Differential Evolution (DEoptim) for Non-Convex Portfolio Optimization

by David Ardia, Kris Boudt, Peter Carl, Katharine M. Mullen and Brian G. Peterson

Abstract The R package **DEoptim** implements the differential evolution algorithm. This algorithm is an evolutionary technique similar to classic genetic algorithms that is useful for the solution of global optimization problems. In this note we provide an introduction to the package and demonstrate its utility for financial applications by solving a non-convex portfolio optimization problem.

Introduction

Differential Evolution (DE) is a search heuristic introduced by Storn and Price (1997). Its remarkable performance as a global optimization algorithm on continuous numerical minimization problems has been extensively explored (Price et al., 2006; Lampinen, 2009). DE has also become a powerful tool for solving optimization problems that arise in financial applications: for fitting sophisticated models (Gilli et al., 2008; Gilli and Schumann, 2009; Gilli and Winker, 2008; Mullen et al., 2009), for performing model selection (Maringer, 2005; Maringer and Meyer, 2008; Maringer and Oyewumi, 2007), or for optimizing portfolios under non-convex settings (Maringer and Oyewumi, 2007; Krink et al., 2009; Krink and Paterlini, 2009; Yollin, 2009). DE is available in R with the package **DEoptim**.

In what follows, we briefly sketch the DE algorithm and discuss the content of the package **DEoptim**. The utility of the package is then explored by solving a non-convex portfolio optimization problem.

Differential evolution

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 (Holland, 1975). As with other evolutionary algorithms, DE solves optimization problems by evolving a population of candidate solutions using alteration and selection operators. DE uses floating-point instead of bit-string encoding of population members, and arithmetic operations instead of logical operations in mutation.

Let NP denote the number of parameter vectors (members) $\boldsymbol{x} \in \mathbb{R}^d$ in the population. In order to create the initial generation, NP guesses for the optimal value of the parameter vector are made, either using random values between upper and lower bounds (defined by the user) or using values given by the user. Each generation involves creation of a new population from the current population members $\{\boldsymbol{x}_i \,|\, i = 1,\ldots,NP\}$, where i indexes the vectors that make up the population. This is accomplished using *differential mutation* of the population members. An initial mutant parameter vector $\boldsymbol{v}_i$ is created by choosing three members of the population, $\boldsymbol{x}_{r_0}$, $\boldsymbol{x}_{r_1}$ and $\boldsymbol{x}_{r_2}$, at random. Then $\boldsymbol{v}_i$ is generated as

$$\boldsymbol{v}_i \doteq \boldsymbol{x}_{r_0} + F \cdot (\boldsymbol{x}_{r_1} - \boldsymbol{x}_{r_2})$$

where F is a positive scale factor, effective values for which are typically less than one. After the first mutation operation, mutation is continued until d mutations have been made, with a crossover probability $CR \in [0,1]$. The crossover probability CR controls the fraction of the parameter values that are copied from the mutant. If an element of the trial parameter vector is found to violate the bounds after mutation and crossover, it is reset in such a way that the bounds are respected (with the specific protocol depending on the implementation). Then, the objective function values associated with the children are determined. If a trial vector has equal or lower objective function value than the previous vector it replaces the previous vector in the population; otherwise the previous vector remains. Variations of this scheme have also been proposed; see Price et al. (2006).

Intuitively, the effect of the scheme is that the shape of the distribution of the population in the search space is converging with respect to size and direction towards areas with high fitness. The closer the population gets to the global optimum, the more the distribution will shrink and therefore reinforce the generation of smaller difference vectors.

For more details on the DE strategy, we refer the reader to Price et al. (2006), Storn and Price (1997) and Lampinen (2009).

The package DEoptim

DEoptim (Ardia and Mullen, 2009) was first published on CRAN in 2005. Early versions were written in pure R. Since version 2.0-0 (published to CRAN in 2009) the package has relied on a interface to a C implementation of DE, which is significantly faster on most problems as compared to the implementation in pure R. Since version 2.0-3, the C implementation dynamically allocates the memory required to store

the population, removing limitations on the number of members in the population and length of the parameter vectors that may be optimized. Since becoming publicly available, **DEoptim** has been used by several authors, e.g., Börner et al. (2007), Higgins et al. (2007), Cao et al. (2009), and Opsina Arango (2009), to solve optimization problems arising in diverse domains.

DEoptim consists of the core function `DEoptim` whose arguments are:

- `fn`: the function to be optimized (minimized). The function should have as its first argument the vector or real-valued parameters to optimize, and return a scalar real result.
- `lower`, `upper`: two vectors specifying scalar real lower and upper bounds on each parameter to be optimized, so that the *i*th element of `lower` and `upper` applies to the *i*th parameter. The implementation searches between `lower` and `upper` for the global optimum of `fn`.
- `control`: a list of tuning parameters: `VTR` (default: `-Inf`), the value to reach, which specifies the global minimum of `fn` if it is known, or if you wish to cease optimization after having reached a certain value. `NP` (default: 50), the number of population members. `itermax` (default: 200), the maximum number of iterations (i.e., population generations) allowed. `F` (default: 0.8), the scaling factor used in the mutation. `CR` (default: 0.9) the crossover probability. For details on the other `control` parameters, the reader is referred to the documentation manual (by typing `?DEoptim`) or to Mullen et al. (2009). For convenience, the function `DEoptim.control()` returns a list (and a member of the S3 class `DEoptim.control`) with default elements of `control`.
- `...`: allows the user to pass additional arguments to the function `fn`.

The output of the function `DEoptim` is a member of the S3 class `DEoptim`. Members of this class have a `plot` method that accepts the argument `plot.type`. `plot.type = "bestmemit"` results in a plot of the parameter values that represent the lowest value of the objective function each generation. `plot.type = "bestvalit"` plots the best value of the objective function each generation. Finally, `plot.type = "storepop"` results in a plot of stored populations (which are only available if these have been saved by setting the `control` argument of `DEoptim` appropriately).

Let's quickly illustrate the package's usage with the minimization of the Rastrigin function in $\mathbb{R}^2$, which is a common test for global optimization:

```
> Rastrigin <- function(x) {
+   sum(x^2 - 10 * cos(2 * pi * x)) + 20
}
```

The global minimum is zero at point $\boldsymbol{x} = (0,0)'$. A perspective plot of the function is shown in Figure 1.

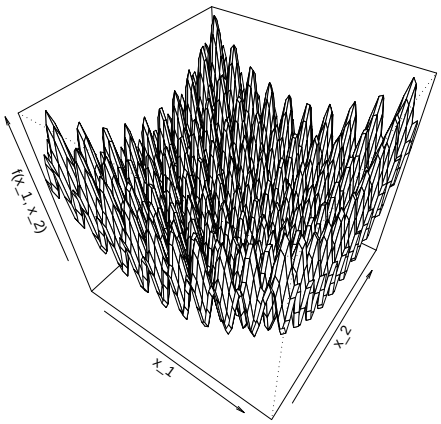

Figure 1: Perspective plot of the `Rastrigin` function.

The function `DEoptim` searches for a minimum of the objective function between `lower` and `upper` bounds. A call to `DEoptim` can be made as follows:

```
> DEoptim(fn = Rastrigin,
+   lower = c(-5, -5),
+   upper = c(5, 5),
+   control = list(storepopfrom = 1))
```

The above call specifies the objective function to minimize, `Rastrigin`, the lower and upper bounds on the parameters, and, via the `control` argument, that we want to store intermediate populations from the first generation onwards (`storepopfrom = 1`). Storing intermediate populations allows us to examine the progress of the optimization in detail. Upon initialization, the population is comprised of 50 random values (50 being the default value of `NP`) drawn uniformly within the `lower` and `upper` bounds. The members of the population generated by the above call are plotted at the end of different generations in Figure 2. `DEoptim` consistently finds the minimum of the function within 200 generations using the default settings. We have observed that **DEoptim** solves the Rastrigin problem more efficiently than the simulated annealing method available in the R function `optim` (for all annealing schedules tried).

Note that **DEoptim** relies on repeated evaluation of the objective function `fn` in order to move the population toward a global minimum. Therefore, 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,

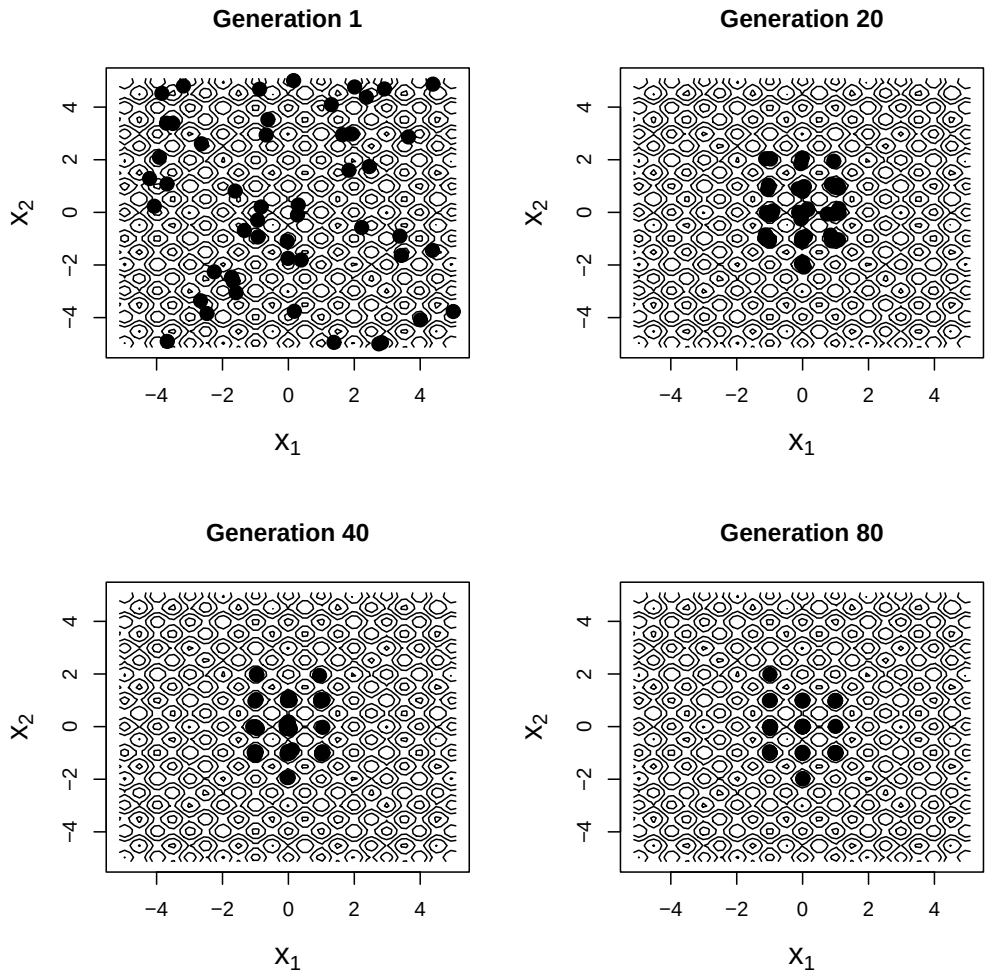

Figure 2: The population associated with various generations of a call to `DEoptim` as it searches for the minimum of the `Rastrigin` function at point $x = (0,0)'$. The minimum is consistently determined within 200 generations using the default settings of `DEoptim`.

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.

Risk allocation portfolios

It is generally accepted that rational investors should allocate their portfolio optimally according to a return/risk criterion. This often amounts to defining optimal portfolios as minimizers of non-linear functions of the portfolio return and risk subject to non-linear constraints. As mentioned previously, the package **DEoptim** is well suited to solve these problems. This was first illustrated by Yollin (2009). **DEoptim** is also the evolutionary optimization strategy used in **PortfolioAnalytics** (Boudt et al., 2010b).

Here we illustrate the use of **DEoptim** to find portfolios whose downside risk exposure is optimized. Value-at-Risk (VaR) and Conditional Value-at-Risk (CVaR or ES) are the most popular measures of downside risk. VaR is the negative value of the portfolio return such that lower returns will only occur with at most a preset probability level, which typically is between one and five percent. CVaR is the negative value of the mean of all return realizations that are below the VaR. Boudt et al. (2010a) propose to use the Euler decomposition of portfolio conditional value-at-risk (CVaR) as an objective function or constraint in the portfolio optimization problem to create portfolios that are aligned with the desired level of risk diversification. This strategy is motivated by the result that the contributions to CVaR can be interpreted as the expectation of the return on the portfolio component conditional on the portfolio return being larger than its VaR loss (Scaillet, 2002).

To illustrate this, consider as a stylized example, a five-asset portfolio invested in the stocks with tickers GE, IBM, JPM, MSFT and WMT. Under the assumption of normality, the percentage CVaR contributions are an explicit function of the mean and covariance matrix. We first download ten years of closing price data using the function `get.hist.quote` of the package **tseries**. Then we compute the monthly log-return series and the mean and covariance matrix estimators.

```
> library("tseries")
> tickers <- c("GE", "IBM", "JPM", "MSFT", "WMT")
> for (ticker in tickers) {
+   close <- get.hist.quote(instrument = ticker,
+     start = "1989-12-01",
+     end = "2009-12-31",
+     retclass = "zoo",
+     quote = "AdjClose",
+     compression = "m")
+   if (ticker == "GE") {
+     P <- close
+   }
+   else {
+     P <- merge(P, close)
+   }
+ }
> R <- diff(log(P))
> colnames(R) <- tickers
> mu <- apply(R, 2, "mean")
> sigma <- cov(R)
```

Investors interested in risk diversification tend to choose portfolios that are close to the equal-weight portfolio. But is the risk exposure of this portfolio effectively well diversified across the different assets? This question can be answered by computing the percentage CVaR contributions with function `ES` in the package **PerformanceAnalytics** (Carl and Peterson, 2010). These percentage CVaR contributions indicate how much each asset contributes to the total portfolio CVaR.

```
> library("PerformanceAnalytics")
> pContribCVaR <- ES(weights = rep(0.2, 5),
+   method = "gaussian",
+   portfolio_method = "component",
+   mu = mu,
+   sigma = sigma)$pct_contrib_ES)
> rbind(tickers, round(pContribCVaR, 3))
        [,1]    [,2]    [,3]    [,4]    [,5]
tickers "GE"    "IBM"   "JPM"   "MSFT"  "WMT"
        "0.185" "0.192" "0.265" "0.235" "0.123"
```

We see that in the equal-weight portfolio, 26.5% of the portfolio CVaR risk is caused by the 20% investment in JPM, while the 20% investment in WMT only causes 12.3% of total portfolio CVaR. The risk contribution of the other investments is close to their portfolio weight. The high risk contribution of JPM is due to its high standard deviation and low average return:

```
> round(mu, 3)
   GE   IBM   JPM  MSFT   WMT
0.007 0.009 0.009 0.017 0.010
> round(diag(sigma)^1/2, 3)
   GE   IBM   JPM  MSFT   WMT
0.003 0.004 0.005 0.005 0.002
```

We now use the function `DEoptim` of the package **DEoptim** to find the portfolio weights for which the portfolio has the lowest CVaR and each investment can contribute at most 22.5% to total portfolio CVaR risk. For this, we first define our objective function to minimize:

```
> 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 - 0.225, 0)
+   out <- tmp1 + 1e3 * tmp2
+ }
```

The first two lines are to ensure that all weights sum up to unity. Note that we introduced the risk allocation constraint though a penalty in the objective function. This penalty is non-differentiable and therefore standard optimization routines cannot be used. Several other optimization routines are available in R (e.g., `optim`, `nlminb`, `constrOptim`) without achieving the same performance as `DEoptim`. These other routines found local minima, took longer, or did not converge, even on our relatively simple stylized example. In contrast, `DEoptim` is more robust in finding a good approximation to the global minimum of this optimization problem:

```
> out <- DEoptim(fn = obj,
+   lower = rep(0, 5),
+   upper = rep(1, 5))
> out$optim$bestval
[1] 0.1063684
> wstar <- out$optim$bestmem
> wstar <- wstar / sum(wstar)
> rbind(tickers, round(wstar, 3))
        par1    par2    par3    par4    par5
tickers "GE"    "IBM"   "JPM"   "MSFT"  "WMT"
        "0.224" "0.215" "0.105" "0.166" "0.291"
> sum(wstar * mu) - mean(mu)
[1] -0.0001759887
```

Note that the main differences with the equal-weight portfolio is the low weight given to JPM and the high weight to WMT. As can be seen from the last two lines, this *minimum risk* portfolio has a lower expected return than the equal weight portfolio.

Suppose now the investor is interested in the most risk diversified portfolio whose expected return is higher than the equal-weight portfolio. This amounts to minimizing the largest CVaR contribution subject to a return target and can be implemented as follows:

```
> obj <- function(w) {
+   if(sum(w) == 0) { w <- w + 1e-2 }
+   w <- w / sum(w)
+   contribCVaR <- ES(weights = w,
+     method = "gaussian",
+     portfolio_method = "component",
+     mu = mu,
+     sigma = sigma)$contribution
+   tmp1 <- max(contribCVaR)
+   tmp2 <- max(mean(mu) - sum(w * mu), 0)
+   out <- tmp1 + 1e3 * tmp2
+ }
> out <- DEoptim(fn = obj,
+   lower = rep(0, 5),
+   upper = rep(1, 5))
> wstar <- out$optim$bestmem
> wstar <- wstar / sum(wstar)
> rbind(tickers, round(wstar, 3))
        par1    par2    par3    par4    par5
tickers "GE"    "IBM"   "JPM"   "MSFT"  "WMT"
        "0.163" "0.212" "0.163" "0.178" "0.285"
> sum(wstar * mu) - mean(mu)
[1] 5.545333e-08
```

This portfolio invests more in the JPM stock and less in the GE (which has the lowest average return) compared to the portfolio with the upper 22.5% percentage CVaR constraint. We refer to Boudt et al. (2010a) for a more elaborate study on using CVaR allocations as an objective function or constraint in the portfolio optimization problem.

A classic risk/return (i.e., CVaR/mean) scatter chart showing the results for portfolios tested by `DEoptim` is displayed in Figure 3. Gray elements denote the results for all tested portfolios. The blue line shows the path of the best member of the population over time, with the darkest solution at the end being the *optimal* portfolio. Note how `DEoptim` does not spend much time computing solutions in the scatter space that are suboptimal, but concentrates the bulk of the calculation time in the vicinity of the final *best* portfolio.

One of the key issues in practice with real portfolios is that a portfolio manager rarely has only a single objective or only a few simple objectives combined. For many combinations of objectives, there is no global optimum, or the constraints and objectives formed lead to a non-convex search space. It may take several hours on very fast machines to get the best answers, and the best answers may not be a true global optimum, they are just *as close as feasible* given potentially competing and contradictory objectives.

When the constraints and objectives are relatively simple, and may be reduced to quadratic, linear, or conical forms, a simpler optimization solver will produce answers more quickly. When the objectives are more layered, complex, and potentially contradictory, as those in real portfolios tend to be, **DEoptim** or a pure random portfolio space as those integrated into **PortfolioAnalytics** provide a portfolio manager with a feasible option for optimizing their portfolio under real-world non-convex constraints and objectives. The **PortfolioAnalytics** framework allows any arbitrary R function to be part of the objective set, and allows the user to set the relative weighting that they want on any specific objective, and use the appropriately tuned optimization solver algorithm to locate portfolios that most closely match those objectives.

Summary

In this note we have introduced DE and **DEoptim**. The package **DEoptim** provides a means of applying the DE algorithm in the R language and environment for statistical computing. DE and the package **DEoptim** have proven themselves to be powerful tools for the solution of global optimization problems in a wide variety of fields. We have referred interested users to Price et al. (2006); Lampinen (2009) and Mullen et al. (2009) for a more extensive introduction, and further pointers to the literature on DE. The

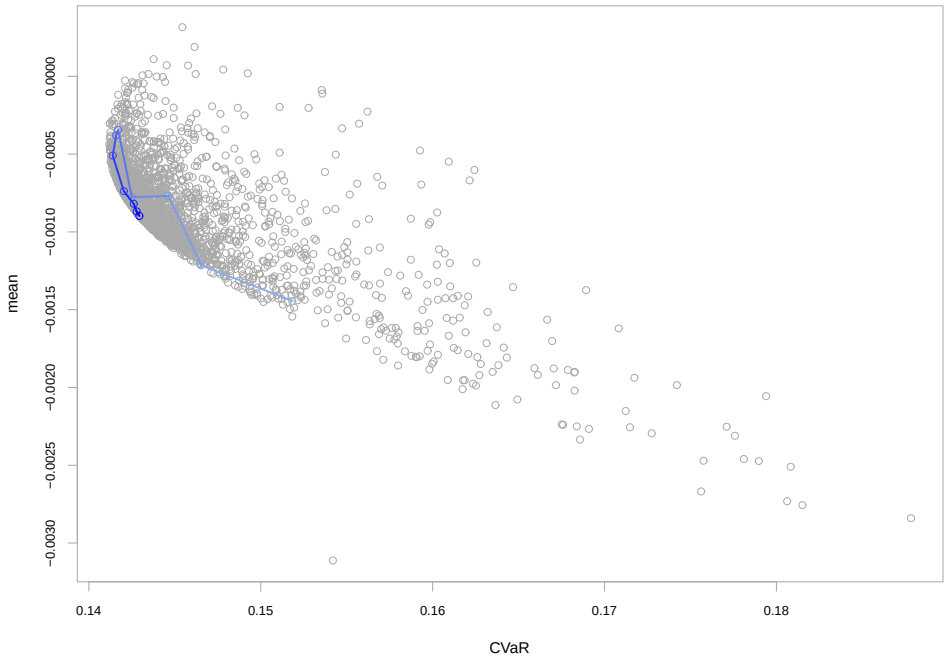

Figure 3: Risk/return scatter chart showing the results for portfolios tested by `DEoptim`.

utility of using **DEoptim** was further demonstrated with a simple example of a stylized non-convex portfolio risk contribution allocation, with users referred to **PortfolioAnalytics** for portfolio optimization using DE with real portfolios under non-convex constraints and objectives.

Acknowledgements

The authors would like to thank J.D. Ospina Arango for providing them with the code of the Rastrigin example and R. Storn for his advocacy of DE and making his code publicly available. Kris Boudt gratefully acknowledges financial support from the National Bank of Belgium.

Disclaimer

The views expressed in this note are the sole responsibility of the authors and do not necessarily reflect those of aeris CAPITAL AG, Guidance Capital Management, Breakwater Trading LLC, or any of its affiliates.

Bibliography

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

J. Börner, S. I. Higgins, J. Kantelhardt, and S. Scheiter. Rainfall or price variability: What determines rangeland management decisions? a simulation-optimization approach to South African savanas. *Agricultural Economics*, 37(2–3):189–200, Sept.–Nov. 2007.

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://r-forge.r-project.org/projects/returnanalytics/. R package version 0.3.

R. Cao, J. M. Vilar, and A. Devia. Modelling consumer credit risk via survival analysis. *Statistics & Operations Research Transactions*, 33(1):3–30, Jan.-June 2009.

P. Carl and B. G. Peterson. *PerformanceAnalytics: Econometric tools for performance and risk analysis.*, 2010. URL http://r-forge.r-project.org/projects/returnanalytics/. R package version 1.0.2.

M. Gilli and E. Schumann. Heuristic optimisation in financial modelling. COMISEF wps-007 09/02/2009, 2009.

M. Gilli and P. Winker. A review of heuristic optimization methods in econometrics. Swiss Institute Research paper series 08-12, Dec. 2008.

M. Gilli, D. G. Maringer, and P. Winker. Applications of heuristics in finance. In D. Schlottmann, C. Weinhardt, and F. Schlottmann, editors, *Handbook on Information Technology in Finance*, chapter 26. Springer-Verlag, Berlin, Heidelberg, 2008.

S. I. Higgins, J. Kantelhardt, S. Scheiter, and J. Börner. Sustainable management of extensively managed savanna rangelands. *Ecological Economics*, 62(1): 102–114, Apr. 2007.

J. H. Holland. *Adaptation in Natural Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

T. Krink and S. Paterlini. Multiobjective optimization using differential evolution for real-world portfolio optimization. *Computational Management Science*, 2009.

T. Krink, S. Mittnik, and S. Paterlini. Differential evolution and combinatorial search for constrained index-tracking. *Annals of Operations Research*, 172: 153–176, 2009.

J. A. Lampinen. A bibliography of differential evolution algorithm, 2009. URL `http://www2lutfi/~jlampine/debibliohtm`.

D. G. Maringer. Portfolio management with heuristic optimization. In *Advanced in Computational Management Science*, volume 8 of *Advances in Computational Management Science*, chapter 14. Springer-Verlag, 2005.

D. G. Maringer and M. Meyer. Smooth transition autoregressive models: New approaches to the model selection problem. *Studies in Nonlinear Dynamics & Econometrics*, 12(1):1–19, Jan. 2008. URL `http://www.bepress.com/snde/vol12/iss1/`. Article nr. 5.

D. G. Maringer and O. Oyewumi. Index tracking with constrained portfolios. *Intelligent Systems in Accounting, Finance & Management*, 15(1–2):57–71, 2007.

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.

J. D. Opsina Arango. Estimacion de un modelo de difusion con saltos con distribucion de error generalizada asimetrica usando algorithmos evolutivos. Master's thesis, Universidad Nacional de Colombia, 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.

O. Scaillet. Nonparametric estimation and sensitivity analysis of expected shortfall. *Mathematical Finance*, 14(1):74–86, 2002.

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.

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

David Ardia
aeris CAPITAL AG, Switzerland
`da@aeris-capital.com`

Kris Boudt
Lessius and K.U.Leuven, Belgium
`kris.boudt@econ.kuleuven.be`

Peter Carl
Guidance Capital Management, Chicago, IL
`pcarl@gsb.uchicago.edu`

Katharine M. Mullen
National Institute of Standards and Technology
Gaithersburg, MD
`katharine.mullen@nist.gov`

Brian G. Peterson
Breakwater Trading LLC, Chicago, IL
`brian@braverock.com`