Custom Moment and Objective Functions

Ross Bennett

August 4, 2014

Abstract

The purpose of this vignette is to demonstrate how to write and use custom moment functions and custom objective functions.

Contents

1 Getting Started

1.1 Load Packages

Load the necessary packages.

```
library(PortfolioAnalytics)

## Loading required package:  zoo
##
## Attaching package:  'zoo'
##
## The following objects are masked from 'package:base':
```

```
##
##     as.Date, as.Date.numeric
##
## Loading required package:  xts
## Loading required package:  PerformanceAnalytics
##
## Attaching package:  'PerformanceAnalytics'
##
## The following object is masked from 'package:graphics':
##
##     legend
```

1.2 Data

The edhec data set from the PerformanceAnalytics package will be used as example data.

```
data(edhec)

# Use the first 4 columns in edhec for a returns object
R <- edhec[, 1:4]
colnames(R) <- c("CA", "CTAG", "DS", "EM")
head(R, 5)

##                CA    CTAG      DS      EM
## 1997-01-31 0.0119  0.0393  0.0178  0.0791
## 1997-02-28 0.0123  0.0298  0.0122  0.0525
## 1997-03-31 0.0078 -0.0021 -0.0012 -0.0120
## 1997-04-30 0.0086 -0.0170  0.0030  0.0119
## 1997-05-31 0.0156 -0.0015  0.0233  0.0315

# Get a character vector of the fund names
funds <- colnames(R)
```

2 Setting the Portfolio Moments

The PortfolioAnalytics framework to estimate solutions to constrained optimization problems is implemented in such a way that the moments of the returns are calculated only once and then used in lower level optimization functions. The `set.portfolio.moments` function computes the first, second, third, and fourth moments depending on the objective function(s) in the `portfolio` object. `set.portfolio.moments` implements methods to compute moments based on sample estimates, higher moments from fitting a statistical factor model based on the work of Kris Boudt (NEED REFERENCE HERE), the Black Litterman model, and the Fully Flexible Framework based on the work of Attilio Meucci.

The moments of the returns are computed based on the objective(s) in the `portfolio` object and return a list where each element is the respective moment estimate.

```r
args(set.portfolio.moments)

## function (R, portfolio, momentargs = NULL, method = c("sample",
##     "boudt", "black_litterman", "meucci"), ...)
## NULL
```

```r
# Construct initial portfolio with basic constraints.
init.portf <- portfolio.spec(assets=funds)
init.portf <- add.constraint(portfolio=init.portf, type="full_investment")
init.portf <- add.constraint(portfolio=init.portf, type="long_only")

# Portfolio with standard deviation as an objective
SD.portf <- add.objective(portfolio=init.portf, type="risk", name="StdDev")

# Portfolio with expected shortfall as an objective
ES.portf <- add.objective(portfolio=init.portf, type="risk", name="ES")
```

Here we see the names of the list object that is returned by `set.portfolio.moments`.

```r
sd.moments <- set.portfolio.moments(R, SD.portf)
names(sd.moments)

## [1] "mu"    "sigma"

es.moments <- set.portfolio.moments(R, ES.portf)
```

```
names(es.moments)

## [1] "mu"    "sigma" "m3"    "m4"
```

3 Custom Moment Functions

In many cases for constrained optimization problems, one may want to estimate moments for a specific use case or further extend the idea of `set.portfolio.moments`. A user defined custom moment function can have any arbitrary named arguments. However, arguments named `R` for the asset returns and `portfolio` for the portfolio object will be detected automatically and handled in an efficient manner. Because of this, it is strongly encouraged to use `R` for the asset returns object and `portfolio` for the portfolio object.

The moment function should return a named list object where the elements represent the moments:

`$mu` first moment; expected returns vector

`$sigma` second moment; covariance matrix

`$m3` third moment; coskewness matrix

`$m4` fourth moment; cokurtosis matrix

The lower level optimization functions expect an object with the structure described above. List elements with the names `mu`, `sigma`, `m3`, and`m4` are matched automatically and handled in an efficient manner.

Here we define a function to estimate the covariance matrix using a robust method.

```
sigma.robust <- function(R, ...){
  out <- list()
  set.seed(1234)
  out$sigma <- MASS::cov.rob(R, method="mcd", ...)$cov
  return(out)
}
```

Now we can use the custom moment function in `optimize.portfolio` to estimate the solution to the minimum standard deviation portfolio.

```
opt.sd <- optimize.portfolio(R, SD.portf,
                             optimize_method="ROI",
                             momentFUN="sigma.robust")

## ROI: R Optimization Infrastructure
## Registered solver plugins:  cplex, glpk, quadprog, symphony.
## Default solver:  ROI_NULL.
##
## Attaching package:  'ROI'
##
## The following object is masked from 'package:PortfolioAnalytics':
##
##    objective

opt.sd

## ***********************************
## PortfolioAnalytics Optimization
## ***********************************
##
## Call:
## optimize.portfolio(R = R, portfolio = SD.portf, optimize_method = "ROI",
##     momentFUN = "sigma.robust")
##
## Optimal Weights:
##     CA   CTAG     DS     EM
## 0.6598 0.1441 0.1961 0.0000
##
## Objective Measure:
##   StdDev
## 0.008646
```

Here we extract the weights and compute the portfolio standard deviation to verify that the the robust estimate of the covariance matrix was used in the optimization.

```
weights <- extractWeights(opt.sd)
sigma <- sigma.robust(R)$sigma

sqrt(t(weights) %*% sigma %*% weights)

##          [,1]
## [1,] 0.008646

extractObjectiveMeasures(opt.sd)$StdDev

##   StdDev
## 0.008646
```

4 Custom Objective Functions

A key feature of `PortfolioAnalytics` is that the name for an objective can be any valid `R` function. `PortfolioAnalytics` was designed to be flexible and modular, and custom objective functions are a key example of this.

Here we define a very simple function to compute annualized standard deviation for monthly data that we will use as an objective function.

```
pasd <- function(R, weights, sigma, N=36){
  R <- tail(R, N)
  tmp.sd <- sqrt(as.numeric(t(weights) %*% sigma %*% weights))
  sqrt(12) * tmp.sd
}
```

A few guidelines should be followed for defining a custom objective function.

- The objective function must return a single value for the optimizer to minimize.
- It is strongly encouraged to use the following argument names in the objective function:

  `R` for the asset returns

  `weights` for the portfolio weights

These argument names are detected automatically and handled in an efficient manner. Any other arguments for the objective function can be for the moments or passed in through the `arguments` list in the objective.

For our `pasd` function, we need custom moments function to return a named list with `sigma` as an element. We can use the `sigma.robust` function we defined in the previous section. Here we construct a portfolio with our `pasd` function as an objective to minimize.

```r
# Construct initial portfolio with basic constraints.
pasd.portf <- portfolio.spec(assets=funds)
pasd.portf <- add.constraint(portfolio=pasd.portf, type="full_investment")
pasd.portf <- add.constraint(portfolio=pasd.portf, type="long_only")

# Portfolio with pasd as an objective
# Note how we can specify N as an argument
pasd.portf <- add.objective(portfolio=pasd.portf, type="risk", name="pasd",
                            arguments=list(N=48))
```

Now we can run the optimization to estimate a solution to our optimization problem.

```r
opt.pasd <- optimize.portfolio(R, pa.portf,
                               optimize_method="ROI",
                               momentFUN="sigma.robust")

## Error:  object 'pa.portf' not found

opt.pasd

## Error:  object 'opt.pasd' not found
```

We now consider an example with a more complicated objective function. Our objective to maximize the fourth order expansion of the Constant Relative Risk Aversion (CRRA) expected utility function as in the Boudt paper and Martellini paper (NEED REFERENCE).

$$EU_{\lambda}(w) = -\frac{\lambda}{2} m_{(2)}(w) + \frac{\lambda(\lambda+1)}{6} m_{(3)}(w) - \frac{\lambda(\lambda+1)(\lambda+2)}{24} m_{(4)}(w)$$

Define a function to compute CRRA estimate. Note how we define the function to use `sigma`, `m3`, and `m4` as arguments that will use the output from a custom moment function. We could compute the moments inside this function, but re-computing the moments thousands of times (i.e. at each iteration) can be very compute intensive.

```r
CRRA <- function(R, weights, lambda, sigma, m3, m4){
  weights <- matrix(weights, ncol=1)
  M2.w <- t(weights) %*% sigma %*% weights
  M3.w <- t(weights) %*% m3 %*% (weights %x% weights)
  M4.w <- t(weights) %*% m4 %*% (weights %x% weights %x% weights)
  term1 <- (1 / 2) * lambda * M2.w
  term2 <- (1 / 6) * lambda * (lambda + 1) * M3.w
  term3 <- (1 / 24) * lambda * (lambda + 1) * (lambda + 2) * M4.w
  out <- -term1 + term2 - term3
  out
}
```

We now define the custom moment function to compute the moments for the objective function.

```r
crra.moments <- function(R, ...){
  out <- list()
  out$sigma <- cov(R)
  out$m3 <- PerformanceAnalytics:::M3.MM(R)
  out$m4 <- PerformanceAnalytics:::M4.MM(R)
  out
}
```

We now set up the portfolio and run the optimization using our custom moments and objective function to maximize CRRA. Note that `type="return"` is used to maximize an objective function.

```r
# Construct initial portfolio with basic constraints.
crra.portf <- portfolio.spec(assets=funds)
crra.portf <- add.constraint(portfolio=crra.portf, type="weight_sum",
                             min_sum=0.99, max_sum=1.01)
crra.portf <- add.constraint(portfolio=crra.portf, type="box",
                             min=0.05, max=0.4)

# Portfolio with crra as an objective
# Note how we can specify lambda as an argument
```

```
crra.portf <- add.objective(portfolio=crra.portf, type="return", name="CRRA",
                            arguments=list(lambda=10))
```

```
opt.crra <- optimize.portfolio(R, crra.portf, optimize_method="DEoptim",
                                 search_size=5000, trace=TRUE, traceDE=0,
                                 momentFUN="crra.moments")

##
## DEoptim package
## Differential Evolution algorithm in R
## Authors:  D. Ardia, K. Mullen, B. Peterson and J. Ulrich

## [1] 0.196 0.384 0.366 0.052

opt.crra

## ***********************************
## PortfolioAnalytics Optimization
## ***********************************
##
## Call:
## optimize.portfolio(R = R, portfolio = crra.portf, optimize_method = "DEoptim",
##     search_size = 5000, trace = TRUE, traceDE = 0, momentFUN = "crra.moments")
##
## Optimal Weights:
##    CA  CTAG    DS    EM
## 0.196 0.384 0.366 0.052
##
## Objective Measures:
##      CRRA
## -0.001087
```

The modular framework of `PortfolioAnalytics` allows one to easily define custom moment functions and objective functions as valid Rfunctions.

TODO: add content to concluding paragraph