Performance test of portfolio optimization with PortfolioAnalytics and RcppRP.

```r
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

library(RcppRP)

## Loading required package:  Rcpp
## Loading required package:  RcppArmadillo
##
## Attaching package:  'RcppRP'
##
## The following object is masked from 'package:PortfolioAnalytics':
##
##     rp_sample

library(rbenchmark)

# Use parallel
require(doMC)

## Loading required package:  doMC
## Loading required package:  foreach
## Loading required package:  iterators
## Loading required package:  parallel

registerDoMC(3)

data(edhec)
R <- edhec[,1:10]
funds <- colnames(R)

weight_seq <- generatesequence(min=0, max=0.45, by=0.001)

# Add basic constraints and objectives
init.portf <- portfolio.spec(assets=funds, weight_seq=weight_seq)
init.portf <- add.constraint(portfolio=init.portf, type="weight_sum",
                             min_sum=0.99, max_sum=1.01)
init.portf <- add.constraint(portfolio=init.portf, type="box", min=0, max=0.45)
init.portf <- add.objective(portfolio=init.portf, type="return", name="mean")
init.portf <- add.objective(portfolio=init.portf, type="risk", name="sd")
```

The first benchmark is to compare creating random portfolios using PortfolioAnalytics and RcppRP. The `rcpp_random_portfolios` function is a C++ implementation of the algorithm used in PortfolioAnalytics for generating random portfolios using the "sample" method.

```
n_portfolios <- 5000
benchmark(pa=random_portfolios(portfolio=init.portf,
                               permutations=n_portfolios,
                               rp_method="sample"),
          rcpp_s=rcpp_random_portfolios(portfolio=init.portf,
                                        n_portfolios=n_portfolios,
                                        method="sample"),
          replications=10)[,1:4]

##     test replications elapsed relative
## 1     pa           10  188.74    6.583
## 2 rcpp_s           10   28.67    1.000
```

A speed improvement of 6x is pretty good, but not that significant overall because I only need to generate the random portfolios one time for a single optimization and can re-use that set of random portfolios assuming the same assets, same constraints, etc. for the optimization.

Test the portfolio optimization of RcppRP and make sure I get the same results as PortfolioAnalytics.

```
# Test with a small number of portfolios
rp <- random_portfolios(portfolio=init.portf, rp_method="sample",
                        permutations=100)

pa_opt <- optimize.portfolio(R=R, portfolio=init.portf,
                             optimize_method="random",
                             rp=rp, trace=TRUE)
xtract <- extractStats(pa_opt)

rcpp_opt <- rp_optimize_v2(R=R, portfolio=init.portf, rp=rp)

# make sure the results are equal
all.equal(xtract, rcpp_opt$rp_results, check.attributes=FALSE)

## [1] TRUE
```

Benchmark the optimization functions of PortfolioAnalytics and RcppRP. The `rp_optimize_v2` uses slimmed down C++ implementations of `constrained_objective` and `optimize.portfolio` from PortfolioAnalytics. The objective, constrained objective, and optimization functions must all be in C++ so that I can "stay in C++ world" for the optimization when calling `constrained_objective` for each set of weights.

```
# Create random portfolios that will be passed to both optimization functions
rp <- random_portfolios(portfolio=init.portf, rp_method="sample",
                        permutations=5000)

benchmark(pa=optimize.portfolio(R=R, portfolio=init.portf,
                                 optimize_method="random", rp=rp),
          rcpp=rp_optimize_v2(R=R, portfolio=init.portf, rp=rp),
          replications=10)[,1:4]

##   test replications elapsed relative
## 1   pa           10 211.027    808.5
## 2 rcpp           10   0.261      1.0
```

This was a pretty simple optimization to maximize mean return per unit standard deviation with full investment and box constraints, but the performance improvement of more than 700x is significant. I haven't tested this with larger scale problems of 100+ assets, but I suspect that the improvement will be similar. This performance improvement will be even more noticeable for the implementation of `optimize.portfolio.rebalancing`.