Large-scale portfolio optimization with DEoptim

Kris Boudt
Lessius and K.U.Leuven

David Ardia
aeris CAPITAL AG

Katharine M. Mullen
NIST

Brian G. Peterson
Cheiron Trading

Abstract

This vignette evaluates the performance of **DEoptim** on a high-dimensional portfolio problem. The setup is the same as in the *R Journal* article Ardia *et al.* (2010); namely minimizing the portfolio CVaR under an upper bound constraint on the percentage CVaR contributions. Because the resulting optimization model is a non-convex programming problem with multiple local optima, **DEoptim** is more apt in solving this problem than gradient-based methods such as `optim` and `nlminb`.

Keywords: portfolio optimization, evolutionary algorithm, Differential Evolution, R software.

1. Introduction

The R package **DEoptim** of Ardia *et al.* (2011) implements several Differential Evolution algorithms. DE belongs to the class of genetic algorithms which use biology-inspired operations of crossover, mutation, and selection on a population in order to minimize an objective function over the course of successive generations. More details on **DEoptim** can be found in Mullen *et al.* (2011). In this vignette we evaluate its performance on a high-dimensional portfolio problem.

In this vignette, we use of **DEoptim** to solve the portfolio optimization problem described Ardia *et al.* (2010), but expand the problem to include 100 free variables. As in Ardia *et al.* (2010), the portfolio CVaR is minimized under an upper bound constraint on the percentage CVaR contributions. See Boudt *et al.* (2010) for the rational underlying this portfolio construction technique.

The optimization of the objective function is a non-convex programming problem. Since **DEoptim** is a stochastic global optimization algorithm, it is more apt to offer a good solution than alternative local optimization methods. For instance, gradient-based methods such as the `L-BFGS-B` and `Nelder-Mead` methods in `optim` and `nlminb` will typically converge to suboptimal solutions on the problem considered here.

The version of this problem described in Ardia *et al.* (2010) was stylized to describe many fewer variables to allow users to complete the optimization on a personal computer in a matter of minutes. Portfolio problems encountered in practice may require days to optimize on a personal computer, and involve several hundred variables. In the present vignette we examine a problem of complexity in-between the stylized, simple case considered by Ardia

et al. (2010), and the most complex problems encountered in typical financial research.

We hope that the R package **DEoptim** will be fruitful for many users. If you use R or **DEoptim**, please cite the software in publications.

2. Setup

The results in this vignette are obtained using R version 2.13.0. The function `getSymbols` in **quantmod** (Ryan 2010) is used to obtain the data. The risk measures in the portfolio objective function are computed using **PerformanceAnalytics** (Carl and Peterson 2010). The initial population in **DEoptim** is generated using the function `random_portfolios` in **PortfolioAnalytics** (Boudt *et al.* 2011).[1]

Computations are performed on a Genuine Intel® dual core CPU P8400 2.26Ghz processor. **DEoptim** relies on repeated evaluation of the objective function in order to move the population toward a global minimum. Users interested in making **DEoptim** run as fast as possible should ensure that evaluation of the objective function is as efficient as possible. Using pure R code, this may often be accomplished using vectorization. Writing parts of the objective function in a lower-level language like C or Fortran may also increase speed.

3. Data

We take 100 randomly sampled stocks from the S&P 500 for which a sufficiently long data history is available. We first download ten years of monthly data using the function `getSymbols` of the package **quantmod** (Ryan 2010). Then we compute the log-return series and the sample mean and covariance matrix.

```
> tickers =  c( "VNO"  , "VMC" ,  "WMT" ,  "WAG"  , "DIS" ,  "WPO"  , "WFC"  , "WDC" ,
+   "WY" ,  "WHR"  , "WMB" ,  "WEC" ,  "XEL"  , "XRX"  , "XLNX"  ,"ZION"  ,"MMM" ,
+   "ABT",  "ADBE" , "AMD"  , "AET" ,  "AFL"  , "APD"  , "ARG"   ,"AA"  ,  "AGN" ,
+   "ALTR" , "MO"   , "AEP" ,  "AXP" ,  "AIG"  , "AMGN" , "APC"   ,"ADI" ,  "AON" ,
+   "APA",   "AAPL" , "AMAT"  ,"ADM" ,  "T"  ,   "ADSK" , "ADP" ,  "AZO" ,  "AVY" ,
+   "AVP",   "BHI"  , "BLL"  , "BAC" ,  "BK"  ,  "BCR"  , "BAX" ,  "BBT" ,  "BDX"  ,
+   "BMS" ,  "BBY"  , "BIG"  , "HRB" ,  "BMC" ,  "BA"  ,  "BMY" ,  "CA"  ,  "COG"  ,
+   "CPB" ,  "CAH"  , "CCL"  , "CAT" ,  "CELG" , "CNP" ,  "CTL"  , "CEPH",  "CERN" ,
+   "SCHW" , "CVX"  , "CB"   , "CI"    ,"CINF"  ,"CTAS" , "CSCO" , "C"   ,  "CLF"  ,
+   "CLX",   "CMS"  , "KO"   , "CCE"  , "CL"   , "CMCSA" ,"CMA" ,  "CSC" ,  "CAG" ,
+   "COP"  , "ED"   , "CEG"   ,"GLW" ,  "COST" , "CVH"  , "CSX" ,  "CMI" ,  "CVS"  ,
+   "DHR" ,  "DE")
>
> library(quantmod);
> getSymbols(tickers, from = "2000-12-01", to = "2010-12-31")
> P <- NULL; seltickers <- NULL
> for(ticker in tickers) {
```

[1] All packages are readily available from CRAN, except for **PortfolioAnalytics** which can be downloaded from https://r-forge.r-project.org/R/?group_id=579 or installed directly within R using the command `install.packages("PortfolioAnalytics", repos="http://R-Forge.R-project.org")`.

```
+   tmp <- Cl(to.monthly(eval(parse(text=ticker))))
+   if(is.null(P)){ timeP = time(tmp) }
+   if( any( time(tmp)!=timeP )) next
+   else P<-cbind(P,as.numeric(tmp))
+   seltickers = c( seltickers , ticker )
+ }
> P = xts(P,order.by=timeP)
> colnames(P) <- seltickers
> R <- diff(log(P))
> R <- R[-1,]
> dim(R)
[1] 120 100
> mu <- colMeans(R)
> sigma <- cov(R)
```

4. Portfolio objective function and constraints

The optimization problem consists of determining the portfolio weights for which the portfolio has the lowest CVaR and each investment can contribute at most 5% to total portfolio CVaR risk. Additionally, weights need to be positive and the portfolio needs to be fully invested.

The level of portfolio CVaR and the CVaR contributions are computed conveniently with the function `ES` in the package **PerformanceAnalytics** (Carl and Peterson 2010). For simplicity, we assume here normality, but also estimators of CVaR and CVaR contributions for non-normal distributions are available in the function `ES`.

The constraint that each asset can contribute at most 5% to total portfolio CVaR risk is imposed through the addition of a penalty function to the objective function. As such, we allow the search algorithm to consider infeasible solutions. A portfolio which is unacceptable for the investor must be penalized enough to be rejected by the minimization process and the larger the violation of the constraint, the larger the increase in the value of the objective function.

```
> library("PerformanceAnalytics")
> obj <- function(w) {
+    if (sum(w) == 0) {
+     w <- w + 1e-2
+    }
+    w <- w / sum(w)
+    CVaR <- ES(weights = w,
+      method = "gaussian",
+      portfolio_method = "component",
+      mu = mu,
+      sigma = sigma)
+    tmp1 <- CVaR$ES
+    tmp2 <- max(CVaR$pct_contrib_ES - 0.05, 0)
+    out <- tmp1 + 1e3 * tmp2
```

```
+    return(out)
+ }
```

The weights need to satisfy additionally a long only and full investment constraint. The current implementation of `DEoptim` allows for bound constraints on the portfolio weights. We call these `lower` and `upper`.

```
> N <- ncol(R)
> minw <- 0
> maxw <- 1
> lower <- rep(minw,N)
> upper <- rep(maxw,N)
```

The full investment constraint is accounted for in two ways. First, we standardize all the weights in the objective function such that they sum up to one. Second, we use the function `random_portfolios` in **PortfolioAnalytics** (Boudt *et al.* 2011) to generate random portfolios that satisfy all constraints. These random portfolios will be used as the initial generation in `DEoptim`.

```
> library("PortfolioAnalytics")
> eps <- 0.025
> weight_seq<-generatesequence(min=minw,max=maxw,by=.001,rounding=3)
> rpconstraint<-constraint(
+     assets=N, min_sum=(1-eps), max_sum=(1+eps),
+      min=lower, max=upper, weight_seq=weight_seq)
assuming equal weighted seed portfolio
> set.seed(1234)
> rp<- random_portfolios(rpconstraints=rpconstraint,permutations=N*10)
> rp <-rp/rowSums(rp)
```

5. Failure of gradient-based methods

The penalty introduced in the objective function is non-differentiable and therefore standard gradient-based optimization routines cannot be used. For instance, `L-BFGS-B` and `Nelder-Mead` methods in `optim` and `nlminb` do not converge.

```
> out <- optim(par = rep(1/N, N), fn = obj,
+      method = "L-BFGS-B", lower = lower, upper = upper)
> out$value

[1] 0.05431692
> out$message
[1] "ERROR: ABNORMAL_TERMINATION_IN_LNSRCH"

> out <- nlminb(start =rep(1/N, N), objective = obj,
+      lower = lower, upper = upper)
> out$objective
```

```
[1] 0.0547231

> out$message

[1] "false convergence (8)"
```

6. Portfolio optimization with DEoptim

In contrast with gradient-based methods, `DEoptim` is designed to consistently find a good approximation to the global minimum of the optimization problem. For complex problems as the one considered here, the performance of **DEoptim** is quite dependent on the DE algorithm used.

We first consider the current default DE algorithm in **DEoptim**, called the "local-to-best" strategy with fixed parameters.

We define convergence when the percentage improvement between iterations is below `reltol=1e-6` after `steptol=150` steps. For some problems, it may take many iterations before the DE algorithm converges. We set the maximum number of iterations allowed to 5000. Progress is printed every 250 iterations.[2] As explained above, the initial generation is set to `rp`.

```
> controlDE <- list(reltol=.000001,steptol=150, itermax = 5000,trace = 250,
+  NP=as.numeric(nrow(rp)),initialpop=rp)
> set.seed(1234)
> start <- Sys.time()
> out <- DEoptim(fn = obj, lower = lower, upper = upper, control = controlDE)
Iteration: 250 bestvalit: 0.064652
Iteration: 500 bestvalit: 0.057199
Iteration: 750 bestvalit: 0.055774
Iteration: 1000 bestvalit: 0.055013
Iteration: 1250 bestvalit: 0.054581
Iteration: 1500 bestvalit: 0.054269
Iteration: 1750 bestvalit: 0.054146
Iteration: 2000 bestvalit: 0.054049
Iteration: 2250 bestvalit: 0.053706
Iteration: 2500 bestvalit: 0.053695
Iteration: 2750 bestvalit: 0.053351
Iteration: 3000 bestvalit: 0.053273
> out$optim$iter
[1] 3055
> out$optim$bestval
[1] 0.05327273
> end <- Sys.time()
> end - start
Time difference of 16.03645 mins
```

[2] For convenience in presentation, we display only the best value for the sected iterations, while the **DEoptim** output in R also displays the best member for each iteration.

We thus see that, at iteration 5000, the "local-to-best" strategy with fixed parameters reaches a solution that is better than the one obtained using the gradient-based methods mentioned above.

A recently proposed DE algorithm with better convergence properties on complex problems is the JADE algorithm proposed by Zhang and Sanderson (2009). JADE combines a "local-to-$p$best" strategy with adaptive parameter control.

The first building block of JADE is thus that the DE algorithm does not always use the best solution of the current generation to mutate the solutions, but one of the randomly chosen $\lfloor 100p\% \rfloor$ best solutions, with $0 < p \leq 1$. The default value of p is 0.2. Even though this strategy is more greedy, it tends to converge faster because it diversifies the population. The "local-to-$p$best" strategy is chosen by setting `strategy=6`.

```
> controlDE <- list(reltol=.000001,steptol=150, itermax = 5000,trace = 250,
+  strategy=6, c=0,
+  NP=as.numeric(nrow(rp)),initialpop=rp)
> set.seed(1234)
> start <- Sys.time()
> out <- DEoptim(fn = obj, lower = lower, upper = upper, control = controlDE)
Iteration: 250 bestvalit: 0.063848
Iteration: 500 bestvalit: 0.058090
Iteration: 750 bestvalit: 0.055960
Iteration: 1000 bestvalit: 0.055235
Iteration: 1250 bestvalit: 0.054884
Iteration: 1500 bestvalit: 0.054369
Iteration: 1750 bestvalit: 0.054269
Iteration: 2000 bestvalit: 0.054089
> out$optim$bestval
[1] 0.05408688
> end <- Sys.time()
> end - start
Time difference of 11.45833 mins
```

The second distinctive feature of Zhang and Sanderson (2009) is to introduce learning about successful parameters in the algorithm. Under this approach, the cross-over probability at generation $g + 1$ is set to $(1 - c)$ the cross-over probability at generation g plus c times the the average of all successful cross-over probabilities at generation g. Similarly, the mutation factor at generation $g + 1$ is equal to $1 - c$ times the previous mutation factor plus c times the average mutation factor of all successful mutations. We take $c = .4$.

```
> controlDE <- list(reltol=.000001,steptol=150, itermax = 5000,trace = 250,
+  strategy=2, c=.4,
+  NP=as.numeric(nrow(rp)),initialpop=rp)
> set.seed(1234)
> start <- Sys.time()
> out <- DEoptim(fn = obj, lower = lower, upper = upper, control = controlDE)
Iteration: 250 bestvalit: 0.074612
```

```
Iteration: 500 bestvalit: 0.068776
Iteration: 750 bestvalit: 0.067991
Iteration: 1000 bestvalit: 0.067894
Iteration: 1250 bestvalit: 0.067887
> out$optim$bestval
[1] 0.06788674
> end <- Sys.time()
> end - start
Time difference of 6.5763 mins
```

The "local-to-1best" strategy with adaptive parameter control converges clearly too fast. It is the combination of "local-to-*p*best" strategy with adaptive parameter control that is the most successful in solving our problem.

```
> controlDE <- list(reltol=.000001,steptol=150, itermax = 5000,trace = 250,
+  strategy=6, c=.4,
+  NP=as.numeric(nrow(rp)),initialpop=rp)
> set.seed(1234)
> start <- Sys.time()
> out <- DEoptim(fn = obj, lower = lower, upper = upper, control = controlDE)
Iteration: 250 bestvalit: 0.087517
Iteration: 500 bestvalit: 0.077481
Iteration: 750 bestvalit: 0.067673
Iteration: 1000 bestvalit: 0.059015
Iteration: 1250 bestvalit: 0.054758
Iteration: 1500 bestvalit: 0.053618
Iteration: 1750 bestvalit: 0.053290
Iteration: 2000 bestvalit: 0.053156
Iteration: 2250 bestvalit: 0.053099
Iteration: 2500 bestvalit: 0.053071
Iteration: 2750 bestvalit: 0.053059
Iteration: 3000 bestvalit: 0.053052
Iteration: 3250 bestvalit: 0.053049
> out$optim$iter
[1] 3451
> out$optim$bestval
[1] 0.0530483
> end <- Sys.time()
> end - start
Time difference of 18.57083 mins
```

We see that with JADE, **DEoptim** converges within 3451 iterations to 0.0530483, which is the lowest obtained by all methods considered in the vignette.

This vignette illustrates that for complex problems, the performance of **DEoptim** is thus quite dependent on the DE algorithm used. It is recommended that users try out several DE algorithms to find out which one is most adapted for their problem.

Furthermore, DE is a stochastic optimizer 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 REvolution's **foreach** package (REvolution Computing 2009).

References

Ardia D, Boudt K, Carl P, Mullen KM, Peterson BG (2010). "Differential Evolution (DEoptim) for Non-Convex Portfolio Optimization." URL http://ssrn.com/abstract=1584905.

Ardia D, Mullen K, Peterson BG, Ulrich J (2011). ***DEoptim**: Differential Evolution Optimization in R*. R package version 2.1-0, URL http://CRAN.R-project.org/package=DEoptim.

Boudt K, Carl P, Peterson BG (2010). "Portfolio Optimization with Conditional Value-at-Risk Budgets."

Boudt K, Carl P, Peterson BG (2011). "PortfolioAnalytics: Portfolio Analysis, including numeric methods for optimization of portfolios." R package version 0.6, URL https://r-forge.r-project.org/R/?group_id=579.

Carl P, Peterson BG (2010). *PerformanceAnalytics: Econometric tools for performance and risk analysis*. R package version 1.0.3.2, URL http://CRAN.R-project.org/package=PerformanceAnalytics.

Mullen K, Ardia D, Gil D, Windover D, Cline J (2011). "DEoptim: An R Package for Global Optimization by Differential Evolution." *Journal of Statistical Software*, **40**(6), 1–26. URL http://www.jstatsoft.org/v40/i06/.

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

Ryan JA (2010). *quantmod: Quantitative Financial Modelling Framework*. R package version 0.3-15, URL http://CRAN.R-project.org/package=quantmod.

Zhang J, Sanderson AC (2009). "JADE: Adaptive Differential Evolution with Optional External Archive." *IEEE Transactions on Evolutionary Computation*, **13**(5), 945–958.

Affiliation:

Kris Boudt
Lessius and K.U.Leuven, Belgium
E-mail: kris.boudt@econ.kuleuven.be

David Ardia
aeris CAPITAL AG, Switzerland
URL: http://perso.unifr.ch/david.ardia/

Katharine Mullen
Ceramics Division, National Institute of Standards and Technology (NIST)
100 Bureau Drive, MS 8520, Gaithersburg, MD, 20899, USA
E-mail: Katharine.Mullen@nist.gov

Brian G. Peterson
Cheiron Trading LLC, Chicago, IL
E-mail: brian@braverock.com