Vignette: Portfolio Optimization with CVaR budgets in PortfolioAnalytics

Kris Boudt, Peter Carl and Brian Peterson

June 1, 2010

Contents

1 General information

Risk budgets are a central tool to estimate and manage the portfolio risk allocation. They decompose total portfolio risk into the risk contribution of each position. Boudt et˜al. (2010a) propose several portfolio allocation strategies that use an appropriate transformation of the portfolio Conditional Value at Risk (CVaR) budget as an objective or constraint in the portfolio optimization problem. This document explains how risk allocation optimized portfolios can be obtained under general constraints in the `PortfolioAnalytics` package of Boudt et˜al. (2010b).

`PortfolioAnalytics` is designed to provide numerical solutions for portfolio problems with complex constraints and objective sets comprised of any R function. It can e.g.˜construct portfolios

that minimize a risk objective with (possibly non-linear) per-asset constraints on returns and drawdowns (Carl et˜al., 2010). The generality of possible constraints and objectives is a distinctive characteristic of the package with respect to RMetrics `fPortfolio` of Wuertz et˜al. (2010). For standard Markowitz optimization problems, use of `fPortfolio` rather than `PortfolioAnalytics` is recommended.

`PortfolioAnalytics` solves the following type of problem

$$\min_w g(w) \quad s.t. \quad \begin{cases} h_1(w) \leq 0 \\ \vdots \\ h_q(w) \leq 0. \end{cases} \tag{1}$$

`PortfolioAnalytics` first merges the objective function and constraints into a penalty augmented objective function

$$L(w) = g(w) + \text{penalty} \sum_{i=1}^{q} \lambda_i \max(h_i(w), 0), \tag{2}$$

where λ_i is a multiplier to tune the relative importance of the constraints. The default values of penalty and λ_i (called `multiplier` in `PortfolioAnalytics`) are 10000 and 1, respectively.

The minimum of this function is found through the *Differential Evolution* (DE) algorithm of Storn and Price (1997) and ported to R by Mullen et˜al. (2009). DE is known for remarkable performance regarding continuous numerical problems (Price et˜al., 2006). It has recently been advocated for optimizing portfolios under non-convex settings by Ardia et˜al. (2010) and Yollin (2009), among others. We use the R implementation of DE in the `DEoptim` package of Ardia and Mullen (2009).

The latest version of the `PortfolioAnalytics` package can be downloaded from R-forge through the following command:

```
install.packages("PortfolioAnalytics", repos="http://R-Forge.R-project.org")
```

Its principal functions are:

- `constraint(assets,min,max,min_sum,max_sum)`: the portfolio optimization specification starts with specifying the shape of the weight vector through the function `constraint`. The weights have to be between `min} and \verb`max" and their sum between `min_sum` and `max_sum`. The first argument `assets` is either a number indicating the number of portfolio assets or a vector holding the names of the assets.

- `add.objective(constraints, type, name)`: `constraints` is a list holding the objective to be minimized and the constraints. New elements to this list are added by the function `add.objective`. Many common risk budget objectives and constraints are prespecified and can be identified by specifying the `type` and `name`.

- `constrained_objective(w, R, constraints)`: given the portfolio weight and return data, it evaluates the penalty augmented objective function in (2).
- `optimize.portfolio(R,constraints)`: this function returns the portfolio weight that solves the problem in (1). R is the multivariate return series of the portfolio components.
- `optimize.portfolio.rebalancing(R,constraints,rebalance_on,trailing_periods`: this function solves the multiperiod optimization problem. It returns for each rebalancing period the optimal weights and allows the estimation sample to be either from inception or a moving window.

Next we illustrate these functions on monthly return data for bond, US equity, international equity and commodity indices, which are the first 4 series in the dataset `indexes`. The first step is to load the package `PortfolioAnalytics` and the dataset. An important first note is that some of the functions (especially `optimize.portfolio.rebalancing`) requires the dataset to be a `xts` object (Ryan and Ulrich, 2010).

```
> library(PortfolioAnalytics)
> data(indexes)
> class(indexes)

[1] "xts" "zoo"

> indexes <- indexes[, 1:4]
> head(indexes, 2)

           US Bonds US Equities Int'l Equities Commodities
1980-01-31  -0.0272      0.0610         0.0462      0.0568
1980-02-29  -0.0669      0.0031        -0.0040     -0.0093

> tail(indexes, 2)

           US Bonds US Equities Int'l Equities Commodities
2009-11-30   0.0134      0.0566         0.0199      0.0150
2009-12-31  -0.0175      0.0189         0.0143      0.0086
```

In what follows, we first illustrate the construction of the penalty augmented objective function. Then we present the code for solving the optimization problem.

2 Setting of the objective function

2.1 Weight constraints

```
> Wcons <- constraint(assets = colnames(indexes[, 1:4]), min = rep(0,
+     4), max = rep(1, 4), min_sum = 1, max_sum = 1)
```

Given the weight constraints, we can call the value of the function to be minimized. We consider the case of no violation and a case of violation. By default, `normalize=TRUE` which means that if the sum of weights exceeds `max_sum`, the weight vector is normalized by multiplying it with `sum(weights)/max_sum` such that the weights evaluated in the objective function satisfy the `max_sum` constraint.

```
> constrained_objective(w = rep(1/4, 4), R = indexes[, 1:4], constraints = Wcons)

[1] 0

> constrained_objective(w = rep(1/3, 4), R = indexes[, 1:4], constraints = Wcons)

[1] 0

> constrained_objective(w = rep(1/3, 4), R = indexes[, 1:4], constraints = Wcons,
+     normalize = FALSE)

[1] 3333.333
```

The latter value can be recalculated as penalty times the weight violation, that is: $10000 \times 1/3$.

2.2 Minimum CVaR objective function

Suppose now we want to find the portfolio that minimizes the 95% portfolio CVaR subject to the weight constraints listed above.

```
> ObjSpec = add.objective(constraints = Wcons, type = "risk", name = "CVaR",
+     arguments = list(p = 0.95), enabled = TRUE)
```

The value of the objective function is:

```
> constrained_objective(w = rep(1/4, 4), R = indexes[, 1:4], constraints = ObjSpec)

       [,1]
ES 0.1253199
```

This is the CVaR of the equal-weight portfolio as computed by the function ES in the PerformanceAnalytics package of Carl and Peterson (2009)

```
> library(PerformanceAnalytics)
> out <- ES(indexes[, 1:4], weights = rep(1/4, 4), p = 0.95, portfolio_method = "component")
> out$MES

          [,1]
[1,] 0.1253199
```

All arguments in the function ES can be passed on through arguments. E.g. to reduce the impact of extremes on the portfolio results, it is recommended to winsorize the data using the option clean="boudt".

```
> out <- ES(indexes[, 1:4], weights = rep(1/4, 4), p = 0.95, clean = "boudt",
+     portfolio_method = "component")
> out$MES

           [,1]
[1,] 0.07124999
```

For the formulation of the objective function, this implies setting:

```
> ObjSpec = add.objective(constraints = Wcons, type = "risk", name = "CVaR",
+     arguments = list(p = 0.95, clean = "boudt"), enabled = TRUE)
> constrained_objective(w = rep(1/4, 4), R = indexes[, 1:4], constraints = ObjSpec)

         [,1]
ES 0.07124999
```

An additional argument that is not available for the moment in ES is to estimate the conditional covariance matrix trough the constant conditional correlation model of Bollerslev (1990).

For the formulation of the objective function, this implies setting:

```
> ObjSpec = add.objective(constraints = Wcons, type = "risk", name = "CVaR",
+     arguments = list(p = 0.95, clean = "boudt", garch = TRUE),
+     enabled = TRUE)
> constrained_objective(w = rep(1/4, 4), R = indexes[, 1:4], constraints = ObjSpec)

         [,1]
ES 0.08381728
```

2.3 Minimum CVaR concentration objective function

Add the minimum 95% CVaR concentration objective to the objective function:

```
> ObjSpec = add.objective(constraints = Wcons, type = "risk_budget_objective",
+     name = "CVaR", arguments = list(p = 0.95, clean = "boudt"),
+     min_concentration = TRUE, enabled = TRUE)
```

The value of the objective function is:

```
> constrained_objective(w = rep(1/4, 4), R = indexes[, 1:4], constraints = ObjSpec)
```

```
[1] 0.02527130
```

We can verify that this is effectively the largest CVaR contribution of that portfolio as follows:

```
> ES(indexes[, 1:4], weights = rep(1/4, 4), p = 0.95, clean = "boudt",
+     portfolio_method = "component")
```

```
$MES
           [,1]
[1,] 0.07124999


$contribution
      US Bonds    US Equities Int'l Equities    Commodities
   0.000593884    0.020748329    0.024636472    0.025271304


$pct_contrib_MES
      US Bonds    US Equities Int'l Equities    Commodities
   0.008335215    0.291204659    0.345775103    0.354685023
```

2.4 Risk allocation constraints

We see that in the equal-weight portfolio, the international equities and commodities investment cause more than 30% of total risk. We could specify as a constraint that no asset can contribute more than 30% to total portfolio risk. This involves the construction of the following objective function:

```
> ObjSpec = add.objective(constraints = Wcons, type = "risk_budget_objective",
+     name = "CVaR", max_prisk = 0.3, arguments = list(p = 0.95,
+         clean = "boudt"), enabled = TRUE)
> constrained_objective(w = rep(1/4, 4), R = indexes[, 1:4], constraints = ObjSpec)
```

```
[1] 1004.601
```

This value corresponds to the penalty parameter which has by default the value of 10000 times the exceedances: $10000 * (0.045775103 + 0.054685023) \approx 1004.601$.

3 Optimization

The penalty augmented objective function is minimized through Differential Evolution. Two parameters are crucial in tuning the optimization: `search_size` and `itermax`. The optimization routine

1. First creates the initial generation of `NP= search_size/itermax` guesses for the optimal value of the parameter vector, using the `random_portfolios` function generating random weights satisfying the weight constraints.

2. Then DE evolves over this population of candidate solutions using alteration and selection operators in order to minimize the objective function. It restarts `itermax` times.

It is important that `search_size/itermax` is high enough. It is generally recommended that this ratio is at least ten times the length of the weight vector. For more details on the use of DE strategy in portfolio allocation, we refer the reader to Ardia et˜al. (2010).

3.1 Minimum CVaR portfolio under an upper 40% CVaR allocation constraint

The functions needed to obtain the minimum CVaR portfolio under an upper 40% CVaR allocation constraint are the following:

```
> ObjSpec <- constraint(assets = colnames(indexes[,1:4]),min = rep(0,4),
+  max=rep(1,4), min_sum=1,max_sum=1 )
> ObjSpec <- add.objective( constraints = ObjSpec, type="risk",
+   name="CVaR", arguments=list(p=0.95,clean="boudt"),enabled=TRUE)
> ObjSpec <- add.objective( constraints = ObjSpec,
+    type="risk_budget_objective", name="CVaR", max_prisk = 0.4,
+    arguments=list(p=0.95,clean="boudt"), enabled=TRUE)
> set.seed(1234)
> out = optimize.portfolio(R= indexes[,1:4],constraints=ObjSpec,
+    optimize_method="DEoptim",itermax=10, search_size=2000)
```

After the call to these functions it starts to explore the feasible space iteratively:

```
Iteration: 1 bestvalit: 0.029506 bestmemit:    0.810000    0.126000    0.010000    0.140000
Iteration: 2 bestvalit: 0.029506 bestmemit:    0.810000    0.126000    0.010000    0.140000
Iteration: 3 bestvalit: 0.029272 bestmemit:    0.758560    0.079560    0.052800    0.112240
Iteration: 4 bestvalit: 0.029272 bestmemit:    0.758560    0.079560    0.052800    0.112240
Iteration: 5 bestvalit: 0.029019 bestmemit:    0.810000    0.108170    0.010000    0.140000
Iteration: 6 bestvalit: 0.029019 bestmemit:    0.810000    0.108170    0.010000    0.140000
Iteration: 7 bestvalit: 0.029019 bestmemit:    0.810000    0.108170    0.010000    0.140000
Iteration: 8 bestvalit: 0.028874 bestmemit:    0.692069    0.028575    0.100400    0.071600
Iteration: 9 bestvalit: 0.028874 bestmemit:    0.692069    0.028575    0.100400    0.071600
Iteration: 10 bestvalit: 0.028874 bestmemit:    0.692069    0.028575    0.100400    0.071600
elapsed time:1.85782111114926
```

If `TRACE=FALSE` the only output in `out` is the weight vector that optimizes the objective function.

```
> out[[1]]
      US Bonds    US Equities Int'l Equities    Commodities
    0.77530240     0.03201150     0.11247491     0.08021119
```

If `TRACE=TRUE` additional information is given such as the value of the objective function and the different constraints.

3.2 Minimum CVaR concentration portfolio

The functions needed to obtain the minimum CVaR concentration portfolio are the following:

```
> ObjSpec <- constraint(assets = colnames(indexes[,1:4]) ,min = rep(0,4),
+  max=rep(1,4), min_sum=1,max_sum=1 )
> ObjSpec <- add.objective( constraints = ObjSpec,
+   type="risk_budget_objective", name="CVaR",
+   arguments=list(p=0.95,clean="boudt"),
+   min_concentration=TRUE,enabled=TRUE)
> set.seed(1234)
> out = optimize.portfolio(R= indexes[,1:4],constraints=ObjSpec,
+  optimize_method="DEoptim",itermax=50, search_size=5000)
```

The iterations are as follows:

```
Iteration: 1 bestvalit: 0.010598 bestmemit:    0.800000    0.100000    0.118000    0.030000
Iteration: 2 bestvalit: 0.010598 bestmemit:    0.800000    0.100000    0.118000    0.030000
Iteration: 3 bestvalit: 0.010598 bestmemit:    0.800000    0.100000    0.118000    0.030000
Iteration: 4 bestvalit: 0.010598 bestmemit:    0.800000    0.100000    0.118000    0.030000
Iteration: 5 bestvalit: 0.010598 bestmemit:    0.800000    0.100000    0.118000    0.030000
Iteration: 45 bestvalit: 0.008209 bestmemit:    0.976061    0.151151    0.120500    0.133916
Iteration: 46 bestvalit: 0.008170 bestmemit:    0.897703    0.141514    0.109601    0.124004
Iteration: 47 bestvalit: 0.008170 bestmemit:    0.897703    0.141514    0.109601    0.124004
Iteration: 48 bestvalit: 0.008170 bestmemit:    0.897703    0.141514    0.109601    0.124004
Iteration: 49 bestvalit: 0.008170 bestmemit:    0.897703    0.141514    0.109601    0.124004
Iteration: 50 bestvalit: 0.008170 bestmemit:    0.897703    0.141514    0.109601    0.124004
elapsed time:4.1324522222413
```

This portfolio has the equal risk contribution characteristic:

```
> out[[1]]
      US Bonds    US Equities Int'l Equities    Commodities
    0.70528537     0.11118139     0.08610905     0.09742419
> ES(indexes[,1:4],weights = out[[1]],p=0.95,clean="boudt",
+  portfolio_method="component")
$MES
           [,1]
[1,] 0.03246264

$contribution
      US Bonds    US Equities Int'l Equities    Commodities
   0.008169565    0.008121930    0.008003228    0.008167917

$pct_contrib_MES
      US Bonds    US Equities Int'l Equities    Commodities
     0.2516605      0.2501931      0.2465366      0.2516098
```

3.3 Dynamic optimization

Dynamic rebalancing of the risk budget optimized portfolio is possible through the function `optimize.portfolio.rebalancing`. Additional arguments are `rebalance\_on} which indicates the rebalancin` or through moving window estimation, where each window has `trailing_periods` observations.

The minimum number of observations in the estimation sample is specified by `training_period`. Its default value is 36, which corresponds to three years for monthly data.

As an example, consider the minimum CVaR concentration portfolio, with estimation from in inception and monthly rebalancing. Since we require a minimum estimation length of total number of observations -1, we can optimize the portfolio only for the last two months.

```
> set.seed(1234)
> out = optimize.portfolio.rebalancing(R= indexes,constraints=ObjSpec, rebalance_on ="months",
+     optimize_method="DEoptim",itermax=50, search_size=5000, training_period = nrow(indexes)-1 )
```

For each of the optimization, the iterations are given as intermediate output:

```
Iteration: 1 bestvalit: 0.010655 bestmemit:    0.800000    0.100000    0.118000    0.030000
Iteration: 2 bestvalit: 0.010655 bestmemit:    0.800000    0.100000    0.118000    0.030000
Iteration: 49 bestvalit: 0.008207 bestmemit:    0.787525    0.124897    0.098001    0.108258
Iteration: 50 bestvalit: 0.008195 bestmemit:    0.774088    0.122219    0.095973    0.104338
elapsed time:4.20546416666773
Iteration: 1 bestvalit: 0.011006 bestmemit:    0.770000    0.050000    0.090000    0.090000
Iteration: 2 bestvalit: 0.010559 bestmemit:    0.498333    0.010000    0.070000    0.080000
Iteration: 49 bestvalit: 0.008267 bestmemit:    0.828663    0.126173    0.100836    0.114794
Iteration: 50 bestvalit: 0.008267 bestmemit:    0.828663    0.126173    0.100836    0.114794
elapsed time:4.1060591666566
overall elapsed time:8.31152777777778
```

The output is a list holding for each rebalancing period the output of the optimization, such as portfolio weights.

```
> out[[1]]$weights
      US Bonds    US Equities Int'l Equities    Commodities
    0.70588695     0.11145087     0.08751686     0.09514531
> out[[2]]$weights
      US Bonds    US Equities Int'l Equities    Commodities
    0.70797640     0.10779728     0.08615059     0.09807574
```

But also the value of the objective function:

```
> out[[1]]$out
[1] 0.008195072
> out[[2]]$out
[1] 0.008266844
```

The first and last observation from the estimation sample:

```
> out[[1]]$data_summary
$first
           US Bonds US Equities Int'l Equities Commodities
1980-01-31  -0.0272       0.061         0.0462      0.0568

$last
           US Bonds US Equities Int'l Equities Commodities
2009-11-30   0.0134      0.0566         0.0199       0.015

> out[[2]]$data_summary
$first
           US Bonds US Equities Int'l Equities Commodities
1980-01-31  -0.0272       0.061         0.0462      0.0568

$last
           US Bonds US Equities Int'l Equities Commodities
2009-12-31  -0.0175      0.0189         0.0143      0.0086
```

Of course, DE is a stochastic optimizaer and typically will only find a near-optimal solution that depends on the seed. The function `optimize.portfolio.parallel` in `PortfolioAnalytics` allows to run an arbitrary number of portfolio sets in parallel in order to develop "confidence bands" around your solution. It is based on RevolutionÂŠs `foreach` package (Computing, 2009).

References

D.˜Ardia and K.˜Mullen. *DEoptim: Differential Evolution Optimization in R*, 2009. URL `http://CRAN.R-project.org/package=DEoptim`. R package version 2.00-04.

D.˜Ardia, K.˜Boudt, P.˜Carl, K.˜Mullen, and B.˜Peterson. Differential evolution (deoptim) for non-convex portfolio optimization. *Mimeo*, 2010.

T.˜Bollerslev. Modeling the coherence in short-run nominal exchange rates: A multivariate generalized ARCH model. *Review of Economics and Statistics*, 72:498–505, 1990.

K.˜Boudt, P.˜Carl, and B.˜G. Peterson. Portfolio optimization with conditional value-at-risk budgets, Jan. 2010a.

K.˜Boudt, P.˜Carl, and B.˜G. Peterson. PortfolioAnalytics: Portfolio analysis, including numeric methods for optimization of portfolios, 2010b. URL `http://braverock.com/R/`. R package version 0.5.

P.˜Carl and B.˜G. Peterson. PerformanceAnalytics: Econometric tools for performance and risk analysis in R, 2009. URL `http://braverock.com/R/`. R package version 1.0.0.

P.˜Carl, B.˜G. Peterson, and K.˜Boudt. Business objectives and complex portfolio optimization. Presentation at R/Finance 2010. Available at: `http://www.rinfinance.com/agenda/2010/Carl+Peterson+Boudt_Tutorial.pdf`, 2010.

R.˜Computing. *foreach: Foreach looping construct for R*, 2009. URL `http://CRAN.R-project.org/package=foreach`. R package version 1.3.0.

K.˜M. Mullen, D.˜Ardia, D.˜L. Gil, D.˜Windover, and J.˜Cline. DEoptim: An R package for global optimization by differential evolution, Dec. 2009.

K.˜V. Price, R.˜M. Storn, and J.˜A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer-Verlag, Berlin, Germany, second edition, Dec. 2006. ISBN 3540209506.

J.˜A. Ryan and J.˜M. Ulrich. *xts: Extensible Time Series*, 2010. URL `http://CRAN.R-project.org/package=xts`. R package version 0.7-0.

R.˜Storn and K.˜Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997. ISSN 0925-5001.

Wuertz, Diethelm, Chalabi, Yohan, Chen, William, Ellis, and Andrew. *Portfolio Optimization with R/Rmetrics*. Rmetrics Association & Finance Online, www.rmetrics.org, April 2010. R package version 2110.79.

G.˜Yollin. R tools for portfolio optimization. In *Presentation at R/Finance conference 2009*, 2009.