Portfolio Optimisation with Threshold Accepting

Enrico Schumann

October 20, 2011

Abstract

A brief tutorial on using TA for portfolio optimisation.

This vignette provides the code for some of the examples from Gilli et al. [2011]. For more details, please see Chapter 13 of the book; the code in this vignette uses the scripts `exampleSquaredRets.R`, `exampleSquaredRets2.R` and `exampleRatio.R`.

We start by attaching the package. We will later on need the function `resample` (see `?sample`).

```
> require("NMOF")
> resample <- function(x, ...) x[sample.int(length(x), ...)]
> set.seed(112233)
```

1 Minimising squares

1.1 A first implementation

This problem serves as a benchmark: we wish to find a long-only portfolio w (weights) that minimises squared returns across all return scenarios. These scenarios are stored in a matrix R of size number of scenarions `ns` times number of assets `na`. More formally, we want to solve the following problem:

$$\min_{w} \Phi \tag{1}$$

$$w'\iota = 1\,,$$
$$0 \leq w_j \leq w_j^{\text{sup}} \quad \text{for } j = 1, 2, \ldots, n_A\,.$$

We set w_j^{sup} to 5% for all assets. Φ is the squared return of the portfolio, $w'R'Rw$, which is similar to the portfolio return's variance. We have

$$\frac{1}{n_S}R'R = \text{Cov}(R) + mm'$$

in which Cov is the variance–covariance matrix operator, which maps the columns of R into their variance–covariance matrix; m is a column vector that holds the column means of R, ie, $m' = \frac{1}{n_S}\iota' R$. For short time horizons, the mean of a column is small compared with the average squared return of the column. Hence, we ignore the matrix mm', and variance and squared returns become equivalent.

For testing purposes we use the matrix `fundData` for R.

```
> na <- dim(fundData)[2L]
> ns <- dim(fundData)[1L]
> winf <- 0
> wsup <- 0.05
> data <- list(R = t(fundData), RR = crossprod(fundData), na = na,
+     ns = ns, eps = 0.5/100, winf = winf, wsup = wsup)
```

The neighbourhood function automatically enforces the bugdet constraint.

```
> neighbour <- function(w, data) {
+     eps <- runif(1) * data$eps
+     toSell <- w > data$winf
```

```
+     toBuy <- w < data$wsup
+     i <- resample(which(toSell), size = 1L)
+     j <- resample(which(toBuy), size = 1L)
+     eps <- min(w[i] - data$winf, data$wsup - w[j], eps)
+     w[i] <- w[i] - eps
+     w[j] <- w[j] + eps
+     w
+ }
```

The objective function.

```
> OF1 <- function(w, data) {
+     Rw <- crossprod(data$R, w)
+     crossprod(Rw)
+ }
> OF2 <- function(w, data) {
+     aux <- crossprod(data$RR, w)
+     crossprod(w, aux)
+ }
```

OF2 uses $R'R$; thus, it does not depend on the number of scenarios. But this is only possible for this very specific problem.

So, we specify a random initial solution w0; set up algo; and run TAopt.

```
> w0 <- runif(na)
> w0 <- w0/sum(w0)
> algo <- list(x0 = w0, neighbour = neighbour, nS = 2000L, nT = 10L,
+     nD = 5000L, q = 0.2, printBar = FALSE, printDetail = FALSE)
> system.time(res <- TAopt(OF1, algo, data))

   user  system elapsed
   4.18    0.00    4.23

> 100 * sqrt(crossprod(fundData %*% res$xbest)/ns)

          [,1]
[1,] 0.3363246

> system.time(res <- TAopt(OF2, algo, data))

   user  system elapsed
   2.71    0.00    2.72

> 100 * sqrt(crossprod(fundData %*% res$xbest)/ns)

          [,1]
[1,] 0.3367206

> cat("Check constraints ...\n")

Check constraints ...

> min(res$xbest)

[1] 0

> max(res$xbest)

[1] 0.05

> sum(res$xbest)

[1] 1
```

Note that we have rescaled the result.

The problem can actually be solved quadratic programming; we use the quadprog package [Turlach and Weingessel, 2011].

```
> if (require(quadprog, quietly = TRUE)) {
+     covMatrix <- crossprod(fundData)
+     A <- rep(1, na)
+     a <- 1
+     B <- rbind(-diag(na), diag(na))
+     b <- rbind(array(-data$wsup, dim = c(na, 1)), array(data$winf,
+         dim = c(na, 1)))
+     system.time({
+         result <- solve.QP(Dmat = covMatrix, dvec = rep(0, na),
+             Amat = t(rbind(A, B)), bvec = rbind(a, b), meq = 1L)
+     })
+     wqp <- result$solution
+     cat("Compare results...\n")
+     100 * sqrt(crossprod(fundData %*% wqp)/ns)
+     100 * sqrt(crossprod(fundData %*% res$xbest)/ns)
+     cat("Check constraints ...\n")
+     min(wqp)
+     max(wqp)
+     sum(wqp)
+ }
```

```
Compare results...
Check constraints ...
[1] 1
```

1.2 Updating

Here we implement the updating of the objective function as described in Gilli et al. [2011].

```
> neighbourU <- function(sol, data) {
+     wn <- sol$w
+     toSell <- wn > data$winf
+     toBuy <- wn < data$wsup
+     i <- resample(which(toSell), size = 1L)
+     j <- resample(which(toBuy), size = 1L)
+     eps <- runif(1) * data$eps
+     eps <- min(wn[i] - data$winf, data$wsup - wn[j], eps)
+     wn[i] <- wn[i] - eps
+     wn[j] <- wn[j] + eps
+     Rw <- sol$Rw + data$R[, c(i, j)] %*% c(-eps, eps)
+     list(w = wn, Rw = Rw)
+ }
> OF <- function(sol, data) crossprod(sol$Rw)
```

Prepare the data.

```
> na <- dim(fundData)[2L]
> ns <- dim(fundData)[1L]
> winf <- 0
> wsup <- 0.05
> data <- list(R = fundData, na = na, ns = ns, eps = 0.5/100, winf = winf,
+     wsup = wsup)
```

Start with a random solution.

```
> w0 <- runif(data$na)
> w0 <- w0/sum(w0)
```

```
> x0 <- list(w = w0, Rw = fundData %*% w0)
> algo <- list(x0 = x0, neighbour = neighbourU, nS = 2000L, nT = 10L,
+     nD = 5000L, q = 0.2, printBar = FALSE, printDetail = FALSE)
> system.time(res2 <- TAopt(OF, algo, data))

   user  system elapsed
   2.01    0.00    2.02

> 100 * sqrt(crossprod(fundData %*% res2$xbest$w)/ns)

          [,1]
[1,] 0.3367288
```

Compare computing times.

References

Manfred Gilli, Dietmar Maringer, and Enrico Schumann. *Numerical Methods and Optimization in Finance*. Elsevier, 2011.

Berwin A. Turlach and Andreas Weingessel. *quadprog: Functions to solve Quadratic Programming Problems.*, 2011. R package version 1.5-4 (S original by Berwin A. Turlach; R port by Andreas Weingessel.).