Vignette for Robust Portfolio Optimization using CVXR

Yifu Kang

6/23/2022

The purpose of this vignette is to demonstrate how to use package CVXR to implement Robust Portfolio Optimization as described in Ceria and Stubbs (2006). Ever since Markowitz had introduced his Mean-Variance Optimization Theory, it has been widely accepted as the main framework in asset allocation and portfolio management. However, skeptics have also appeared as practitioners found the Mean-Variance Optimization is sensitive to slight changes in input parameters, such as estimates of expected return and covariance matrix.

Lots of efforts and analysis have been made to demonstrate that Mean-Variance portfolio is sensitive to small changes in input data. Not only the weights of assets in optimal portfolio would have huge differences, the financial impact of the optimal portfolio associated with the input data would show stong biases, such as Sharpe Ratio and overall return. And it is believed that there is much more estimation risk coming from the estimates of expected return rather than the estimates of higher moments. To address the uncertainty in estimates of expected return, approaches including James-Stein estimators(J. D. Jobson (1981)) and Black-Litterman(Fischer Black (1990)) approach have been applied in order to create a more stable and realistic optimal portfolio. James-Stein estimators shrink the estimates of expected return towards the true expected return, while Black-Litterman approach generates an estimates of expected return combining with investors' personal views on specific assets. Other strategies like increasing the risk aversion parameter, resampling portfolio and adding additional constraints had also been used to control estimation error during the construction process of portfolio.

In this vignette, we would construct Mean-Variance Portfolio using a method called Robust Portfolio Optimization. Compared to methods mentioned above, robust optimization considers the estimation error in the process of optimizing portfolio, rather than preprocessing the input data. It gives the investors a more conservative and prudent way to construct portfolio by minimizing the worst case return given a confidence region. The portfolio construction in this vignette utilize a sophisticated framework of Robust Portfolio

Optimization introduced in Ceria and Stubbs (2006).

The classical Markowitz Mean-Variance Portfolio is known with two basic equivalent versions. The first and the most common one is to minimize the portfolio variance subject to a target return constraint on the portfolio. The second form that is used in this Robust Portfolio Optimization, is to maximize the portfolio return subject to a target portfolio variance constraint. The latter version of classical Markowitz Mean-Variance Portfolio has following mathematical form:

$$\begin{aligned} \max_{w} \quad & \bar{\boldsymbol{\alpha}}^{\boldsymbol{T}} w \\ \text{s.t.} \quad & e^T w = 1 \\ & w^T Q w \leq \nu \\ & w \geq 0 \end{aligned} \tag{1}$$

,

where $\bar{\alpha}$ is the estimated expected return vector and Q is the true returns' covariance matrix. Assuming α is normally distributed, and given an estimated expected return $\bar{\alpha}$ and an estimation error matrix Σ, it's assumed that α lies in the confidence region

$$(\alpha - \bar{\alpha})^T \Sigma^{-1} (\alpha - \bar{\alpha}) \leq \kappa^2,$$

where $\kappa^2 = \chi_n^2(1-\eta)$ and χ_n^2 is the inverse cumulative distribution function of the chi-squared distribution with n degrees of freedom. To obtain the maximum difference between estimated expected return and true expected return of the portfolio, we have following optimization problem

$$\begin{aligned} \max_{\alpha} \quad & \bar{\alpha}^T w - \alpha^T w \\ \text{s.t.} \quad & (\alpha - \bar{\alpha})^T \Sigma^{-1} (\alpha - \bar{\alpha}) \leq \kappa^2 \end{aligned} \tag{2}$$

Solving above problem gives us the lowest possible value of true expected return with respect to the estimated expected return as

$$\alpha = \bar{\alpha} - \sqrt{\frac{\kappa^2}{w^T \Sigma w}} \Sigma w.$$

Then the lowest possible value of true expected return of the portfolio is

$$\alpha^T w = \bar{\alpha}^T w - \kappa \left\| \Sigma^{1/2} w \right\|.$$

Since we want to simultaneously maximize the expected return of the portfolio and minimize the difference between estimated expected return and true expected return, we then have following optimization problem

$$\begin{aligned} \max_{w} \quad & \bar{\alpha}^T w - \kappa \left\| \Sigma^{1/2} w \right\| \\ \text{s.t.} \quad & e^T w = 1 \\ & w^T Q w \leq \nu \\ & w \geq 0 \end{aligned} \tag{3}$$

In the following sections, we will use R code to show how to calculate optimal weights of above Robust Portfolio Optimization and its variants. For more details of how to derive above Robust Portfolio Optimization problem, please refer to Ceria and Stubbs (2006).

In this Vignette, we introduce how to use ***CVXR*** package to implement Robust Portfolio Optimization. Also, we use two different sets of data to compute the efficient frontiers for comparison of classical optimization and robust optimization.

1.1 Load Packages

Load the necessary packages. As the main package we use is CVXR. we want to specify some key API that are from this important package:

CVXR: 1. *Variable()*: class used to initiate the optimization variable

2. *quad_form()*: function used to calculate quadratic form of $x^T P x$

3. *Problem()*: class representing a convex optimization problem

4. *Maximize/Minimize*: class representing an optimization objective for maximization/minimization

5. *solve()*: function to solve a DCP compliant optimization problem

MASS package is imported for its function *mvrnorm* to simulate multivariate normal distribution, and ***expm*** package is imported for its function *sqrtm()* to get the squared root of a matrix.

```
library(PortfolioAnalytics)
library(CVXR)
library(MASS)
library(expm)
```

1.2 Load Data

The dataset we use is provided in Tom Idzorek (2003). Readers can import the data by loading Rdata file "data2006.Rdata". There are three variables: expectedReturn, stdDev and corrMat, which are the true expected return vector α, standard deviation vector and correlation matrix for 8 investments. With standard deviation vector and tru correlation matrix, we can calculate the true covariance matrix Q.

Using the true expected return and true covariance matrix, we randomly generate a time-series of normal distribution returns and compute the average as the estimated expected return $\bar{\alpha}$. Meanwhile, we compute the covariance of the time series as the estimated covariance matrix and use it as the estimation error matrix Σ.

```
load('data2006.Rdata') # inlcude three variable: expectedReturn, stdDev, corrMat
(expectedReturn)
(stdDev)
(corrMat)


# compute the true covariance matrix
trueCovMat <- diag(stdDev) %*% corrMat %*% diag(stdDev)


# simulate time series of returns
samples <- mvrnorm(100, expectedReturn, trueCovMat)
# compute the estimated expected return and estimated covariance matrix
estimatedReturn <- apply(samples, 2, mean)
estimatedCovMat <- cov(samples)
```

1.3 Robust Portfolio Optimization

In this section, we provide the R code for computing the optimal robust portfolio. We first instantiate the optimization objective, *weight*, by *Variable()* function. The one necessary argument on *Variable()* function is an integer to specify the length of the objective vector. Then we set the adjusted estimated return of portfolio by below formula where κ is simply set to be 1

$$\bar{\alpha}^T w - \kappa \left\| \Sigma^{1/2} w \right\|.$$

Note that 2-norm of $\Sigma^{1/2}w$ can be computed by base function *norm()* and set the second argument to be '2'. The variance of the portfolio is set by *quad_form* function by setting the first argument as *weight* variable and the second as true covariance matrix.

```
# initiate demand variable
n <- length(expectedReturn) #8
weight <- Variable(n)


kappa <- 1


# initiate robust return variable
squaredCovMat <- sqrtm(estimatedCovMat) # squared root of estimation error matrix
ret <- t(estimatedReturn) %*% weight - kappa * norm(squaredCovMat %*% weight, '2')


# initiate risk variable
risk <- quad_form(weight, trueCovMat)
```

1.3.1 Maximum Return Objective

The optimization problem we want to solve in this part is

$$\begin{aligned} \max_{w} \quad & \bar{\alpha}^T w - \kappa \left\| \Sigma^{1/2} w \right\| \\ \text{s.t.} \quad & e^T w = 1 \\ & w^T Q w \leq \nu \\ & w \geq 0 \end{aligned} \tag{4}$$

We can see here the adjusted estimated expect return is the objective we want to optimize over *weight*. Hence we create a new *objective* variable equal to it. Constraints of the optimization problem can be set by a list containing Boolean variables.

1. Full investment constraint is set by *sum(weight)==1*.
2. Long-only constraint is set by *weight >= 0*.
3. Limited risk constraint is set by *risk <=* ν. In this example, we set ν to be 0.1.

We then use class *Problem()* to set up the objective function. The first argument of *Problem()* should be a Maximization class with the objective being its argument and the second is constraints. The last step is

using *solve()* function, where we specify the solver to be SCS.

```
# set objective
objective <- ret

# set constraints of budget, long-only and limited risk
constraints <- list(sum(weight) == 1, weight >= 0, risk <= 0.1)

# set the problem and solve it with SCS solver
prob <- Problem(Maximize(objective), constraints)
result <- solve(prob, solver='SCS')

# get optimal weight
result$getValue(weight)
# get return
t(estimatedReturn) %*% result$getValue(weight)
```

1.3.2 Maximum Utility Objective

In this part, we show another form of Robust Portfolio Optimization where it maximizes the utility of portfolio:

$$\begin{aligned} \max_{w} \quad & \bar{\boldsymbol{\alpha}}^{\boldsymbol{T}} w - \boldsymbol{\kappa} \left\| \Sigma^{1/2} w \right\| - \frac{p}{2} w^T Q w \\ \text{s.t.} \quad & e^T w = 1 \\ & w \geq 0 \end{aligned} \tag{5}$$

We simply set the risk aversion parameter p to be 1. Constraints change to only full investment constraint and long-only constraint.

```
# set objective
riskAversion <- 1
objective <- ret - riskAversion * risk

# set constraints of full investment, long-only
constraints <- list(sum(weight) == 1, weight >= 0)
```

```
# set the problem and solve it with SCS solver
prob <- Problem(Maximize(objective), constraints)
result <- solve(prob, solver='SCS')


# get optimal weight
result$getValue(weight)
# get return
t(estimatedReturn) %*% result$getValue(weight)
```

1.3.3 Minimum Risk Objective

The third form of Robust Portfolio Optimization is to minimize the variance of the portfolio given a minimum target return constraint:

$$\begin{aligned} \min_{w} \quad & w^T Q w \\ \text{s.t.} \quad & e^T w = 1 \\ & \bar{\boldsymbol{\alpha}}^{\boldsymbol{T}} w - \boldsymbol{\kappa} \left\| \Sigma^{1/2} w \right\| \geq r \\ & w \geq 0 \end{aligned} \tag{6}$$

Compared to constraints in 1.3.2, we add a target return constraint by *ret >= μ*. Also, since we want to minimize variance, the first argument of *Problem()* class should be a *Minimization()* class.

```
# set objective
objective <-  risk


# set constraints of target return, full investment and long-only
constraints <- list(sum(weight) == 1, weight >= 0, ret >= -0.02)


# set the problem and solve it with SCS solver
prob <- Problem(Minimize(objective), constraints)
result <- solve(prob, solver='SCS')


# get optimal weight
result$getValue(weight)
# get return
```

```
t(estimatedReturn) %*% result$getValue(weight)
```

1.4 Variants related to Robust Portfolio Optimization

In this section, we discuss two variants of Robust Portfolio Optimization.

1.4.1 Maximum Active Return

Active portfolio managers are more interested in the expected values of active returns and active risks. To give a optimization problem seeking active returns and risks, we transform the original Robust Portfolio Optimization into below form:

$$\begin{aligned} \max_{w} \quad & \bar{\boldsymbol{\alpha}}^{\boldsymbol{T}}(w-b) - \boldsymbol{\kappa}\left\|\Sigma^{1/2}(w-b)\right\| \\ \text{s.t.} \quad & e^T w = 1 \\ & (w-b)^T Q(w-b) \leq \nu \\ & w \geq 0 \end{aligned} \tag{7}$$

where b is the benchmark weight. In our below example, we set the benchmark weight to be a equal weight.

```
squaredCovMat <- sqrtm(estimatedCovMat)

# initiate variable
weight <- Variable(n)
benchmarkWeight <- rep(1/n, n)

# update adjusted return and variance
ret <- t(estimatedReturn) %*% (weight - benchmarkWeight)
      - kappa * norm(squaredCovMat %*% (weight - benchmarkWeight), '2')
risk <- quad_form((weight - benchmarkWeight), trueCovMat)

# set objective
objective <- ret

# set constraints of full investment, long-only and limited risk
```

```
constraints <- list(sum(weight) == 1, weight >= 0, risk <= 0.1)


# solve the optimization problem with SCS solver
prob <- Problem(Maximize(objective), constraints)
result <- solve(prob, solver='SCS')


# get optimal weight
result$getValue(weight)
# get return
t(estimatedReturn) %*% result$getValue(weight)
```

1.4.2 Robust Portfolio Optimization with Zero Net Alpha

The previous Robust Portfolio Optimization seems too conservative as the net adjustment towards estimated expected return is always downwards. However, one should expect there are approximately as many true expected returns larger than the corresponding estimates as there are smaller than the estimates. In order to build this expectation into our model, we add one linear constraint to problem(2):

$$e^T D(\alpha - \bar{\alpha}) = 0.$$

where D is some symmetric invertible matrix. Then problem(2) would become

$$\begin{aligned} \max_{\alpha} \quad & \bar{\alpha}^T w - \alpha^T w \\ \text{s.t.} \quad & (\alpha - \bar{\alpha})^T \Sigma^{-1} (\alpha - \bar{\alpha}) \leq \kappa^2 \\ & e^T D(\alpha - \bar{\alpha}) = 0 \end{aligned} \tag{8}$$

The solution to (8) is

$$\alpha = \bar{\alpha} - \sqrt{\frac{\kappa^2}{(\Sigma w - \frac{e^T D\Sigma w}{e^T D\Sigma D^T e}\Sigma D^T e)^T \Sigma^{-1} (\Sigma w - \frac{e^T D\Sigma w}{e^T D\Sigma D^T e}\Sigma D^T e)}} (\Sigma w - \frac{e^T D\Sigma w}{e^T D\Sigma D^T e}\Sigma D^T e)$$

Furthermore we have

$$\alpha^T w = \bar{\alpha}^T w - \kappa \left\| (\Sigma - \frac{1}{e^T D\Sigma D^T e}\Sigma D^T e e^T D\Sigma)^{1/2} w \right\|.$$

Then the Robust Portfolio Optimization with zero net alpha can be written as:

$$\begin{aligned} \max_{w} \quad & \bar{\alpha}^T w - \kappa \left\| (\Sigma - \frac{1}{e^T D \Sigma D^T e} \Sigma D^T e e^T D \Sigma)^{1/2} w \right\| \\ \text{s.t.} \quad & e^T w = 1 \\ & w^T Q w \leq \nu \\ & w \geq 0 \end{aligned} \tag{9}$$

In our example, we set $D = \Sigma^{-1}$.

```
e <- rep(1, n)
D <- solve(estimatedCovMat)

# initiate variable
weight <- Variable(n)

# update adjusted return and variance
params <- matrix(1 / (t(e) %*% D %*% estimatedCovMat %*% t(D) %*% e), n, n)
adjustedSquaredCovMat <- sqrtm(estimatedCovMat
                  -  params * (estimatedCovMat %*% t(D) %*% e %*% t(e) %*% D %*% estimatedCovMat))



ret <- t(estimatedReturn) %*% weight - kappa * norm(adjustedSquaredCovMat %*% weight, '2')
risk <- quad_form(weight, trueCovMat)

# set objective
objective <- ret

# set constraints of full investment, long-only and limited risk
constraints <- list(sum(weight) == 1, weight >= 0, risk <= 0.1)

# solve the optimization problem with SCS solver
prob <- Problem(Maximize(objective), constraints)
result <- solve(prob, solver='SCS')
```

```
# get optimal weights
result$getValue(weight)
# get return
t(estimatedReturn) %*% result$getValue(weight)
```

2.1 Numerical Example

In this part, we use two different sets of data to plot efficient frontiers including true frontier, estimated frontier, actual frontier and robust actual frontier. True frontier is plotted with true expected return and true covariance matrix, while estimated frontier is plotted with estimated expected return and true covariance matrix. The actual frontier is plotted by multiplying weights from estimated frontier by true expected return. Last, the robust actual frontier is plotted by multiplying weights from robust portfolio optimization by true expected return.

2.1.1 Frontiers Example 1

In the first example, we still employ the previous true expected return and true covariance matrix. There are two main functions we create. The first function is *getFrontiers(n, target, retVec, covMat)*. It computes the optimal weight under classical Markowitz Mean-Variance Optimization (1). The four arguments specify the length of optimization objective, the largest variance of the portfolio, the expected return vector and covariance matrix by order. It returns a vector with first digit to be the maximized expected return of portfolio and the other digits to be the optimal weights.

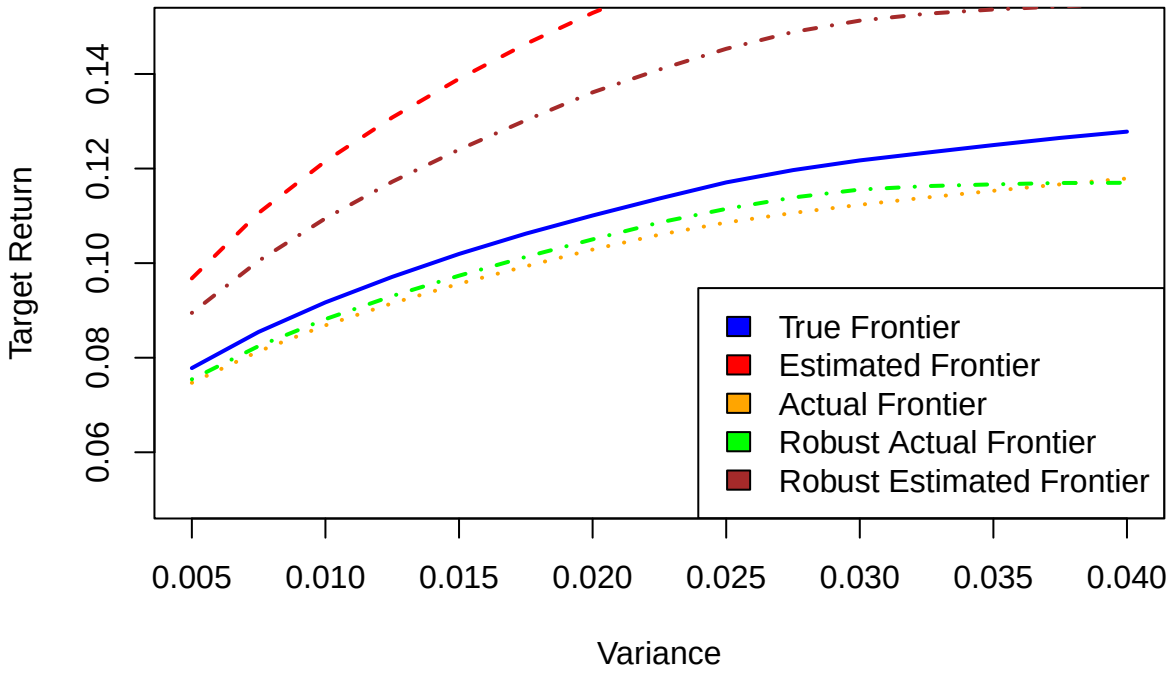
The second function is *getRobustFrontiers(n, target, retVec, covMat, estimatedCovMat, kappa=1)*. It computes the optimal weight under Robust Portfolio Optimization which maximizes expected return of portfolio. The six arguments specify the length of optimization objective, the largest variance of the portfolio, the estimated expected return vector, true covariance matrix, estimated covariance matrix and kappa by order. It returns a vector with first digit to be the maximized expected return of portfolio and the other digits to be the optimal weights. Particularly, this function uses the Robust Portfolio Optimization with zero net alpha setting. Matrix D is preset as an identity matrix.

Using two functions above, we calculate optimal expected return of portfolio with respect to each variance point within possible range. Here we set the variance range to be (0.005, 0.04) with interval 0.001. For each frontier, we use the average of 10 trails as the finalized frontier.

```r
# function to get classical Markowitz weights
getFrontiers <- function(n, target, retVec, covMat){
  w <- Variable(n)
  ret <- t(retVec) %*% w
  risk <- quad_form(w, covMat)

  constraints <- list(sum(w) == 1, w >= 0, risk <= target)

  objective <- ret
  prob <- Problem(Maximize(objective), constraints)
  result <- solve(prob, solver='SCS')
  return(c(result$getValue(ret), result$getValue(w)))
}

getRobustFrontiers <- function(n, target, retVec, covMat, estimatedCovMat, kappa=1){
  e <- rep(1, n)
  D <- diag(e)

  param <- matrix(1 / (t(e) %*% D %*% estimatedCovMat %*% t(D) %*% e), n, n)
  adjustedSquaredCovMat <- sqrtm(estimatedCovMat
                 -  param * (estimatedCovMat %*% t(D) %*% e %*% t(e) %*% D %*% estimatedCovMat))

  w <- Variable(n)
  ret <- t(retVec) %*% w - kappa * norm(adjustedSquaredCovMat %*% w, '2')
  risk <- quad_form(w, covMat)

  constraints <- list(sum(w) == 1, w >= 0, risk <= target)

  objective <- ret
  prob <- Problem(Maximize(objective), constraints)
  result <- solve(prob, solver='SCS')
  return(c(t(retVec) %*% result$getValue(w), result$getValue(w)))
```

```
}

vars <- seq(0.005, 0.04, 0.0025)

trueFrontier <- c()
estimatedFrontier <- rep(0, length(vars))
actualFrontier <- rep(0, length(vars))
robustActualFrontier <- rep(0, length(vars))
robustEstimatedFrontier <- rep(0, length(vars))



for (variance in vars){
  trueFrontier <- c(trueFrontier, getFrontiers(n, variance, expectedReturn, trueCovMat)[1])
}

# using the average of 10 trails as frontier
for (i in 1:10){
  samples <- mvrnorm(10, expectedReturn, trueCovMat)
  estimatedReturn <- apply(samples, 2, mean)
  estimatedCovMat <- cov(samples)

  singleEstimatedFrontier <- c()
  singleActualFrontier <- c()
  singleRobustActualFrontier <- c()
  singleRobustEstimatedFrontier <- c()

  for (variance in vars){
    tmp <- getFrontiers(n, variance, estimatedReturn, trueCovMat)
    singleEstimatedFrontier <- c(singleEstimatedFrontier, tmp[1])

    weights <- tmp[2:length(tmp)]
    singleActualFrontier <- c(singleActualFrontier, t(weights) %*% expectedReturn)
```

```
    tmp2 <- getRobustFrontiers(n, variance, estimatedReturn, trueCovMat, estimatedCovMat, 1)
    singleRobustEstimatedFrontier <- c(singleRobustEstimatedFrontier, tmp2[1])

    robustWeights <- tmp2[2:length(tmp)]
    singleRobustActualFrontier <- c(singleRobustActualFrontier, t(robustWeights) %*% expectedReturn)
  }
  estimatedFrontier <- estimatedFrontier + singleEstimatedFrontier / 10
  actualFrontier <- actualFrontier + singleActualFrontier / 10
  robustActualFrontier <- robustActualFrontier + singleRobustActualFrontier / 10
  robustEstimatedFrontier <- robustEstimatedFrontier + singleRobustEstimatedFrontier / 10
}

plot(vars, trueFrontier, type='l', xlab='Variance', ylab='Target Return', ylim=c(0.05, 0.15),
       col='blue', lty=1, lwd=2)
lines(vars, estimatedFrontier, col='red', lty=2, lwd=2)
lines(vars, actualFrontier, col='orange', lty=3, lwd=2)
lines(vars, robustActualFrontier, col='green', lty=4, lwd=2)
lines(vars, robustEstimatedFrontier, col='brown', lty=4, lwd=2)

legend("bottomright", legend=c("True Frontier", 'Estimated Frontier', 'Actual Frontier', 'Robust Actual
       fill=c('blue', 'red', 'orange', 'green', 'brown'))
```



2.1.2 Frontiers Example 2

The second dataset is provided in Burak Kicuk (2020). The data set includes 360 monthly returns of 11 sectors based on the Global Industrial Classification Standard. We calculate the true return vector μ and the true covariance matrix Σ as the sample average and the sample covariance matrix of these returns. We refer the reader to Burak Kicuk (2020) for more details about the dataset.

The process of generating frontiers is the same as previous example. However, this time we change the function *getRobustFrontiers()* by using standard Robust Portfolio Optimization without zero net alpha adjustment and using another estimation error matrix of a diagonal matrix.

```
data <- readxl::read_xlsx('data2020.xlsx', sheet = 'Returns', col_names = FALSE)
```

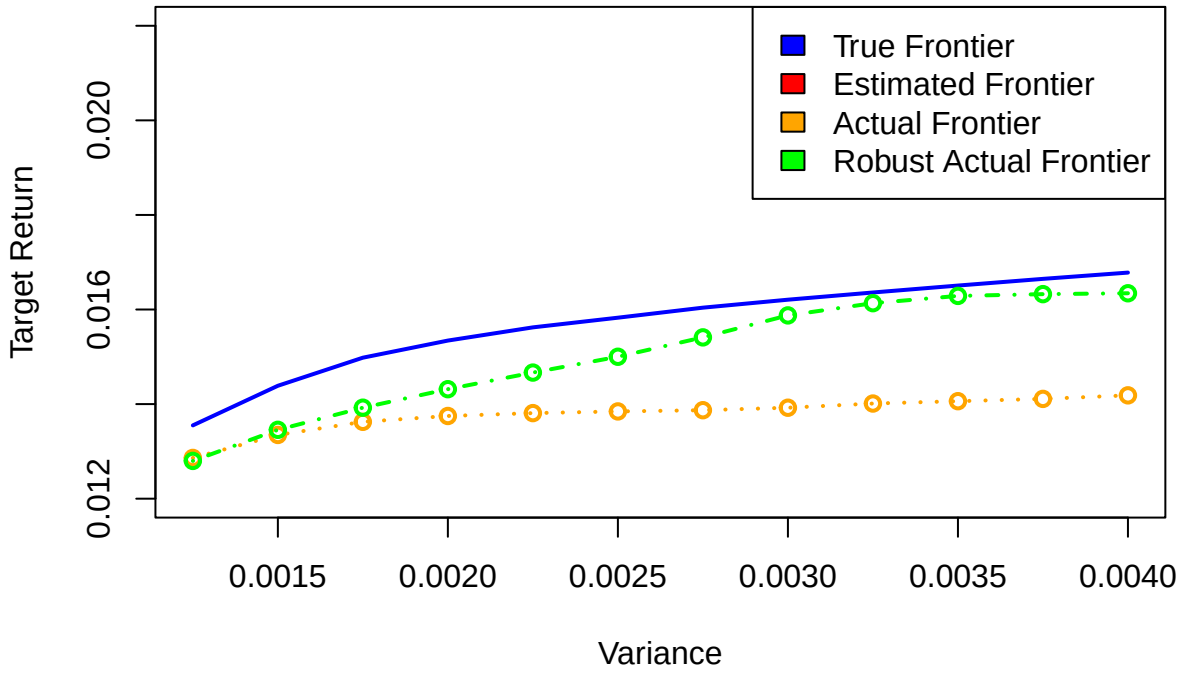
```
## New names:
## * `` -> `...1`
## * `` -> `...2`
## * `` -> `...3`
## * `` -> `...4`
## * `` -> `...5`
## * `` -> `...6`
## * `` -> `...7`
```

```
## * `` -> `...8`
## * `` -> `...9`
## * `` -> `...10`
## * `` -> `...11`
```

```
expectedReturn <- apply(data, 2, mean)
trueCovMat <- cov(data)

m <- length(expectedReturn)

getRobustFrontiers <- function(n, target, retVec, covMat, kappa=1){

  u <- sqrtm(diag(c(1, 0.001, 1, 1, 1, 1, 1, 0.001, 1, 1, 1)))

  w <- Variable(n)
  ret <- t(retVec) %*% w - kappa * norm(u %*% w, '2')
  risk <- quad_form(w, covMat)

  constraints <- list(sum(w) == 1, w >= 0, risk <= target)

  objective <- ret
  prob <- Problem(Maximize(objective), constraints)
  result <- solve(prob, solver='SCS')
  return(c(t(retVec) %*% result$getValue(w), result$getValue(w)))
}

vars <- seq(0.00125, 0.004, 0.00025)

trueFrontier <- c()
estimatedFrontier <- rep(0, length(vars))
actualFrontier <- rep(0, length(vars))
robustFrontier <- rep(0, length(vars))
```

```
for (variance in vars){
  trueFrontier <- c(trueFrontier, getFrontiers(m, variance, expectedReturn, trueCovMat)[1])
}

for (i in 1:10){
  samples <- mvrnorm(10, expectedReturn, trueCovMat)
  estimatedReturn <- apply(samples, 2, mean)
  estimatedCovMat <- cov(samples)

  singleEstimatedFrontier <- c()
  singleActualFrontier <- c()
  singleRobustFrontier <- c()

  for (variance in vars){
    tmp <- getFrontiers(m, variance, estimatedReturn, trueCovMat)
    singleEstimatedFrontier <- c(singleEstimatedFrontier, tmp[1])

    weights <- tmp[2:length(tmp)]
    singleActualFrontier <- c(singleActualFrontier, t(weights) %*% expectedReturn)

    tmp2 <- getRobustFrontiers(m, variance, estimatedReturn, trueCovMat, 1)
    robustWeights <- tmp2[2:length(tmp)]
    singleRobustFrontier <- c(singleRobustFrontier, t(robustWeights) %*% expectedReturn)
  }
  estimatedFrontier <- estimatedFrontier + singleEstimatedFrontier / 10
  actualFrontier <- actualFrontier + singleActualFrontier / 10
  robustFrontier <- robustFrontier + singleRobustFrontier / 10
}

plot(vars, trueFrontier, type='l', xlab='Variance', ylab='Target Return', ylim=c(0.012, 0.022),
       col='blue', lty=1, lwd=2)
lines(vars, estimatedFrontier, col='red', type='o', lty=2, lwd=2)
```

```
lines(vars, actualFrontier, col='orange', type='o', lty=3, lwd=2)
lines(vars, robustFrontier, col='green', type='o', lty=4, lwd=2)


legend("topright", legend=c("True Frontier", 'Estimated Frontier', 'Actual Frontier', 'Robust Actual Fr
       fill=c('blue', 'red', 'orange', 'green'))
```



Reference

Burak Kicuk, Gerard Cornuejols. 2020. "Incorporating Black-Litterman Views in Portfolio Construction When Stock Returns Are a Mixture of Normals."

Ceria, Sebastián, and Robert Stubbs. 2006. "Incorporating Estimation Errors into Portfolio Selection: Robust Portfolio Construction." *Journal of Asset Management* 7 (April).

Fischer Black, Robert B Litterman. 1990. "Asset Allocation: Combining Investors' Views with Market Equilibrium." *Fixed Income Research*. Goldman Sachs.

J. D. Jobson, Robert M Korkie. 1981. "Putting Markowitz Theory to Work." *Journal of Portfolio Management* 7: 70–74.

Tom Idzorek, Jill Adrogue. 2003. "Using Black-Litterman Return Forecasts for Asset Allocation Results in Diversified Portfolios."