \text{inf}$ is the risk aversion parameter, $\Sigma$ is the estimated covariance matrix of asset returns, and $\boldsymbol{w}$ is the vector of weights. Quadratic utility maximizes return while penalizing variance risk. The risk aversion parameter $\lambda$ controls how much portfolio variance is penalized, and when $\lambda = 0$ it becomes a maximum mean return problem of Section 3, and as $\lambda \to \text{inf}$, it becomes the global minimum variance problem of Section 4.

## 5.1 Portfolio Object

The logic of PortfolioAnalytics is such that when objectives of both return and risk are specified, the portfolio will be a maximum quadratic utility portfolio, and this is reflected in the quadratic utility specification object `pspec_qu` below, where we use a risk aversion parameter value $\lambda = 20$ by setting `risk_aversion = 0`.

```
pspec_qu <- portfolio.spec(assets = fund_edhec)
pspec_qu <- add.constraint(pspec_qu, type = "full_investment")
pspec_qu <- add.constraint(pspec_qu, type = "long_only")
# Add objectives
pspec_qu <- add.objective(portfolio = pspec_qu, type = "return", name = "mean")
pspec_qu <- add.objective(portfolio = pspec_qu, type = "risk", name = "var",
                          risk_aversion = 20)
```

## 5.2 Optimization

Use of `optimize.portfolio` with `optimize_method = "CVXR"` and the above `pspec_qu` gives the following result.

```
opt_qu <- optimize.portfolio(ret_edhec, pspec_qu, optimize_method = "CVXR")
opt.outputMvo(opt_qu, ret_edhec, digits =3)
#> $Wgts
#>    CA  CTAG    DS    EM   EMN    ED   FIA    GM   LSE    MA    RV    SS    FF
#> 0.195 0.000 0.000 0.000 0.000 0.000 0.639 0.166 0.000 0.000 0.000 0.000 0.000
#>
#> $Mean
#> [1] 0.054
#>
#> $StdDev
#> [1] 0.031
#>
#> $SR
#> [1] 1.756
```

# 6 Minimizing Expected Shortfall

The Expected Shortfall (ES) of a portfolio P with return $r_P$ is the expected value of the portfolio return conditioned on the return being larger than the *value-at-risk* (VaR) of the portfolio. Taking loss as a positive quantity by using $-r_P$, the ES of a portfolio is

$$\begin{aligned} ES_\gamma(r_P) = ES_\gamma(\boldsymbol{w}) &= -E(r_P | r_P \leq q_\gamma(\boldsymbol{w})) \\ &= -E(\boldsymbol{w}'\boldsymbol{r} | \boldsymbol{w}'\boldsymbol{r} \leq q_\gamma(\boldsymbol{w})) \end{aligned}$$

where $q_\gamma$ is $\gamma$-quantile, with $\gamma$ an upper tail probability of the distribution of $-r_P$. The negative of $q_\gamma$ is the VaR of $-r_P$. Often the $\gamma$ values 0.01 or 0.05 are used, in which cases ES is referred to as a "tail risk" measure. But one could also choose $\gamma = 0.25$ or $\gamma = 0.5$, in which case ES is just a "downside risk" measure. If a user specifies a value $\gamma > 0.5$, then they are specifying the "confidence level" of the VaR, which is typically 0.95 or 0.99, and in such cases PortfolioAnalytics will take $1 - \gamma$ as the tail probability.

It was shown by Rockafellar, Uryasev, et al. (2000), who used the term *conditional value-at-risk* (CVaR) for what is now called ES, that a minimum ES (MES) portfolio is the result of the minimization:

$$\min_{\boldsymbol{w}} ES_\gamma(\boldsymbol{w}) = \min_{\boldsymbol{w},t} F_\gamma(\boldsymbol{w}, t)$$

where

$$F_\gamma(\boldsymbol{w}, t) = -t + \frac{1}{\gamma} \int [t - \boldsymbol{w'r}]^+ \cdot f(\boldsymbol{r}) d\boldsymbol{r}$$

.

Since the above optimization does not specify a portfolio mean return constraint, it is often referred to as a *global minimum expected shortfall* (GMES) portfolio.

Assuming that $n\gamma$ is an integer, the empirical data-based version of the above formula is

$$\hat{F}_\gamma(\boldsymbol{w}, t) = -t + \frac{1}{n\gamma} \sum_{i=1}^{n} [t - \boldsymbol{w'r_i}]^+$$

and when $n\gamma$, it is replaced by the smallest integer greater than $n\gamma$ otherwise. The positive part function, $[t - \boldsymbol{w'r_i}]^+$, can easily be converted to a collection of linear constraints, hence, the minimization of ES is equivalent to solving a linear programming problem.

NOTE: PortfolioAnalytics uses $p$ in place of $\gamma$, as the the example below.

## 6.1 GMES Portfolio Specification Object

The PortfolioAnalytics default tail probability for MES portfolios is 0.05, but the user may specify alternative probabilities, as in the `pspec_es_1` code line below.

```
pspec_es <- portfolio.spec(assets = fund_edhec)
pspec_es <- add.constraint(pspec_es, type = "full_investment")
pspec_es <- add.constraint(pspec_es, type = "long_only")
# Add objective of minimizing ES by using the default gamma, which means tail probability 0.05
pspec_gmes <- add.objective(portfolio = pspec_es, type = "risk", name = "ES")
# Add objective of minimizing ES by using the specific gamma=0.1
pspec_gmes_1 <- add.objective(portfolio = pspec_es, type = "risk", name = "ES",
                              arguments = list(p=0.1))
```

## 6.2 GMES Optimization

We show below the computation of long-only GMES portfolios for tail probabilities 0.05 and 0.10.

```
# GMES with default gamma=0.05
opt_gmes <- optimize.portfolio(ret_edhec, pspec_gmes, optimize_method = "CVXR")
opt_gmes
#> **********************************
#> PortfolioAnalytics Optimization
#> **********************************
#>
#> Call:
#> optimize.portfolio(R = ret_edhec, portfolio = pspec_gmes, optimize_method = "CVXR")
#>
```

```
#> Optimal Weights:
#>     CA   CTAG     DS     EM    EMN     ED    FIA     GM    LSE     MA     RV     SS     FF
#> 0.0000 0.0444 0.0000 0.0000 0.5111 0.0000 0.0680 0.1689 0.0000 0.0000 0.0000 0.2076 0.0000
#>
#> Objective Measures:
#>      ES
#> 0.01281
# GMES with specific gamma=0.1
opt_gmes_1 <- optimize.portfolio(ret_edhec, pspec_gmes_1, optimize_method = "CVXR")
opt_gmes_1
#> ***********************************
#> PortfolioAnalytics Optimization
#> ***********************************
#>
#> Call:
#> optimize.portfolio(R = ret_edhec, portfolio = pspec_gmes_1, optimize_method = "CVXR")
#>
#> Optimal Weights:
#>     CA   CTAG     DS     EM    EMN     ED    FIA     GM    LSE     MA     RV     SS     FF
#> 0.0000 0.0000 0.0000 0.0000 0.3202 0.0000 0.5524 0.0556 0.0000 0.0000 0.0000 0.0718 0.0000
#>
#> Objective Measures:
#>       ES
#> 0.009039
```

It is interesting to see that both of the above two GMES portfolios have the same non-zero weights, but the those non-zero weights are somewhat different for the two portfolios.

NOTE: Since PortfolioAnalytics does not yet have an MES type replacement for opt.outputMvo, we just printed the objects `opt_gmes` and `opt_gmes_1`.

# 7 Minimizing Portfolio Coherent Second Moment

Here we describe the PortfolioAnalytics capability to compute portfolios that have a *minimum coherent second moment* (MCSM) risk. The coherent second moment (CSM) risk measure is a special case of higher moment coherent risk measures introduced by Krokhmal (2007).

A MCSM tail probability $\gamma$ portfolio is a solution to the data-based version of the optimization problem

$$\min_{\boldsymbol{w},t} \quad -t + \gamma^{-1}||(t - \boldsymbol{w'r})^{+}||_2$$

where $||X^{+}||_2$ is the square root of the expected value of the square of the random variable $X^{+}$. The data-based version is obtained by replacing the expected value with the sample average across portfolio returns $\boldsymbol{w'r_i}$.

The MCSM optimization problem is a convex problem with a second-order cone constraint, which is referred to as a *second-order cone programming* (SOCP) problem. PortfolioAnalytics solves this problem as follows, by using CVXR with with the SCS solver tailored to this particular SOCP problem.

## 7.1 Portfolio Specification Object

An MCSM portfolio for the ret_edhec data, without a mean return constraint is based on the following mcsm specification object:

```
pspec_csm <- portfolio.spec(assets = fund_edhec)
pspec_csm <- add.constraint(pspec_csm, type = "full_investment")
pspec_csm <- add.constraint(pspec_csm, type = "long_only")
# Add objective of minimizing CSM
pspec_mcsm <- add.objective(portfolio = pspec_csm, type = "risk", name = "CSM",
                            arguments = list(p=0.05))
```

### 7.2 Optimization

```
opt_mcsm <- optimize.portfolio(ret_edhec, pspec_mcsm, optimize_method = "CVXR")
opt_mcsm
#> **********************************
#> PortfolioAnalytics Optimization
#> **********************************
#>
#> Call:
#> optimize.portfolio(R = ret_edhec, portfolio = pspec_mcsm, optimize_method = "CVXR")
#>
#> Optimal Weights:
#>     CA   CTAG     DS     EM    EMN     ED    FIA     GM    LSE     MA     RV     SS     FF
#> 0.0000 0.0250 0.0000 0.0000 0.4609 0.0000 0.0246 0.2798 0.0000 0.0000 0.0000 0.2097 0.0000
#>
#> Objective Measures:
#>     CSM
#> 0.01298
```

It is interesting to see that the above tail probability 5% MCSM portfolio has the same non-zero weights as the tail probability 5% MES portfolio, but the non-zero weights are noticeably different: to 3 digits the 5% MES weights for (EMN, FIA, MA, SS) are (0.089, 0.418, 0.435, 0.058) while those for the 5% MCSM are (0.197, 0.514, 0.226, 0.063), e.g., the EMN weight has more than doubled, and the MA weight has been reduced by almost one-half.

NOTE: Since PortfolioAnalytics does not yet have an MCSM type replacement for opt.outputMvo, we just printed the objects `opt_mcsm`.

## 8 Maximizing Mean Return Per Unit Risk

There are three basic types of risk measures: variance or standard deviation, ES and CSM. The problem of maximizing mean return per unit risk can be solved in a clever way by minimizing risk with a target return constraint, as is described below. For all three of these types of problems, both return and risk objectives should be used in PortfolioAnalytics. Then for each of these three optimization problems an appropriate argument needs to be given to the `optimize.portfolio` to specify the type of problem, as we describe below.

### 8.1 Maximum Sharpe Ratio Portfolios

The Sharpe Ratio of a random return $r_P$ of a portfolio $P$ is defined as:

$$\frac{E(r_P) - r_f}{\sqrt{Var(r_P)}}.$$

where $Var(r_P)$ is a quadratic form in the portfolio weights and the asset returns covariance matrix. As in the general case of minimizing portfolio variance subject to linear equality and inequality constraints, a portfolio manager needs to maximize the Sharpe ratio subject to such constraint.

While maximization of the Sharpe ratio subject to such constraints, there is a well-know method of converting it to a quadratic programming, which is for example described in Section 6.4 of Cornuejols, Pena, and Tutuncu (2018) for the case of general linear equality and inequality constraints. For the case of fully-invested long-only portfolios, their method is to solve the optimization problem

$$\begin{aligned} \underset{w}{minimize} \quad & w'\Sigma w \\ s.t. \quad & (\hat{\mu} - r_f\mathbf{1})^T w = 1 \\ & \mathbf{1}^T w = \kappa \\ & \kappa > 0 \end{aligned}$$

thereby obtaining a solution$(w^*, \kappa^*)$ with $k^* > 0$, and computing the maximized Sharpe ratio as $\tilde{w}^* = w^*/\kappa^*$.

When creating the portfolio, the argument `maxSR = TRUE` should be specified in the function `optimize.portfolio` to distinguish the quadratic utility maximization problem. NOTE: When both mean and var/StdDev objectives are included in the portfolio specification object, `optimize.portfolio` maximizes quadratic utility.

```
# Create portfolio object
pspec_sr <- portfolio.spec(assets = fund_edhec)
## Add constraints of maximizing Sharpe Ratio
pspec_sr <- add.constraint(pspec_sr, type = "full_investment")
pspec_sr <- add.constraint(pspec_sr, type = "long_only")
## Add objectives of maximizing Sharpe Ratio
pspec_sr <- add.objective(pspec_sr, type = "return", name = "mean")
pspec_sr <- add.objective(pspec_sr, type = "risk", name = "var")

# Optimization
optimize.portfolio(ret_edhec, pspec_sr, optimize_method = "CVXR", maxSR = TRUE)
#> **********************************
#> PortfolioAnalytics Optimization
#> **********************************
#>
#> Call:
#> optimize.portfolio(R = ret_edhec, portfolio = pspec_sr, optimize_method = "CVXR",
#>     maxSR = TRUE)
#>
#> Optimal Weights:
#>     CA   CTAG     DS     EM    EMN     ED    FIA     GM    LSE     MA     RV     SS     FF
#> 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.8046 0.1954 0.0000 0.0000 0.0000 0.0000 0.0000
#>
#> Objective Measures:
#>     mean
#> 0.004139
#>
#>
#>   StdDev
#> 0.007957
#>
#>
#> Sharpe Ratio
#>       0.5202
```

## 8.2 Maximum ES ratio Portfolios

The ES ratio(ESratio), which is also called STARR in PortfolioAnalytics, is defined as:

$$\frac{E(r_P) - r_f}{ES_\gamma(r_P)}$$

Similar to constructing a maximum Sharpe Ratio, the problem of maximizing the ES ratio can be formulated as a constrained ES minimization problem, which has been implemented in `optimize.portfolio`.

When creating the portfolio specification object for maximizing the ES ratio, both the return and ES objectives must be specified, and `ESratio = TRUE` is the default. If `ESratio = FALSE` is used, the action will be to minimize ES, ignoring the return objective. We note that the argument `ESratio = TRUE` is equivalent to `maxSTARR = TRUE`, which is used in other vignettes.

```
# Create portfolio object
pspec_ESratio <- portfolio.spec(assets = fund_edhec)
## Add constraints of maximizing return per unit ES
pspec_ESratio <- add.constraint(pspec_ESratio, type = "full_investment")
pspec_ESratio <- add.constraint(pspec_ESratio, type = "long_only")
## Add objectives of maximizing return per unit ES
pspec_ESratio <- add.objective(pspec_ESratio, type = "return", name = "mean")
pspec_ESratio <- add.objective(pspec_ESratio, type = "risk", name = "ES", arguments = list(p=0.05))

# Optimization
optimize.portfolio(ret_edhec, pspec_ESratio, optimize_method = "CVXR", ESratio = TRUE)
#> **********************************
#> PortfolioAnalytics Optimization
#> **********************************
#>
#> Call:
#> optimize.portfolio(R = ret_edhec, portfolio = pspec_ESratio,
#>     optimize_method = "CVXR", ESratio = TRUE)
#>
#> Optimal Weights:
#>     CA   CTAG     DS     EM    EMN     ED    FIA     GM    LSE     MA     RV     SS     FF
#> 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.6713 0.3287 0.0000 0.0000 0.0000 0.0000 0.0000
#>
#> Objective Measures:
#>     mean
#> 0.004152
#>
#>
#>      ES
#> 0.01604
#>
#>
#> ES ratio
#>   0.2588
```

## 8.3 Maximum CSM ratio Portfolios

The CSM ratio of a random return $r_P$ of a portfolio $P$ is defined as:

$$\frac{E(r_P) - r_f}{CSM_\gamma(r_P)}$$

Similar to maximizing Sharpe Ratio, the problem of maximizing a portfolio's CSM ratio can be formulated as a CSM minimization problem.

When creating the portfolio, both return and CSM objectives must be provided. The argument `CSMratio =` is used to specify this optimization problem, and the default value is `CSMratio = TRUE`. If `CSMratio = FALSE`, the action will be to minimize CSM, ignoring the return objective.

```
# Create portfolio object
pspec_CSMratio <- portfolio.spec(assets = fund_edhec)
## Add constraints of maximizing return per unit CSM
pspec_CSMratio <- add.constraint(pspec_CSMratio, type = "full_investment")
pspec_CSMratio <- add.constraint(pspec_CSMratio, type = "long_only")
## Add objectives of maximizing return per unit CSM
pspec_CSMratio <- add.objective(pspec_CSMratio, type = "return", name = "mean")
pspec_CSMratio <- add.objective(pspec_CSMratio, type = "risk", name = "CSM",
                                arguments = list(p=0.05))

# Optimization
optimize.portfolio(ret_edhec, pspec_CSMratio, optimize_method = "CVXR", CSMratio = TRUE)
#> **********************************
#> PortfolioAnalytics Optimization
#> **********************************
#>
#> Call:
#> optimize.portfolio(R = ret_edhec, portfolio = pspec_CSMratio,
#>     optimize_method = "CVXR", CSMratio = TRUE)
#>
#> Optimal Weights:
#>     CA   CTAG     DS     EM    EMN     ED    FIA     GM    LSE     MA     RV     SS     FF
#> 0.0000 0.0723 0.0000 0.0000 0.0000 0.0000 0.0000 0.9277 0.0000 0.0000 0.0000 0.0000 0.0000
#>
#> Objective Measures:
#>     mean
#> 0.004088
#>
#>
#>     CSM
#> 0.02559
#>
#>
#> CSM ratio
#>    0.1598
```

# 9 Backtest Performance of MCSM, MES and MV Portfolios

CVXR solvers provide the second-order cone problem (SOCP) optimization capability required to compute MCSM portfolios. In this Section we use this capability first to compute MCSM portfolios, and compare their performance with that of MES and MV portfolios, which we do by computing and plotting out-of-sample cumulative gross returns (CGR). Then we do likewise for maximum CSMratio, MESratio and Sharpe ratio portfolios.

For these performance comparisons, we use CRSP® daily returns data from the `PCRA` package `stocksCRSPdaily` data set.[2] Specifically, the data set consists of daily returns of the 30 smallcap stocks with the largest market

[2] CRSP® stands for the Center for Research in Security Prices, LLC.

capitalizations among the 106 smallcap stocks in `stocksCRSPdaily`, from inception in 1993 until 2015. This the name of this data set is `retD_CRSP`.

```
# Get daily returns of the 30 smallcap stocks
library(PCRA)
library(PortfolioAnalytics)
library(xts)
stocksCRSPdaily <- getPCRAData(dataset = "stocksCRSPdaily")

smallcapTS <- selectCRSPandSPGMI(
  periodicity = "daily",
  stockItems = c("Date", "TickerLast", "CapGroupLast", "Return"),
  factorItems = NULL,
  subsetType = "CapGroupLast",
  subsetValues = "SmallCap",
  outputType = "xts")

# find top 30 small cap stocks based on the market capitalization
smallcapDT <- factorsSPGMI[CapGroupLast == "SmallCap"]
scSize <- smallcapDT[, mean(LogMktCap), by = "TickerLast"]
names(scSize)[2] <- "Size"
scSize <- scSize[order(scSize$Size, decreasing = TRUE),]
sc30largest <- scSize[,TickerLast][1:30]

# daily return of top 30 stocks
retD_CRSP <- smallcapTS[ , sc30largest]
```

The tickers of the stocks in `retD_CRSP` are"

```
names(retD_CRSP)
#>  [1] "AVP"  "PBI"  "ITT"  "MUR"  "GHC"  "THC"  "AMD"  "FMC"  "BMS"  "DDS"  "R"    "J"    "RDC"  "DBD
```

```
# monthly return of top 30 stocks needed for monthly rebalancing
ep <- endpoints(retD_CRSP, on= "months", k=1)
prod1 <- function(x){apply(x+1, 2, prod)}
retM_CRSP <- period.apply(retD_CRSP, INDEX = ep, FUN = prod1) - 1
```

The back-testing here is relatively slow, and takes most of the time in this Vignette. To give a sense of typical times, we provide the computing times for ad MacBook Air with M2, 8-Core CPU, 8-Core GPU and 16-core Neural Engine. The back-testing in Sections 9.1 and Section 9.2 takes about 3 minutes each to run, and the average running time for generating efficient frontiers in Section 9.3 is about 30 seconds.

## 9.1 Backtesting of GMV, GMES, GMCSM Portfolios

Here we use the daily returns data set `retD_CRSP` to generate comparative back-tests of Global Minimum Variance (GMV), Global Minimum ES (GMES) and Global Minimum CSM (GMCSM) portfolios. The strategy is to re-balance the portfolio at the end of each month with a rolling window of 500 days (approximately two years). We compare the performances of the portfolios with plots of their cumulative gross returns and maximum drawdowns.

The results are shown in Figure 9.1.

```
# Generate GMV, GMES and GMCSM portfolios
pspec_sc <- portfolio.spec(assets = sc30largest)
pspec_sc <- add.constraint(pspec_sc, type = "full_investment")
pspec_sc <- add.constraint(pspec_sc, type = "long_only")
```

```r
pspec_GMV <- add.objective(pspec_sc, type = "risk", name = "var")
pspec_GMES <- add.objective(pspec_sc, type = "risk", name = "ES")
pspec_GMCSM <- add.objective(pspec_sc, type = "risk", name = "CSM")

# Optimize with 500-Day Rolling Window and Monthly Rebalancing
bt.GMV <- optimize.portfolio.rebalancing(retD_CRSP, pspec_GMV,
                                         optimize_method = "CVXR",
                                         rebalance_on = "months",
                                         rolling_window = 500)
bt.ES <- optimize.portfolio.rebalancing(retD_CRSP, pspec_GMES,
                                         optimize_method = "CVXR",
                                         rebalance_on = "months",
                                         rolling_window = 500)
bt.CSM <- optimize.portfolio.rebalancing(retD_CRSP, pspec_GMCSM,
                                         optimize_method = "CVXR",
                                         rebalance_on = "months",
                                         rolling_window = 500)

# Extract time series of portfolio weights
wts.GMV <- extractWeights(bt.GMV)
wts.GMV <- wts.GMV[complete.cases(wts.GMV),]

wts.ES <- extractWeights(bt.ES)
wts.ES <- wts.ES[complete.cases(wts.ES),]

wts.CSM <- extractWeights(bt.CSM)
wts.CSM <- wts.CSM[complete.cases(wts.CSM),]

# Compute cumulative returns of three portfolios
GMV <- Return.rebalancing(retM_CRSP, wts.GMV)
ES <- Return.rebalancing(retM_CRSP, wts.ES)
CSM <- Return.rebalancing(retM_CRSP, wts.CSM)

# Combine GMV, ES and CSM portfolio cumulative returns
ret.comb <- na.omit(merge(GMV, ES, CSM, all=F))
names(ret.comb) <- c("GMV", "GMES", "GMCSM")

backtest.plot(ret.comb, colorSet = c("black", "darkblue", "darkgreen"), ltySet = c(3, 2, 1))
```

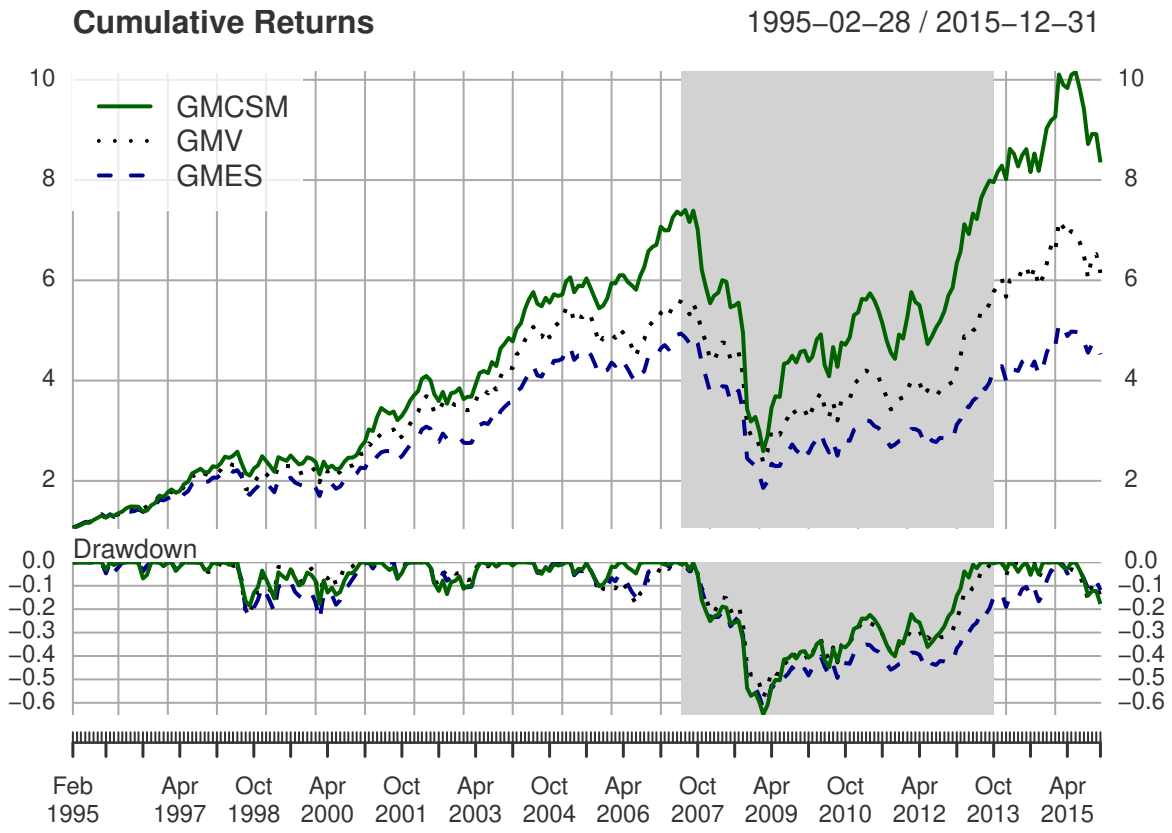


Fig 9.1 Cumulative Gross Returns and Maximum Drawdowns of GMES, GMCSM, and GMV Portfolios

## 9.2 Backtesting Maximum SR, ESratio, and CSMratio Portfolios

Here we follow the same methodology as in the previous subsection, except now our optimal portfolios are the maximum return-to-risk ratio portfolios maxSR, maxESration, and maxCSMratio. The results are shown in Figure 9.2.

```
# Generate Sr, ESr and CSMr portfolios
pspec_sc_ratio <- add.objective(pspec_sc, type = "return", name = "mean")
pspec_Sr <- add.objective(pspec_sc_ratio, type = "risk", name = "var")
pspec_ESr <- add.objective(pspec_sc_ratio, type = "risk", name = "ES")
pspec_CSMr <- add.objective(pspec_sc_ratio, type = "risk", name = "CSM")

# Optimize with 500-Day Rolling Window and Monthly Rebalancing
bt.Sr <- optimize.portfolio.rebalancing(retD_CRSP, pspec_Sr, maxSR = TRUE,
                                        optimize_method = "CVXR",
                                        rebalance_on = "months",
                                        rolling_window = 500)
bt.ESr <- optimize.portfolio.rebalancing(retD_CRSP, pspec_ESr,
                                        optimize_method = "CVXR",
                                        rebalance_on = "months",
                                        rolling_window = 500)
bt.CSMr <- optimize.portfolio.rebalancing(retD_CRSP, pspec_CSMr,
                                        optimize_method = "CVXR",
```

```
                                        rebalance_on = "months",
                                        rolling_window = 500)

# Extract time series of portfolio weights
wts.Sr <- extractWeights(bt.Sr)
wts.Sr <- wts.Sr[complete.cases(wts.Sr),]

wts.ESr <- extractWeights(bt.ESr)
wts.ESr <- wts.ESr[complete.cases(wts.ESr),]

wts.CSMr <- extractWeights(bt.CSMr)
wts.CSMr <- wts.CSMr[complete.cases(wts.CSMr),]

# Compute cumulative returns of three portfolios
Sr <- Return.rebalancing(retM_CRSP, wts.Sr, rebalance_on = "months")
ESr <- Return.rebalancing(retM_CRSP, wts.ESr, rebalance_on = "months")
CSMr <- Return.rebalancing(retM_CRSP, wts.CSMr, rebalance_on = "months")

# Combine Sr, ESr and CSMr portfolio cumulative returns
ret.comb.ratios <- na.omit(merge(Sr, ESr, CSMr, all=F))
names(ret.comb.ratios) <- c("Sharpe ratio", "ES ratio", "CSM ratio")

backtest.plot(ret.comb.ratios, colorSet = c("black", "darkblue", "darkgreen"), ltySet = c(3, 2, 1))
```

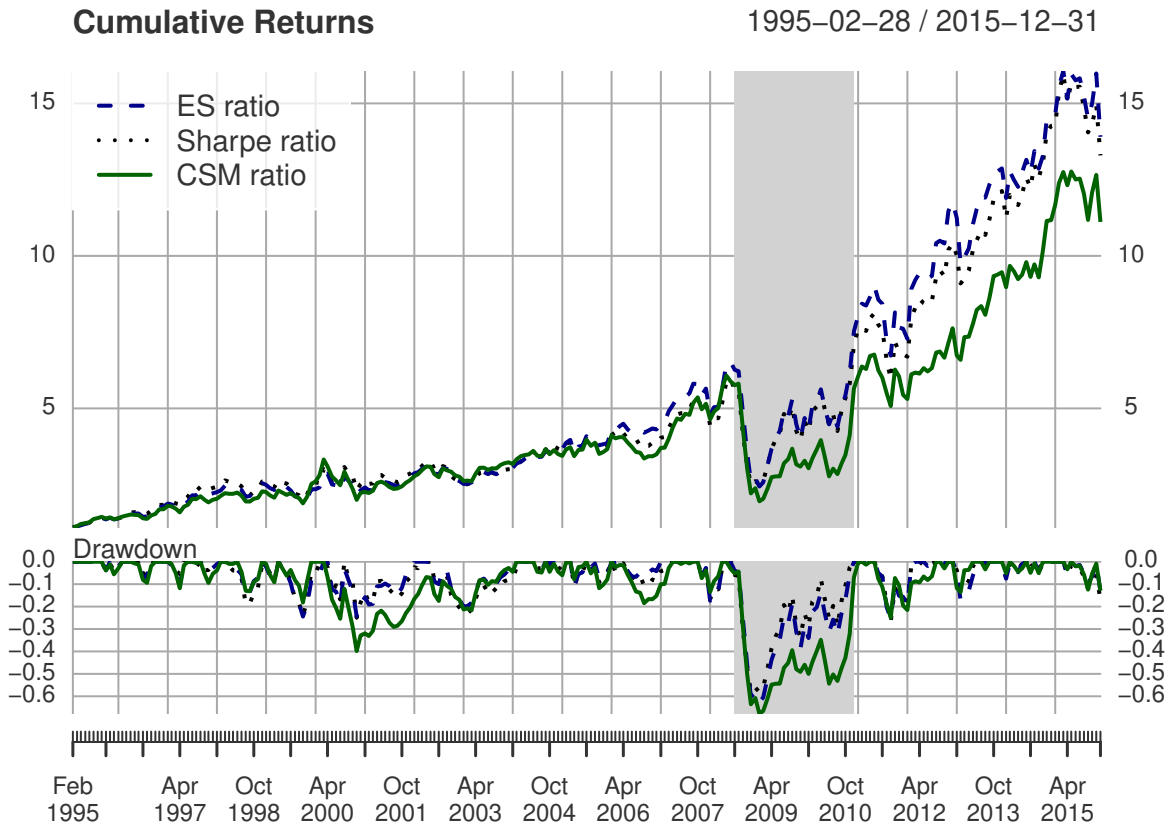


Fig 9.2 Cumulative Gross Returns and Maximum Drawdowns of maxSR, maxESratio, and maxCSMratio Portfolios.

# 10 Multiple Efficient Frontiers of Equal Risk Type Portfolios

In this Section we illustrate computation of efficient frontiers for mean-StdDev, mean-ES and mean-CSM portfolios for the cases: (1) Single portfolios, and (2) Multiple portfolios of the same risk type.

For this purpose, we use the data set `retM_CRSP_5` that contains monthly returns of 30 stocks from 2011-01 to 2015-12. These monthly returns are obtained from the Section 9 daily returns `retD_CRSP` for 1993 to 2015, using the following code.

```
# monthly return of top 30 stocks in last 5 years
ep <- endpoints(retD_CRSP, on= "months", k=1)
prod1 <- function(x){apply(x+1, 2, prod)}
retM_CRSP <- period.apply(retD_CRSP, INDEX = ep, FUN = prod1) - 1
retM_CRSP_5 <- tail(retM_CRSP, 60)
```

The following code line plots the `retD_CRSP` returns time series in Figure 10.1.

```
tsPlotMP(retM_CRSP_5, layout = c(2,15), yname = "RETURNS",
         stripText.cex = 0.7, axis.cex = 0.7)
```

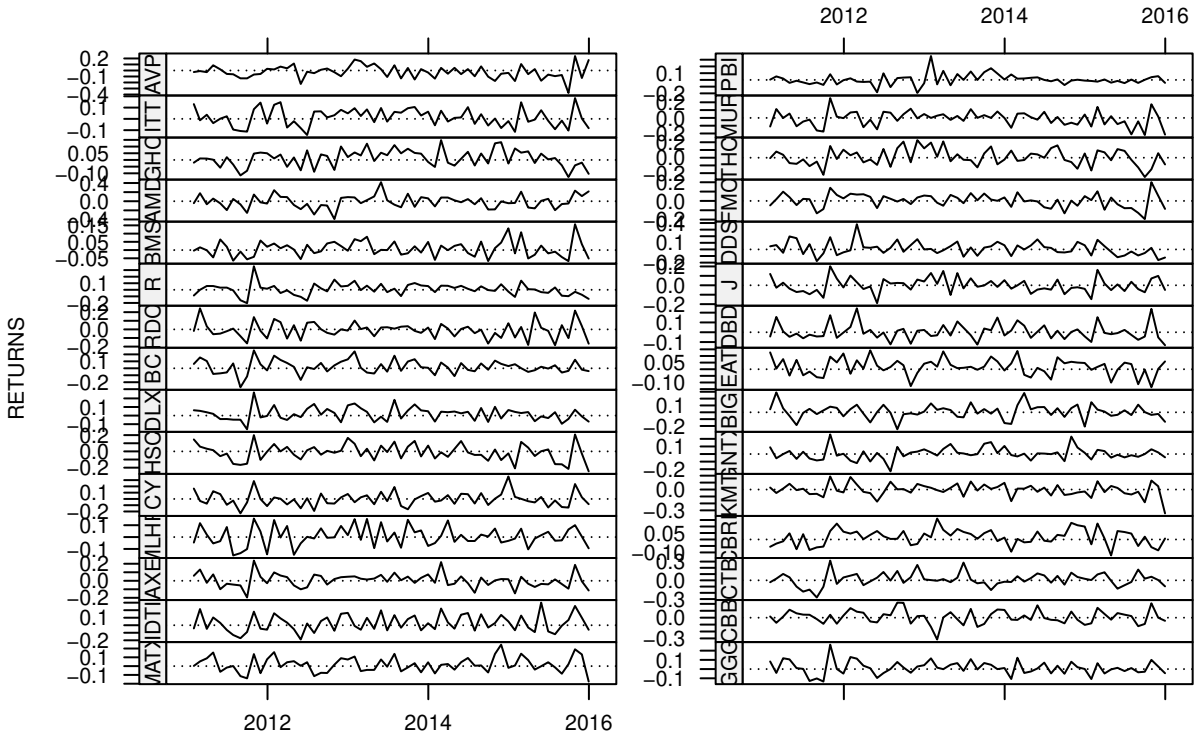


Fig 10.1 Monthly Returns of 30 Smallcap Stocks for 2011-2015.

The functions `create.EfficientFrontier` and `chart.EfficientFrontier` are the main functions for this Section. The first of these two functions computes portfolios efficient frontier mean return and risk values, where the portfolio is specified to have any one of a variety of weights constraints, and risk types. Then the `chart.EfficientFrontier` uses the mean return and risk values to plot the efficient frontier, or a small set of efficient frontiers corresponding to different portfolio specification. The following subsection provide illustrative examples.

## 10.1 Mean-Variance Efficient Frontiers

The code chunk below illustrated use the above two functions to create, and plot in Figure 10.2, the efficient frontier of a long-only minimum variance (MV) portfolios for the `retM_CRSP_5` monthly returns of 30 stocks for 2011 to 2015. The straight line tangent to the efficient frontier has a slope equal to the maximum achievable Sharpe ratio (SR) value of 0.416. The smallest mean-StdDev combination, indicated by a sold dot, is that of a global minimum variance (GMV) portfolio.

```
# mean-var efficient frontier
pspec_sc <- portfolio.spec(names(retM_CRSP_5))
pspec_sc <- add.constraint(pspec_sc, type = "full_investment")
pspec_sc <- add.constraint(pspec_sc, type = "long_only")

meanvar.ef <- create.EfficientFrontier(R = retM_CRSP_5,
                      portfolio = pspec_sc, type = "mean-StdDev")

chart.EfficientFrontier(meanvar.ef, match.col = "StdDev", type = "l",
      chart.assets = FALSE, main = NULL,
      RAR.text = "Max Sharpe ratio", pch = 1)
```

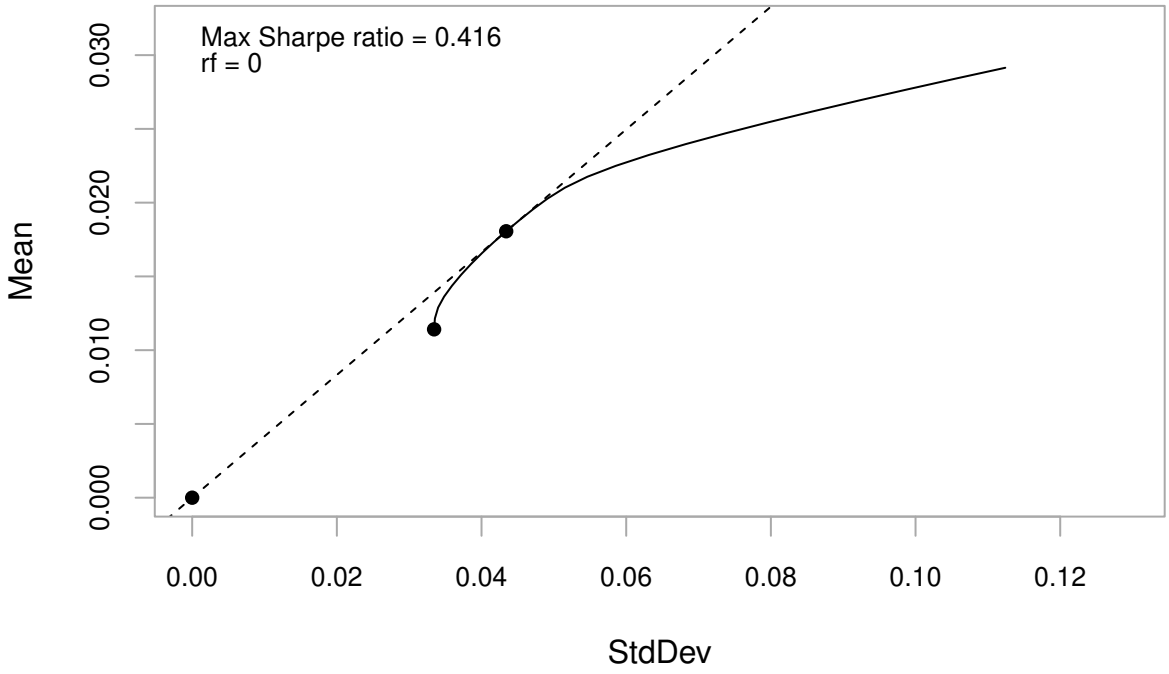


Fig 10.2 MV Efficient Frontier of Long-Only Portfolio for Monthly CRSP Returns 2011-2015.

The maximum Sharpe ratio in Figure 10.2 is computed automatically by the `chart.EfficientFrontier` function. One can check this value by computing the maximum Sharpe ratio on the efficient frontier, using a fairly large set of efficient frontier mean and standard deviation values, with the following code.

```
meanvar.ef$frontier[, 1:2]
#>                  mean     StdDev
#> result.1  0.01141173 0.03344369
#> result.2  0.01215050 0.03355371
#> result.3  0.01288927 0.03399953
#> result.4  0.01362804 0.03482453
```

```
#> result.5  0.01436681 0.03591935
#> result.6  0.01510558 0.03717268
#> result.7  0.01584435 0.03856204
#> result.8  0.01658312 0.04007328
#> result.9  0.01732189 0.04169314
#> result.10 0.01806066 0.04340948
#> result.11 0.01879943 0.04521131
#> result.12 0.01953820 0.04709002
#> result.13 0.02027697 0.04913130
#> result.14 0.02101574 0.05153252
#> result.15 0.02175451 0.05463076
#> result.16 0.02249328 0.05854817
#> result.17 0.02323205 0.06313372
#> result.18 0.02397082 0.06825285
#> result.19 0.02470959 0.07379462
#> result.20 0.02544836 0.07967088
#> result.21 0.02618713 0.08581858
#> result.22 0.02692591 0.09219807
#> result.23 0.02766468 0.09876525
#> result.24 0.02840345 0.10549834
#> result.25 0.02914222 0.11240968
```

```
sr <- meanvar.ef$frontier[, 1]/meanvar.ef$frontier[, 2]
maximumSR <- max(sr)
meanMaxSR <-  meanvar.ef$frontier[, 1][sr == max(sr)]
stdevMaxSR <- meanvar.ef$frontier[, 2][sr == max(sr)]
dat <- (round(c(maximumSR, meanMaxSR, stdevMaxSR), 3))
dat <- data.frame(dat)
names(dat) <- NULL
row.names(dat) <- c("maximum SR", "maxSRport Mean", "maxSRport Stdev")
dat
#>
#> maximum SR      0.416
#> maxSRport Mean  0.018
#> maxSRport Stdev 0.043
```

On the other hand, recall that in Section 8.1 we discussed a direct method of computing the maximum Sharpe ratio portfolio in Section 8.1, and it is of interest to check how accurate the above search method is for computing the maximum Sharpe ratio. The following code shows that the above computation of the maximum Sharpe ratio agrees with the direct method two 3 significant digits.

```
# Mean-StdDev Efficient Frontier
pspec_MV <- add.objective(pspec_sc, type = "risk", name = "var")
pspec_MV <- add.objective(portfolio = pspec_MV, type = "return", name = "mean")
opt_MV <- optimize.portfolio(retM_CRSP_5, pspec_MV, optimize_method = "CVXR", maxSR = TRUE)
opt.outputMvo(opt_MV, retM_CRSP_5, annualize = FALSE, digits = 3)
#> $Wgts
#>   AVP   PBI   ITT   MUR   GHC   THC   AMD   FMC   BMS   DDS     R     J   RDC   DBD    BC   EAT   DL
#> 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.075 0.020 0.000 0.000 0.000 0.000 0.000 0.317 0.07
#>
```

```
#> $Mean
#> [1] 0.018
#>
#> $StdDev
#> [1] 0.044
#>
#> $SR
#> [1] 0.416
```

One can easily create and plot multiple overlaid mean-StdDev efficient frontiers for a set of two or more constraints. The following code does so for a set of constraints consisting of a long-only constraint, a long box constraint, and a long-short box constraint, and the results are shown in Figure 10.3.

```
pspec_sc_init <- portfolio.spec(assets = sc30largest)
pspec_sc_init <- add.constraint(pspec_sc_init, type = "full_investment")

# Portfolio with long-only constraints
pspec_sc_lo <- add.constraint(portfolio = pspec_sc_init, type = "long_only")

# Portfolio with long-only box constraints
pspec_sc_lobox <- add.constraint(portfolio = pspec_sc_init, type = "box",
                                 min = 0.02, max = 0.1)

# Portfolio with long-short box constraints
pspec_sc_lsbox <- add.constraint(portfolio = pspec_sc_init, type = "box",
                                 min = -0.1, max = 0.1)

# Combine the portfolios into a list
portf_list <- combine.portfolios(list(pspec_sc_lo, pspec_sc_lobox, pspec_sc_lsbox))

# Plot the efficient frontier overlay of the portfolios with varying constraints
legend_labels <- c("Long Only", "Long Only Box (0.02,0.1)", "Long Short Box (-0.01,0.1)")
chart.EfficientFrontierOverlay(R = retM_CRSP_5, portfolio_list = portf_list,
                               type = "mean-StdDev", match.col = "StdDev",
                               legend.loc = "bottomright", chart.assets = FALSE,
                               legend.labels = legend_labels, cex.legend = 1,
                               labels.assets = FALSE, lwd = c(3,3,3),
                               col = c("black", "dark red", "dark green"),
                               main = NULL,
                               xlim = c(0.03, 0.11), ylim = c(0.005, 0.035))
```

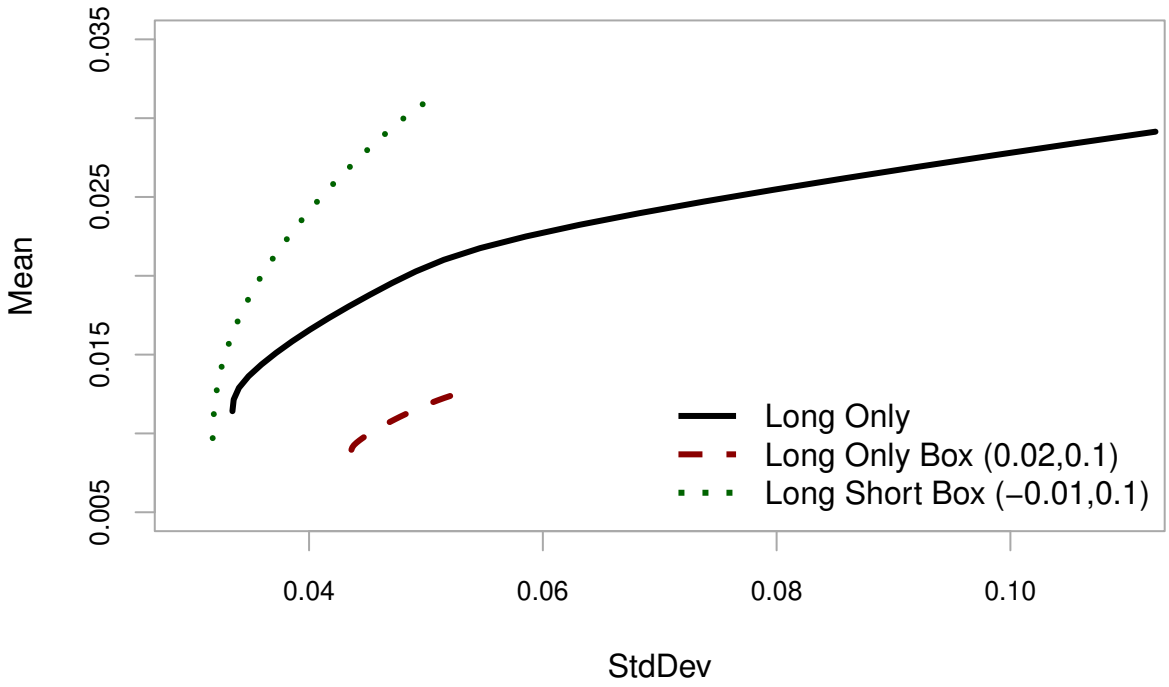


Fig 10.3 MV Efficient Frontiers with Long-Only, Long Box and Long-Short Box Constraints for Monthly CRSP Returns 2011-2015.

The plot clearly shows that the long-short box constrained portfolio has the best performance, though it also requires shorting which may not be possible for many real-world portfolios.

## 10.2 Mean-ES Efficient Frontiers

Figure 10.4 displays the minimum ES (MES) efficient frontier for a long-only portfolio of the monthly CRSP returns from 2011 to 2015. The straight line tangent to the efficient frontier has a slope equal to the maximum achievable mean return-to-MES ratio of 0.264. The smallest mean-ES combination, indicated by a sold dot, is that of a global minimum ES (GMES) portfolio.

```
# Mean-ES Efficient Frontier
meanes.ef <- create.EfficientFrontier(R = retM_CRSP_5, portfolio = pspec_sc, type = "mean-ES")
chart.EfficientFrontier(meanes.ef, match.col = "ES", type = "l",
                        chart.assets = FALSE, main = NULL,
                        RAR.text = "Max ES ratio", pch = 1)
```

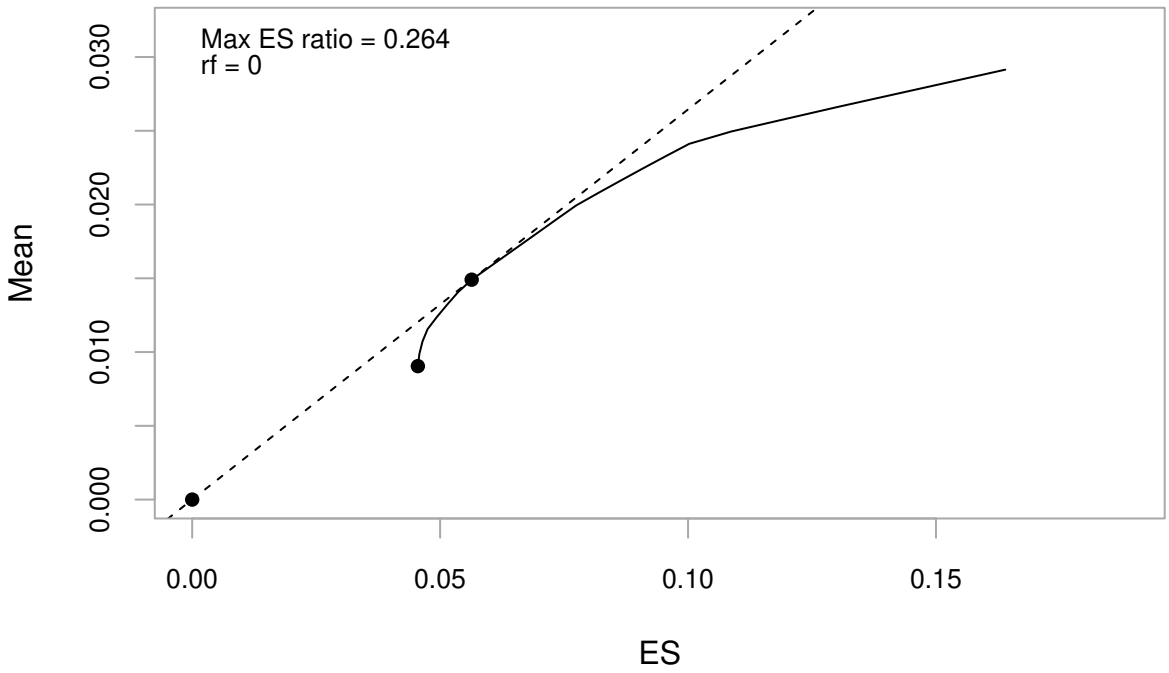


Fig 10.4 MES Efficient Frontier of Long-Only Portfolio for Monthly CRSP Returns 2011-2015.

Figure 10.5 shows the results of using the following code to generate overlaid MES efficient frontiers for the Long-Only, Long Box, and Long-Short Box constraints used for Figure 10.3.

```
legend_labels <- c("Long Only ES (p=0.05)",
                   "Long Only Box ES (p=0.05)", "Long Short Box ES (p=0.05)")
chart.EfficientFrontierOverlay(R = retM_CRSP_5, portfolio_list = portf_list,
                               type = "mean-ES", match.col = "ES",
                               legend.loc = "bottomright", chart.assets = FALSE,
                               legend.labels = legend_labels, cex.legend = 1,
                               labels.assets = FALSE, lwd = c(3,3,3),
                               col = c("black", "dark red", "dark green"),
                               main = NULL,
                               xlim = c(0.03, 0.17), ylim = c(0.005, 0.035))
```

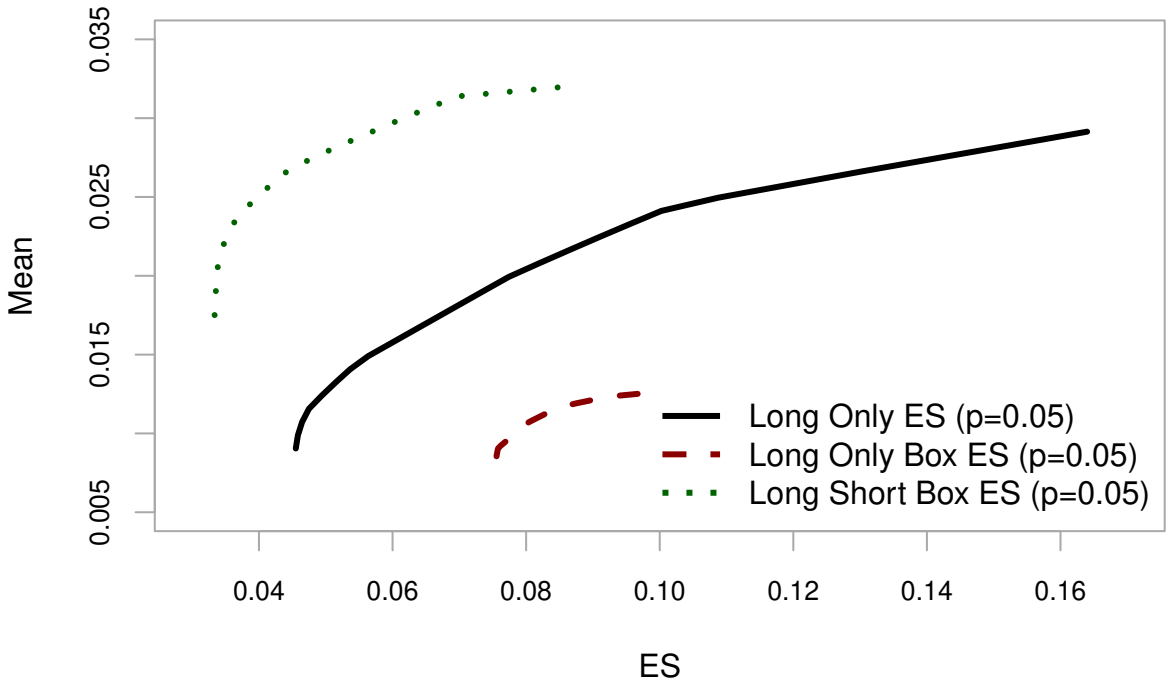


Fig 10.5 MES Efficient Frontiers for the Long-0nly, Long Box and Long-Short Box Constraints for Monthly CRSP Returns 2011-2015.

For each mean-ES efficient frontier, the left endpoint is the global minimum ES and its corresponding mean return, and the right endpoint is the maximum mean return and and its corresponding ES.

Note that the ordering of the three constrained portfolio efficient frontiers in Figure 10.5 is the same as for the mean-StdDev efficient frontiers in Figure 10.3 based the same constraints.

Relative to the Long-Only efficient frontier: (a) The Long Box constraint is quite severe, and results in a very limited range pf return and risk possibilities, and (b) The Long-Short Box Constraint results in a substantially better range of return possibilities. However, short positions entail costs and risks that many investors will avoid.

Finally, one can also create plots of overlaid MES efficient frontiers for different tail probability values $\gamma$, as is illustrated by following code and Figure 10.6:

```
# Create long-only ES portfolios with different tail probabilities
ES_05 <- add.objective(portfolio = pspec_sc_lo, type = "risk", name = "ES",
                       arguments = list(p=0.05))

ES_10 <- add.objective(portfolio = pspec_sc_lo, type = "risk", name = "ES",
                       arguments = list(p=0.1))

ES_15 <- add.objective(portfolio = pspec_sc_lo, type = "risk", name = "ES",
                       arguments = list(p=0.15))

# Combine the portfolios into a list
portf_ES_list <- combine.portfolios(list(ES_05, ES_10, ES_15))

# Plot the efficient frontier overlay of the portfolios with varying tail probabilities
legend_ES_labels <- c("ES (p=0.05)", "ES (p=0.1)", "ES (p=0.15)")
chart.EfficientFrontierOverlay(R = retM_CRSP_5, portfolio_list = portf_ES_list,
```

```
                                type = "mean-ES", match.col = "ES",
                                legend.loc = "bottomright", chart.assets = FALSE,
                                legend.labels = legend_ES_labels, cex.legend = 1,
                                labels.assets = FALSE, lwd = c(3,3,3),
                                col = c("black", "dark red", "dark green"),
                                main = NULL,
                                xlim = c(0.035, 0.165), ylim = c(0.005, 0.03))
```

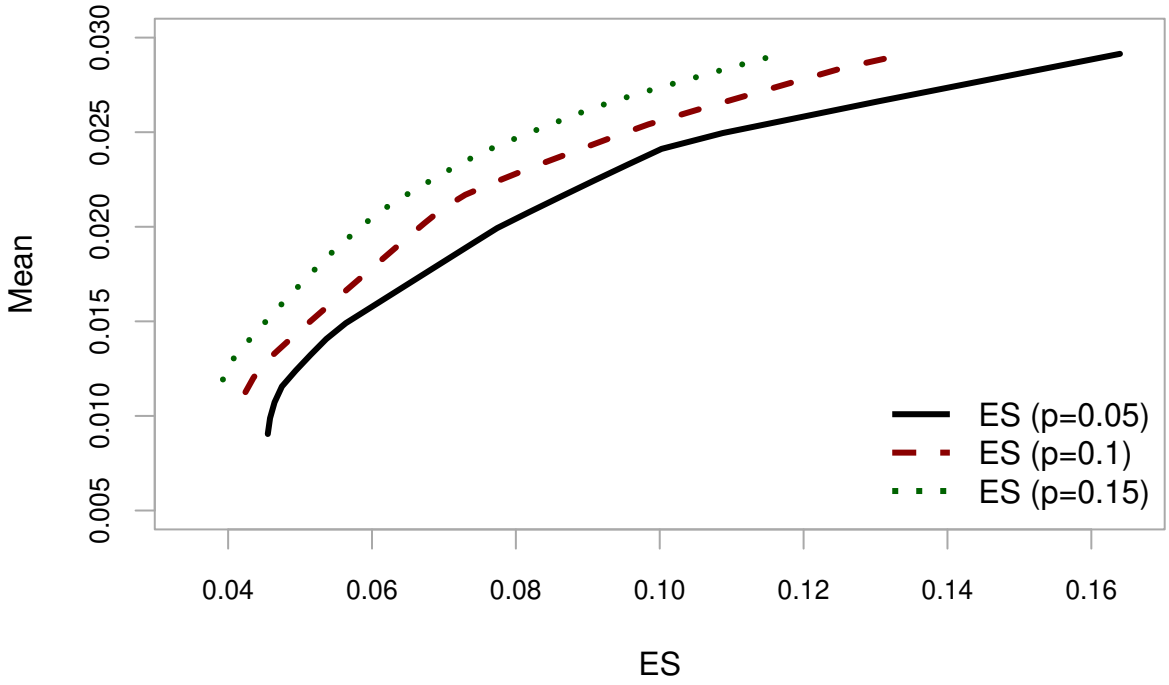


Fig 10.6 MES Efficient Frontiers Based on Tail Probabilities 0.05, 0.10, 0.15, for Monthly CRSP Returns 2011-2015.

For MES portfolios with equal mean returns, the one with a larger tail probability will have better risk performance, i.e., a smaller risk, than an ES portfolio with a smaller tail probability. Alternatively, for portfolios with equal MES values, ones for larger tail probabilities will have better mean returns performance, i.e., higher mean returns, than ones with smaller tail probabilities. These natural behaviors are reflected in the three mean-ES efficient frontiers in Figure 10.7.

## 10.3 Mean-CSM Efficient Frontier

The code below generates the default tail probability 0.05 mean-CSM (MCSM) efficient frontier displayed in Figure 10.7, where the straight line tangent to the efficient frontier has a slope equal to the maximum achievable meanCSMratio value of 0.261. The smallest mean return-MCSM combination, indicated by a sold dot, is that of a global minimum CSM (GMCSM) portfolio.

```
# Mean-CSM Efficient Frontier
meancsm.ef <- create.EfficientFrontier(R = retM_CRSP_5, portfolio = pspec_sc,
                                       type = "mean-CSM")
chart.EfficientFrontier(meancsm.ef, match.col = "CSM", type = "l",
                        chart.assets = FALSE, main = NULL,
                        RAR.text = "Max CSM ratio", pch = 1)
```

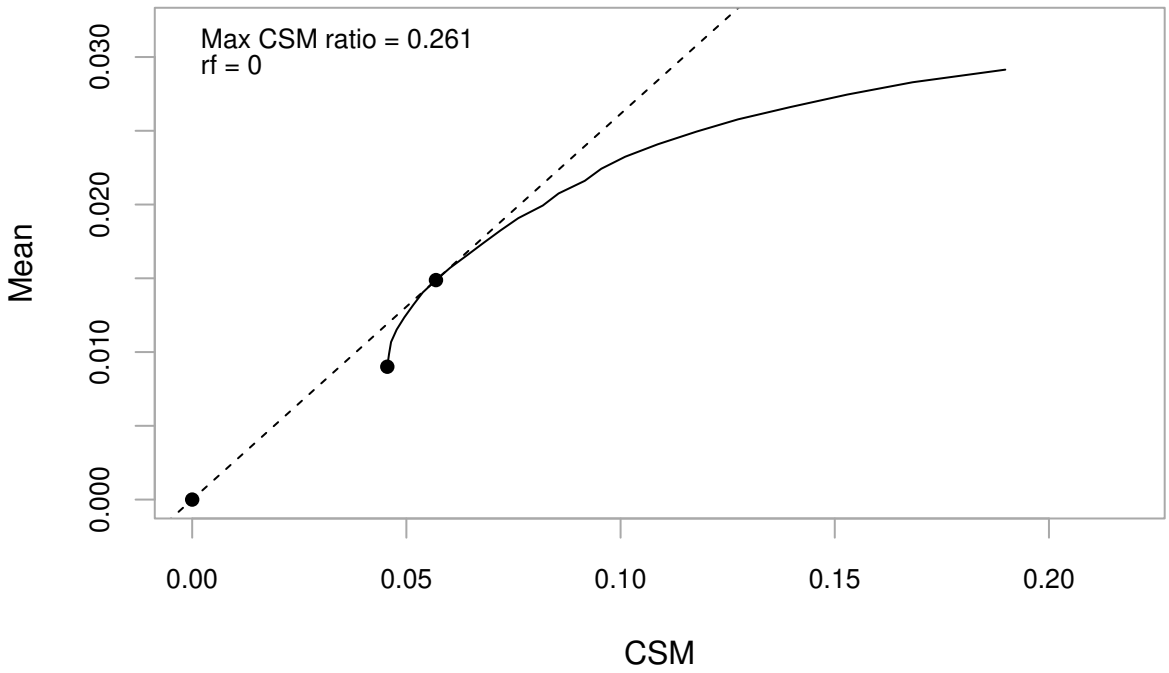


Fig 10.7 Minimum CSM Efficient Frontiers for for Monthly CRSP Returns 2011-2015.

Using code similar to that used to create Figures 10.5 and 10.6, you can created CSM efficient frontiers for several types of weight constraints, and several tail probabilities.

# 11 Multiple Efficient Frontiers With Different Risk Types

The method of comparing two or more efficient frontiers in Section 10 only works for portfolios that are all based on the same risk type, e.g., all Variance/StdDev, or all ES, or all CSM portfolios. But it is highly desirable to be able to compare efficient frontiers of two or more portfolios that are based on different risk types, e.g., a StdDev risk portfolio and an ES risk portfolio, using either one of those risk types as the basis of comparison.

## 11.1 An Efficient Frontiers Comparison Function

The function `chart.EfficientFrontierCompare` allows you to make the above type of comparisons. The arguments of this function are

```
args(chart.EfficientFrontierCompare)
#> function (R, portfolio, risk_type, n.portfolios = 25, match.col = c("StdDev",
#>     "ES"), guideline = NULL, main = "Efficient Frontiers", plot_type = "l",
#>     cex.axis = 0.5, element.color = "darkgray", legend.loc = NULL,
#>     legend.labels = NULL, cex.legend = 0.8, xlim = NULL, ylim = NULL,
#>     ..., chart.assets = TRUE, labels.assets = TRUE, pch.assets = 21,
#>     cex.assets = 0.8, col = NULL, lty = NULL, lwd = NULL)
#> NULL
```

where `risk_type` is the risk to be compared, and `match.col` is the vector of the risk types of the portfolios whose efficient frontiers are to be compared. If there are only two frontiers in the comparison, the default of the argument `guideline` is `TRUE`, which results horizontal and vertical dotted lines, whose lengths measure the decrease in risk of the minimum risk portfolios, and a particular increase in mean return that will be clear from the efficient frontiers plot.

Figure 10.8 compares StdDev and ES frontiers, when plotted versus StdDev risk. Since a StdDev frontier is an efficient frontier when plotted versus StdDev risk, it is not surprising to see the StdDev frontier dominate the ES frontier.

```
# Compare StdDev of minStd and minES portfolios with guideline
chart.EfficientFrontierCompare(R = retM_CRSP_5, portfolio = pspec_sc, risk_type = "StdDev",
                               match.col = c("StdDev", "ES"), lwd = c(2, 2),
                               xlim = c(0.0,0.14), ylim = c(0.005,0.032))
```

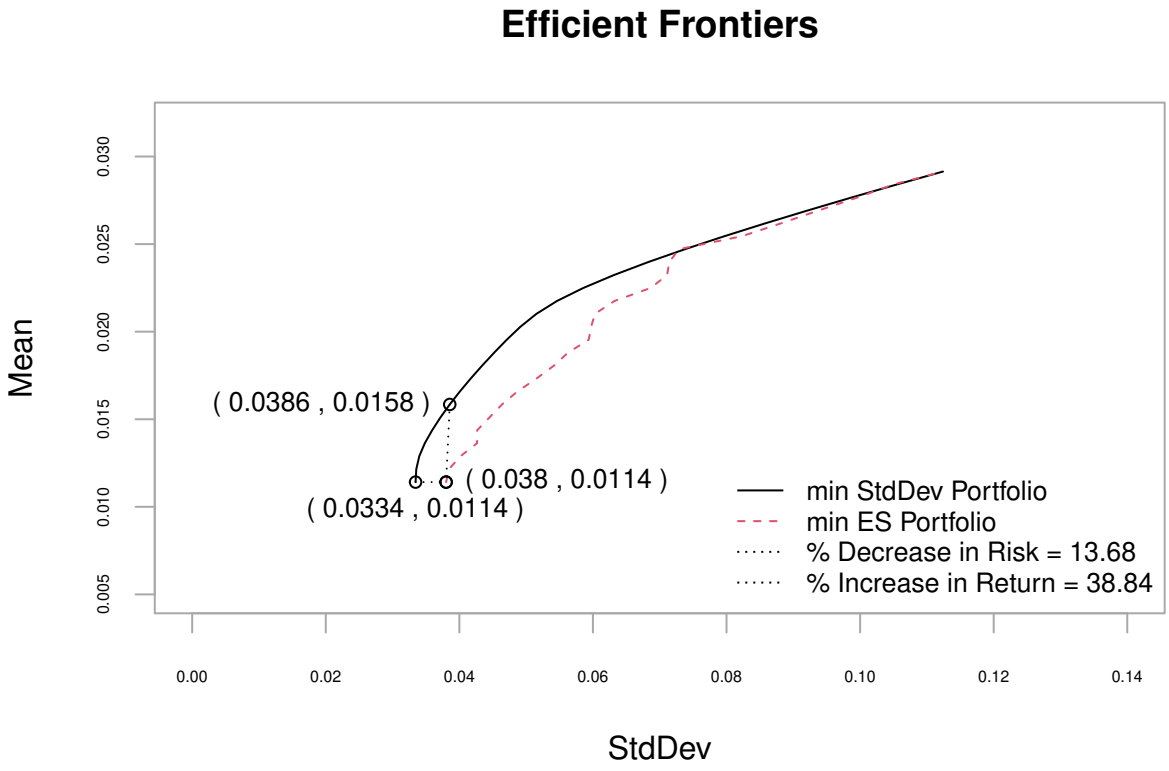


Fig 10.8 StdDev and ES Portfolio Frontiers Versus StdDev Risk.

Now we interchange `StdDev` and `ES` in the above Example, and get the result in Figure 10.9, where the ES efficient frontier dominates the StdDev frontier as expected.

```
# Compare ES of minStd and minES portfolios with guideline
chart.EfficientFrontierCompare(R = retM_CRSP_5, portfolio = pspec_sc, risk_type = "ES",
                               match.col = c("ES", "StdDev"), lwd = c(2, 2),
                               xlim = c(0.0,0.14), ylim = c(0.005,0.032))
```

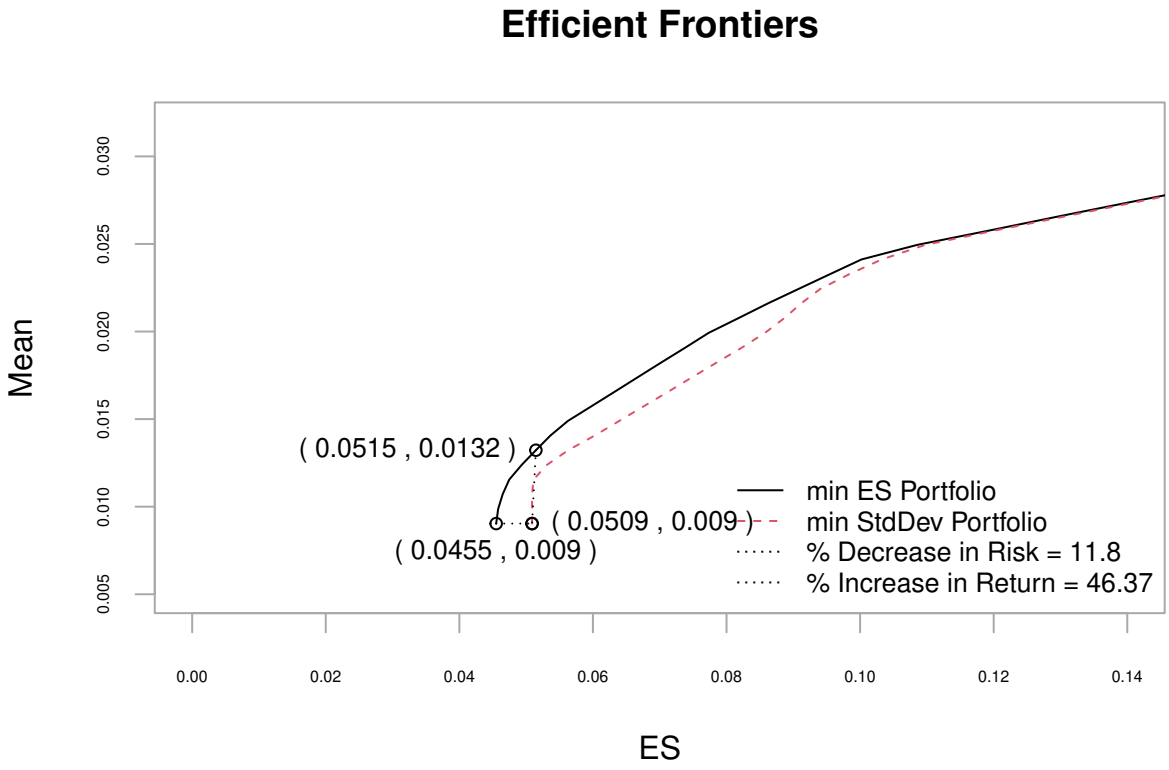


Fig 10.9 StdDev and ES Portfolio Frontiers Versus ES Risk.

This next example compares ES, CSM and StdDev portfolio frontiers versus ES risk. Not surprisingly the ES frontier dominates since it is an efficient frontier. The CSM frontier is only a tiny bit worse than the ES efficient frontier, which is also not surprising since the ES and CSM portfolio both penalize downside risk beyond threshold determined by same tail probability. Finally, the StdDev portfolio frontier performance is not surprisingly the worst.

```
# Compare ES of minStd, minES and minCSM portfolios without guideline
chart.EfficientFrontierCompare(R = retM_CRSP_5, portfolio = pspec_sc, risk_type = "ES",
                               match.col = c("StdDev", "ES", "CSM"), guideline = FALSE,
                               col = c("darkred","darkgreen","blue"), lwd = c(1, 1.5, 1.5),
                               lty = c("dotted", "solid", "dashed"))
```

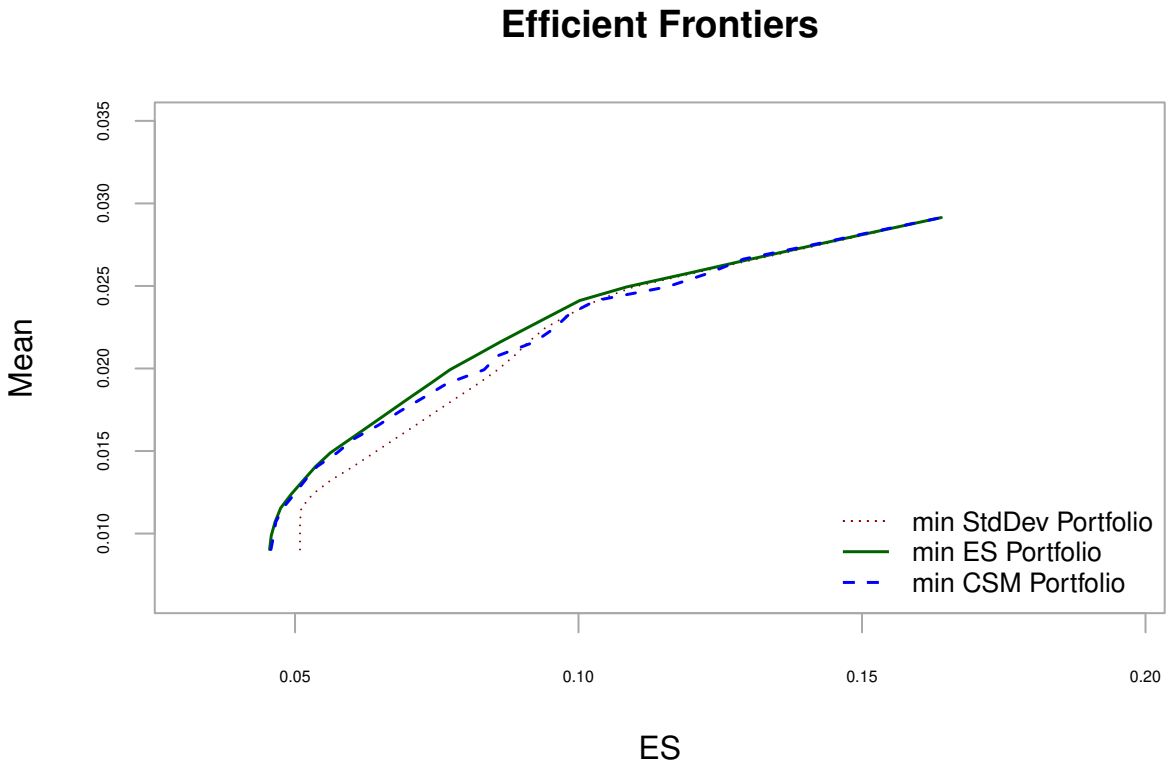


Fig 10.9 ES, CMS and StdDev Frontiers Versus ES Risk

## 11.2 A Risk Extraction Function

The `chart.EfficientFrontierCompare` function makes use of the convenience function `extract_risk`, which extracts the values of the StdDev, ES, and CMS risks of a given portfolio. This what allows `chart.EfficientFrontierCompare` to plot efficient frontiers of different risk types versus a particular risk measure value of a user's choice.

The following line shows the arguments of `extract_risk`:

```
extract_risk(R, w, ES_alpha = 0.05, CSM_alpha = 0.05, moment_setting = NULL)
```

The asset returns `R` is an xts object, which must be specified, and the portfolio weights vector `w` of an optimized portfolio must be specified. But `ES_alpha` and `CSM_alpha` are optional arguments, with default values 0.05. The `moment_setting` argument allows a portfolio manager to specify an alternative covariance matrix in place of a sample covariance matrix used to compute an MV portfolio. For example, the manager might want to use a shrinkage covariance matrix, or a robust covariance matrix that, unlike the sample covariance matrix, is not much influenced by outliers. For information about specifying alternative robust covariance matrices, see the Vignette *robustCovMatForPA.pdf* and the demo code *demo_robustCovMatForPA.R*.

Here we create minimum standard deviation and minimum ES portfolios of the 30 CRSP monthly returns from 2011 to 2015 in the data set retM_CRSP_5, and use their portfolio weight vectors to extract the values of the mean, and the values of the StdDev, ES and CMS risk measures for each of these two portfolios.

```
minStdDev_port <- add.objective(pspec_sc, type = "risk", name = "StdDev")
minStdDev_opt <- optimize.portfolio(retM_CRSP_5, minStdDev_port,
                                    optimize_method = "CVXR")
minStdDev_w <- minStdDev_opt$weight
```

```
minES_port <- add.objective(pspec_sc, type = "risk", name = "ES")
minES_opt <- optimize.portfolio(retM_CRSP_5, minES_port,
                                optimize_method = "CVXR")
minES_w <- minES_opt$weight

# Extract risk StdDev portfolio
extract_risk(retM_CRSP_5, minStdDev_w)
#> $mean
#> [1] 0.01141173
#>
#> $StdDev
#>             [,1]
#> [1,] 0.03344369
#>
#> $ES
#> [1] 0.05088981
#>
#> $CSM
#> [1] 0.05615236

# Extract risk ES portfolio
extract_risk(retM_CRSP_5, minES_w)
#> $mean
#> [1] 0.009032428
#>
#> $StdDev
#>             [,1]
#> [1,] 0.04359565
#>
#> $ES
#> [1] 0.04574218
#>
#> $CSM
#> [1] 0.04578213
```

It is not surprising to see that the StdDev of the first portfolio above is smaller than that of the second portfolio, and it is not surprising to see that the ES of the second portfolio is smaller than that of the first.

Now we create a minimum StdDev portfolio using a robust covariance matrix estimate instead of the sample covariance matrix, and use its weight vector along with the retM_CRSP_5 data set to extract the portfolio mean return and the values of the risk measures StdDev, ES and CMS.

```
minStdDev_opt_covRob <-optimize.portfolio(retM_CRSP_5, minStdDev_port,optimize_method = "CVXR",
                                          momentFUN = 'custom.covRob.Mcd')
extract_risk(retM_CRSP_5, minStdDev_w, moment_setting = minStdDev_opt_covRob$moment_values)
#> $mean
#>             [,1]
#> [1,] 0.01471509
#>
#> $StdDev
#>             [,1]
#> [1,] 0.03227816
#>
```

```
#> $ES
#> [1] 0.05088981
#>
#> $CSM
#> [1] 0.05615236
```

Here we see that while the mean return and standard deviation of the robust covariance matrix minimum StdDev portfolio are different than those for the sample covariance based minStdDev_opt portfolio above, the ES and CSM values are unchanged.

# 12 General Multiple Efficient Frontiers

Section 11 showed how to create plots of multiple efficient frontiers that have not only different constraint types as in earlier Sections, but also how to create plots that also have different objectives. However, there are other types of portfolio efficient frontier comparisons that one may want to make, that are not characterized by only by different constraint types and/or different objectives. A leading example is the case where one wants to compare efficient frontiers of MV portfolios that use different covariance matrix estimators besides the standard sample covariance matrix. For example, one may wish to compare MV portfolios based on different covariance matrices, e.g., the sample covariance matrix, and one or more robust covariance matrix estimates, or one or more shrinkage covariance matrix estimates.

PortfolioAnalytics now has the function `plotFrontiers` that supports such comparisons. Here we use this function to compare the frontiers of MV portfolios based on the sample covariance matrix, and on a robust covariance matrix estimate called the `covRobRocke` estimator.

Considering that PortfolioAnalytics already provides functions to generate efficient frontiers, which are `meanvar.efficient.frontier`, `meanes.efficient.frontier` and `meancsm.efficient.frontier`, we create a new function `plotFrontiers` to visualize all the frontier values for broader comparison.

```
# Compare mean-var frontiers with classic and robust
# covariance matrix estimators.
sampleCov = meanvar.efficient.frontier(pspec_sc, retM_CRSP_5, optimize_method = 'CVXR')
robustCov = meanvar.efficient.frontier(pspec_sc, retM_CRSP_5, optimize_method = 'CVXR',
                                       momentFUN = 'custom.covRob.TSGS')
plotFrontiers(retM_CRSP_5, frontiers=list(sampleCov, robustCov), risk='StdDev')
```

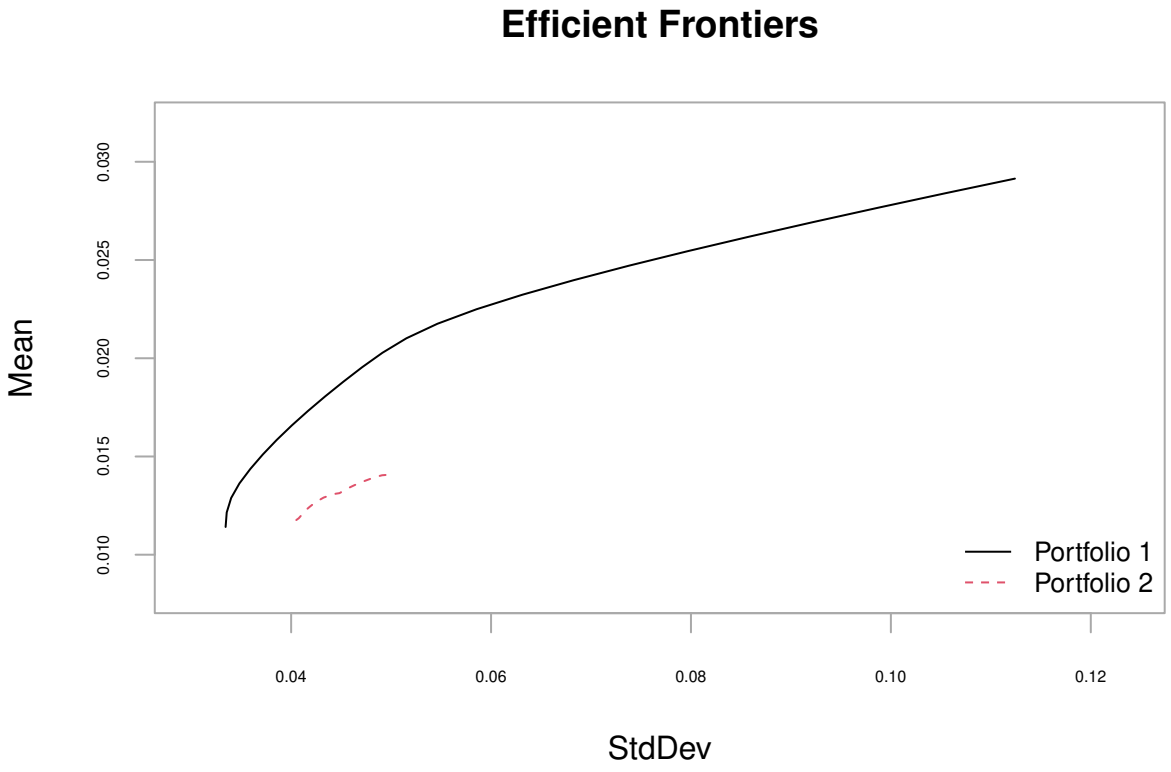


Fig 10.10

## Acknowledgements

This vignette aims to show the extended functionality of PortfolioAnalytics. The CVXR solvers capability and an extensive coherent second moment (CSM) capability in PortfolioAnalytics were implemented by Xinran Zhao when she was a graduate student in the University of Washington Department of Applied Mathematics Computational Finance and Risk Management M.S. degree program (MS-CFRM). For part of this work, she was supported by a Google Summer of Code (GSoC 2022) project funding, with mentors Doug Martin and Steve Murray.

Then during 2023 and 2024 Xinran continued to developed more PortfolioAnalyics functionality not limited to the implementation of CVXR and CSM, but more focused on their applications, including for portfolio performance back-testing capability and efficient frontiers analysis. The results of that effort are well reflected in this Vignette.

We also note that Xinran's MS-CFRM program classmate Yifu Kang, who was funded by another Portfolio-Analytics GSoC 2022 project, expanded PortfolioAnalytics by adding the capability to use robust covariance matrix estimators in minimum variance portfolio optimization. His efforts were very much appreciated.

Finally, this project could not have been completed without the help of Brian Peterson, who is an Author and the Maintainer of the PortfolioAnalytics package on CRAN. The work reflected in this Vignette would not have been possible without Brian's extensive contributions to the development of PortfolioAnalytics over many years.

# Reference

Cornuejols, Gerard, Javier Pena, and Reha Tutuncu. 2018. "Optimization Methods in Finance Second Edition."

Krokhmal, Pavlo A. 2007. "Higher Moment Coherent Risk Measures."

Rockafellar, R Tyrrell, Stanislav Uryasev, et al. 2000. "Optimization of Conditional Value-at-Risk." *Journal of Risk* 2: 21–42.