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).

```r
> require("NMOF")
> resample <- function(x, ...)
\[x[\text{sample.int}(\text{length}(x), \ldots)]\]
> set.seed(112233)
```

## 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 scenarios \( n_s \) times number of assets \( n_a \). More formally, we want to solve the following problem:

\[
\begin{align*}
\min_w \Phi \\
\text{s.t.} \quad w^t i &= 1, \\
0 &\leq w_j \leq w_{j}^{\text{sup}} \quad \text{for} \ j = 1, 2, \ldots, n_A.
\end{align*}
\]

We set \( w_{j}^{\text{sup}} \) to 5\% for all assets. \( \Phi \) is the squared return of the portfolio, \( w^t R^t R w \), which is similar to the portfolio return’s variance. We have

\[
\frac{1}{n_S} R^t R = \text{Cov}(R) + m m^t
\]

in which \( \text{Cov} \) 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, \( m^t = \frac{1}{n_S} R^t i \). 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 \( m m^t \), and variance and squared returns become equivalent.

For testing purposes we use the matrix \text{fundData} for \( R \).

```r
> na <- dim(fundData)[2L]
> ns <- dim(fundData)[1L]
> winf <- 0.0; wsup <- 0.05
> data <- list(R = t(fundData),
\quad RR = \text{crossprod}(\text{fundData}),
\quad na = na,
\quad ns = ns,
\quad eps = 0.5/100,
\quad winf = \text{winf},
\quad wsup = \text{wsup},
\quad \text{resample} = \text{resample})
```

The neighbourhood function automatically enforces the budget constraint.

```r
> neighbour <- function(w, data){
\quad eps <- \text{runif}(1L) * data$eps
\quad toSell <- w > data$winf
\quad w[\text{sample.int}(\text{length}(w), \ldots, \text{toSell})] <- \text{data$winf}
\quad return(w)
}
toBuy <- w < data$wsup
i <- data$resample(which(toSell), size = 1L)
j <- data$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.

We specify a random initial solution $w_0$ and define all settings in a list algo.

> w0 <- runif(na); w0 <- w0/sum(w0)
> algo <- list(x0 = w0,
neighbour = neighbour,
nS = 2000L,
nT = 10L,
nD = 5000L,
q = 0.20,
printBar = FALSE,
printDetail = FALSE)

We can now run TAopt, first with OF1 ...

> system.time(res <- TAopt(OF1, algo, data))
24.821 0.008 2.093

> 100 * sqrt(crossprod(fundData %*% res$xbest)/ns)
[,1]
[1,] 0.33635

...and then with OF2.

> system.time(res <- TAopt(OF2, algo, data))
15.718 0.019 1.326

> 100*sqrt(crossprod(fundData %*% res$xbest)/ns)
[,1]
[1,] 0.33655

Note that we have rescaled the results (see the book for details). Both results are similar, but OF2 typically requires less time. We check the contraints.

> min(res$xbest) ## should not be smaller than data$winf
[1] 0
The problem can actually be solved quadratic programming; we use the quadprog package [Turlach and Weingessel, 2019].

```r
> if (require("quadprog", quietly = TRUE)) {
  covMatrix <- crossprod(fundData)
  A <- rep(1, na); a <- 1
  B <- rbind(diag(na), diag(na))
  b <- rbind(array(-data$wsup, dim = c(na, 1L)),
             array( data$winf, dim = c(na, 1L)))
  system.time({
    result <- solve.QP(Dmat = covMatrix,
                       dvec = rep(0,na),
                       Amat = t(rbind(A,B)),
                       bvec = rbind(a,b),
                       meq = 1L)
  })
  wqp <- result$solution
  cat("Compare results...\n")
  cat("QP: ", 100 * sqrt( crossprod(fundData %*% wqp)/ns ),"\n")
  cat("TA: ", 100 * sqrt( crossprod(fundData %*% res$xbest)/ns ) ,"\n")
  cat("Check constraints ...\n")
  cat("min weight: ", min(wqp), "\n")
  cat("max weight: ", max(wqp), "\n")
  cat("sum of weights: ", sum(wqp), "\n")
}
Compare results...
QP: 0.33612
TA: 0.33655
Check constraints ...
min weight: -9.957e-17
max weight: 0.05
sum of weights: 1
```

### 1.2 Updating

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

```r
> neighbourU <- function(sol, data){
  wn <- sol$w
  toSell <- wn > data$winf
  toBuy <- wn < data$wsup
  i <- data$resample(which(toSell), size = 1L)
  j <- data$resample(which(toBuy), size = 1L)
  eps <- runif(1) * data$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)
```
```r
> OF <- function(sol, data) crossprod(sol$Rw)

Prepare the data list (we reuse several items used before).

> data <- list(R = fundData, na = na, ns = ns, 
  eps = 0.5/100, winf = winf, wsup = wsup, 
  resample = resample)

We start, again, with a random solution, and also use the same number of iterations as before.

> 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 = 0.20,
  printBar = FALSE,
  printDetail = FALSE)

> system.time(res2 <- TAopt(OF, algo, data))

    user  system elapsed
   0.689   0.000   0.689

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

   [,1]
[1,] 0.33653

This should be faster, and we arrive at the same solution as before.

1.3 Redundant assets

We duplicate the last column of fundData.

> fundData <- cbind(fundData, fundData[, 200L])

Thus, while the dimension increases, the column rank stays unchanged.

> dim(fundData)

     [1] 500 201

> qr(fundData)$rank

     [1] 200

> qr(cov(fundData))$rank

     [1] 200

Checking the weight of the last asset (which was zero), we know that the solution to our model must be unchanged, too.

> if (require("quadprog", quietly = TRUE))
  wqp[200L]

     [1] 4.2811e-17

We redo our example.
```
But a number of QP solvers have problems with such cases.

```r
> if (require("quadprog", quietly = TRUE)) {
  covMatrix <- crossprod(fundData)
  A <- rep(1, na); a <- 1
  B <- rbind(-diag(na), diag(na))
  b <- rbind(array(-data$wsup, dim = c(na, 1L)),
             array( data$winf, dim = c(na, 1L)))
  cat(try(result <- solve.QP(Dmat = covMatrix,
                              dvec = rep(0, na),
                              Amat = t(rbind(A, B)),
                              bvec = rbind(a, b),
                              meq = 1L))
}
```

Error in solve.QP(Dmat = covMatrix, dvec = rep(0, na), Amat = t(rbind(A, : matrix D in quadratic function is not positive definite!

But TA can handle this case.

```r
> 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 = 0.20,
              printBar = FALSE,
              printDetail = FALSE)
> system.time(res3 <- TAopt(OF, algo, data))
```

```
user  system elapsed
0.682  0.000  0.681
```

```r
> 100*sqrt(crossprod(fundData %*% res3$xbest$w)/ns)
```

```
[,1]
[1,] 0.33671
```

Final check: weights for asset 200 and its twin, asset 201.

```r
> res3$xbest$w[200:201]
```

```
[1] 0 0
```

See Gilli et al. [2011, Section 13.2.5] for a discussion of rank-deficiency and its (computational and empirical) consequences for such problems.

### References