DEoptim: An R Package for Global Optimization by Differential Evolution

Katharine M. Mullen
NIST

David Ardia
aeris CAPITAL AG

David L. Gil
NIST

Donald Windover
NIST

James Cline
NIST

Abstract

This article describes the R package **DEoptim** which implements the differential evolution algorithm for the global optimization of a real-valued function of a real-valued parameter vector. The implementation of differential evolution in **DEoptim** interfaces with C code for efficiency. The utility of the package is illustrated via case studies in fitting a Parratt model for X-ray reflectometry data and a Markov-Switching Generalized AutoRegressive Conditional Heteroskedasticity (MSGARCH) model for the returns of the Swiss Market Index.

Keywords: global optimization, evolutionary algorithm, differential evolution, R software.

1. Introduction

Optimization algorithms inspired by the process of natural selection have been in use since the 1950s (Mitchell 1998). John Holland invented the genetic algorithm in the 1960s, which uses biology-inspired operations of crossover, mutation, and selection on a population comprised of bit strings in order to minimize an objective function over the course of successive generations (Holland 1975). Genetic algorithms proved themselves to be useful heuristic methods for global optimization, in particular for combinatorial optimization problems. In the 1990s Rainer Storn and Kenneth Price developed a variation on genetic algorithms they termed *differential evolution* (DE) (Storn and Price 1997). DE uses floating-point instead of bit-string encoding of population members, and arithmetic operations instead of logical operations in mutation. DE is particularly well-suited to find the global optimum of a real-valued function of real-valued parameters, and does not require that the function be either continuous or differentiable. In the roughly fifteen years since its invention, DE has been successfully applied in a wide variety of fields, from computational physics to operations research, as Price, Storn, and Lampinen (2006) catalogue.

Many implementations of DE are currently available. A web-based list of DE programs for general purpose optimization is maintained by Rainer Storn at `http://www.icsi.berkeley.edu/~storn/code.html`. Programs from this list for which the source code is readily available are summarized in Table 1. Commercial software such as Mathematica, MATLAB's GA toolbox, and a variety of special-purpose programs for optical and X-ray physics also implement

Program	Language	Authors	Cross-platform
DeApp	java	Storn	Yes
DeWin	MS Visual C++	Storn	No
DeMat	MATLAB	Storn	No
DiffEvol	scilab	Di Carlo & Jarausch	Yes
DESolver	MS Visual C++	Godwin	No
DE_Fortran90	Fortran 90	Wang	Yes
DeMat for Pascal	Pascal	Geldon & Gauden	Yes
DEoptim	R	Ardia & Mullen	Yes

Table 1: Implementations of differential evolution for general purpose optimization.

DE[1].

The **DEoptim** implementation of DE was motivated by our desire to extend the set of algorithms available for global optimization in the R language and environment for statistical computing (R Development Core Team 2009). R enables rapid prototyping of objective functions, access to a wide array of tools for statistical modeling, and ability to generate customized plots of results with ease (which in many situations makes use of R preferable over the use of compiled programs in languages like java, MS Visual C++, Fortran 90 or Pascal). Furthermore, R is released in open-source form under the terms of the GNU General Public License, meaning that packages implemented for it do not require the purchase of commercial software. R also has a large and growing user base interested in optimization. **DEoptim** has been published on the Comprehensive R Archive Network and is available at `http://cran.r-project.org/web/packages/DEoptim/`. Since becoming publicly available it has been used by a variety of authors, e.g., Börner, Higgins, Kantelhardt, and Scheiter (2007), Higgins, Kantelhardt, Scheiter, and Boerner (2007), Cao, Vilar, and Devia (2009), and Opsina Arango (2009), to solve optimization problems arising in diverse domains.

In the remainder of this manuscript we elaborate on **DEoptim**'s implementation and use. In Section 1.1, the package is introduced via a simple example. Section 2 describes the underlying algorithm. Section 3 describes the R implementation and functions as a user manual. **DEoptim** is then illustrated via two cases studies, involving fitting a Parratt recursion model for X-ray reflectometry data (in Section 4) and a Markov-Switching Generalized Autoregressive Conditional Heteroscedasticity(MSGARCH) model for log-returns of the Swiss Market Index (in Section 5). Section 6 contains a summary and conclusions.

1.1. An introductory example

Minimization of the Rastrigin function of $x \in \Re^D$

$$f(x) = \sum_{j=1}^{D} \left(x_j^2 - 10\cos\left(2\pi x_j\right) + 10\right)$$

[1]Certain commercial equipment, instruments, or materials are identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

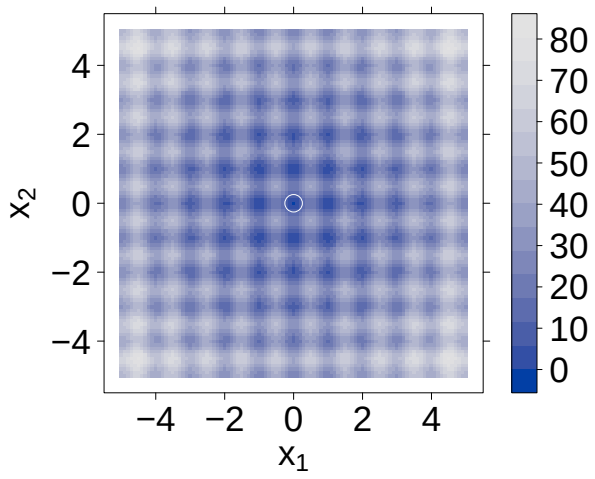

Figure 1: A contour plot of the two-dimensional Rastrigin function $f(x)$. The global minimum $f(x) = 0$ is at $(0, 0)$ and is marked with an open white circle.

for $D = 2$ is a common test for global optimization algorithms.

This function is possible to represent in R as

```
R> rastrigin <- function(x) 10 * length(x) + sum(x^2 - 10 * cos(2 *
+     pi * x))
```

As shown in Figure 1, for $D = 2$ the function has a global minimum $f(x) = 0$ at the point $(0, 0)$.

In order to minimize this function using **DEoptim**, the R interpreter is invoked, and the package is loaded with the command

```
R> library("DEoptim")
```

`DEoptim` searches for minima of the objective function between lower and upper bounds on each parameter to be optimized. Therefore in the call to `DEoptim` we specify vectors that comprise the lower and upper bounds; these vectors are the same length as the parameter vector. The call to `DEoptim` can be made as

```
R> est.ras <- DEoptim(rastrigin, lower = c(-5, -5), upper = c(5,
+     5), control = list(storepopfrom = 1, trace = FALSE))
```

Note that the vector of parameters to be optimized must be the first argument of the objective function `fn` passed to `DEoptim`. The above call specifies the objective function to minimize, `rastrigin`, the lower and upper bounds on the the parameters, and, via the `control` argument, that we want to store intermediate populations from the first generation onwards (`storepopfrom = 1`), and do not want to print out progress information each generation

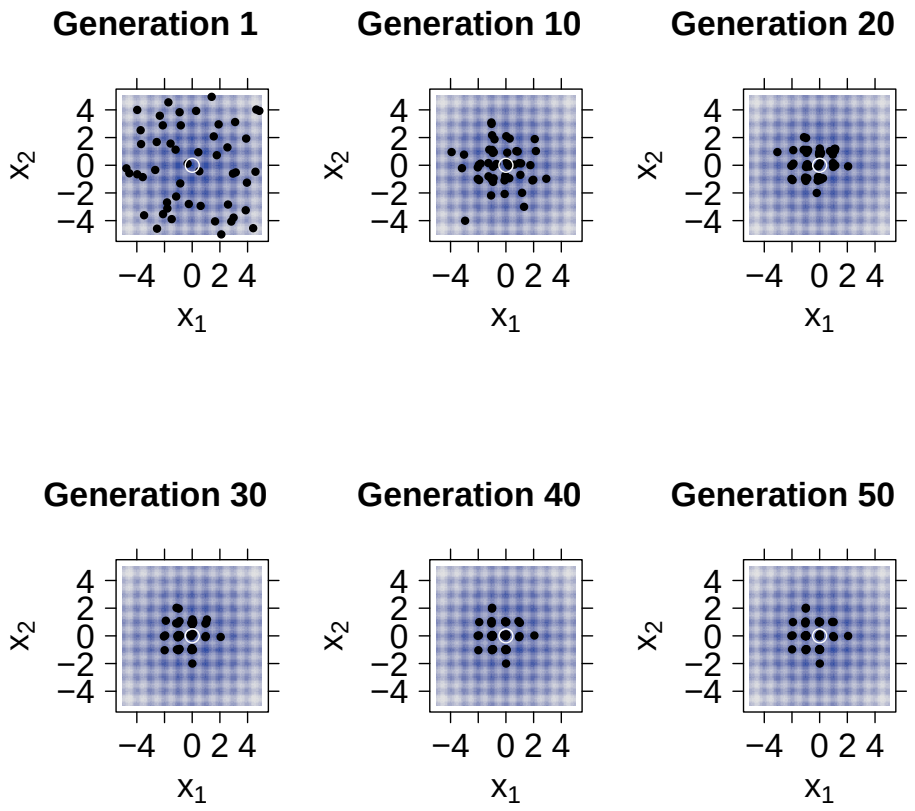

Figure 2: The population associated with various generations of a call to `DEoptim` as it searches for the minimum of the Rastrigin function (marked with an open white circle). The minimum is consistently determined within 200 generations using the default settings of `DEoptim`.

(`trace = FALSE`). Storing intermediate populations allows us to examine the progress of the optimization in detail. Upon initialization, the population is comprised of 50 vectors x of length two, with x_i a random value drawn from the uniform distribution over the values defined by the associated lower and upper bound. The operations of crossover, mutation, and selection explained in Section 2 transform the population so that the members of successive generations are more likely to represent the global minimum of the objective function. 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 Rastrigen problem more efficiently than the simulated annealing method found in the R function `optim` (for all annealing schedules we have tried).

2. The differential evolution algorithm

We sketch the classical DE algorithm here and refer interested readers to the work of Storn

and Price (1997) and Price *et al.* (2006) for details. The algorithm is an evolutionary technique which at each generation transforms a set of parameter vectors termed the population into another set of parameter vectors, the members of which are more likely to minimize the objective function. The variable NP represents the number of parameter vectors in the population. At generation 0, NP guesses for the optimal value of the parameter vector are made, either using random values between upper and lower bounds for each parameter or using values given by the user. Each generation involves creation of a new population from the current population members, where g indexes generation, i indexes the vectors that make up the population, and j indexes the values in each population member vector. This is accomplished using *differential mutation* of the population members. A mutant parameter vector $v_{i,g}$ is created by choosing three members of the population, $x_{r0,g}$, $x_{r1,g}$ and $x_{r2,g}$, at random. Then $v_{i,g}$ is generated as

$$v_{i,g} = x_{r0,g} + \mathrm{F} * (x_{r1,g} - x_{r2,g}) \tag{1}$$

where F is a positive scale factor. Effective values of F are typically less than 1.

After the first mutation operation, mutation is continued until either length(x) mutations have been made or rand $> CR$, where CR is a crossover probability $CR \in [0, 1]$, and where here and throughout rand is used to denote a random number from $U(0, 1)$. The crossover probability CR controls the fraction of the parameter values that are copied from the mutant. CR approximates but does not exactly represent the probability that a parameter value will be inherited from the mutant, since at least one mutation always occurs.

If an element v_j of the parameter vector is found to violate the bounds after mutation and crossover, it is reset. In the implementation **DEoptim**, if $v_j > \mathrm{upper}_j$, it is reset as $v_j = \mathrm{upper}_j - \mathrm{rand} * (\mathrm{upper}_j - \mathrm{lower}_j)$, and if $v_j < \mathrm{lower}_j$, it is reset as $v_j = \mathrm{lower}_j + \mathrm{rand} * (\mathrm{upper}_j - \mathrm{lower}_j)$. Then the objective function values associated with the children v are determined. If a trial vector $v_{i,g}$ has equal or lower objective function value than the vector $x_{i,g}$, $v_{i,g}$ replaces $x_{i,g}$ in the population; otherwise $x_{i,g}$ remains. Variations on this theme are possible, some of which are described in the following section. Values of rand and CR that have been found to be most effective for a variety of problems are described in Price *et al.* (2006). Reasonable default values for many problems are given in the following section.

3. Implementation

DEoptim was first published on the Comprehensive R Archive Network (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.

The implementation is used by calling the R function `DEoptim`, the arguments of which are:

- `fn`: The objective function to be minimized. This function should have as its first argument the vector of real-valued parameters to optimize, and return a scalar real result.

- `lower`, `upper`: Vectors specifying scalar real lower and upper bounds on each parameter to be optimized, so that the ith element of `lower` and `upper` applies to the ith parameter. The implementation searches between `lower` and `upper` for the global optimum of `fn`.
- `control`: A list of control parameters, discussed below.

The `control` argument is a list, the following elements of which are currently interpreted:

- `VTR`: The value to reach. Specify the global minimum of `fn` if it is known, or if you wish to cease optimization after having reached a certain value. The default value is `-Inf`.
- `strategy`: This defines the differential evolution strategy used in the optimization procedure, described below in the terms used by Price *et al.* (2006):
  - `1`: DE / rand / 1 / bin (classical strategy). This strategy is the classical approach described in Section 2.
  - `2`: DE / local-to-best / 1 / bin. In place of the classical DE mutation given in (1), the expression

    $$v_{i,g} = \text{old}_{i,g} + (\text{best}_g - \text{old}_{i,g}) + x_{r0,g} + \text{F} * (x_{r1,g} - x_{r2,g})$$

    is used, where $\text{old}_{i,g}$ and best_g are the ith member and best member, respectively, of the previous population. This strategy is currently used by default.
  - `3`: DE / best / 1 / bin with jitter. In place of the classical DE mutation given in (1), the expression

    $$v_{i,g} = \text{best}_g + \text{jitter} + \text{F} * (x_{r1,g} - x_{r2,g})$$

    is used, where jitter is defined as $0.0001 * \text{rand} + F$.
  - `4`: DE / rand / 1 / bin with per-vector-dither. In place of the classical DE mutation given in (1), the expression

    $$v_{i,g} = x_{r0,g} + \text{dither} * (x_{r1,g} - x_{r2,g})$$

    is used, where dither is calculated as $\text{dither} = F + \text{rand} * (1 - F)$.
  - `5`: DE / rand / 1 / bin with per-generation-dither. The strategy described for `4` is used, but dither is only determined once per-generation.
  - any value not above: variation to DE / rand / 1 / bin: either-or-algorithm. In the case that $\text{rand} < 0.5$, the classical strategy described for `1` is used. Otherwise, the expression

    $$v_{i,g} = x_{r0,g} + 0.5 * (F + 1.0) * (x_{r1,g} + x_{r2,g} - 2 * x_{r0,g})$$

    is used.
- `bs`: If `FALSE` then every mutant will be tested against a member in the previous generation, and the best value will survive into the next generation. This is the standard trial vs. target selection described in Section 2. If `TRUE` then the old generation and `NP` mutants will be sorted by their associated objective function values, and the best `NP` vectors will proceed into the next generation (this is best of parent and child selection). The default value is `FALSE`.

- `NP`: Number of population members. The default value is `50`.
- `itermax`: The maximum iteration (population generation) allowed. The default value is `200`.
- `CR`: Crossover probability from interval [0,1]. The default value is `0.9`.
- `F`: Stepsize from interval [0,2]. The default value is `0.8`.
- `trace`: Logical indicating whether printing of progress occurs at each iteration. The default value is `TRUE`.
- `initialpop`: An initial population used as a starting population in the optimization procedure, specified as a matrix in which each row represents a population member. May be useful to speed up the convergence. Default to `NULL`, so that the initial population is generated randomly within the lower and upper boundaries.
- `storepopfrom`: From which population should the following intermediate populations be stored in memory. Default to `itermax+1`, i.e., no intermediate population is stored.
- `storepopfreq`: The frequency of populations' storage. The default value is `1`, i.e. every intermediate population is stored.
- `checkWinner`: Logical value indicating whether to re-evaluate the objective function using the winning parameter vector if this vector remains the same between generations. This may be useful for the optimization of a noisy objective function. If `checkWinner=TRUE` and `avWinner=FALSE` then the value associated with re-evaluation of the objective function is used in the next generation. Default to `FALSE`.
- `avWinner`: Logical value. If `checkWinner=TRUE` and `avWinner=TRUE` then the objective function value associated with the winning member represents the average of all evaluations of the objective function over the course of the 'winning streak' of the best population member. This option may be useful for optimization of noisy objective functions, and is interpreted only if `checkWinner=TRUE`. The default value is `TRUE`.

The default value of `control` is the return value of `DEoptim.control()`, which is a list (and a member of the S3 class `DEoptim.control`) with the above elements and specified default values.

The return value of the `DEoptim` function is a member of the S3 class `DEoptim`. Members of this class have a `plot` method that accepts the argument `plot.type`. When `retVal` is an object returned by **DEoptim**, calling `plot(retVal, plot.type = "bestmemit")` results in a plot of the parameter values that represent the lowest value of the objective function each generation. Calling `plot(retVal, plot.type = "bestvalit")` plots the best value of the objective function each generation. Calling `plot(retVal, 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). A summary method for objects of S3 class `DEoptim` also exists, and returns the best parameter vector, the best value of the objective function, the number of generations optimization ran, and the number of times the objective function was evaluated.

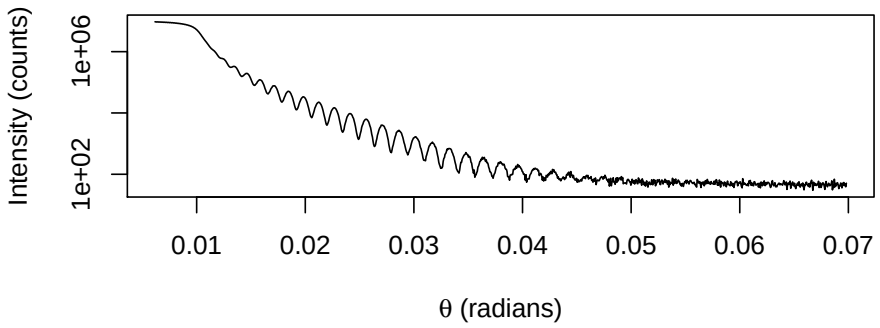

Figure 3: XRR measurements of Pt layers on SiO_2 substrate.

4. Application I: X-ray reflectometry

X-ray reflectometry (XRR) is a measurement method that uses the interference of X-rays (i.e., photons with a wavelength in the approximate range of 0.01 nm–10 nm) caused by changes in a material's electron density to characterize thin films or other layered structures at the nanometer to micrometer scale. The data collected consists of pairs of incident/scattered angle and scattered X-ray intensities, $\{(\theta_k, I_k)\}$, typically over a range of about 5 degrees. Information regarding the density and thickness of each layer, and on the roughness of the interface between layers and at the surface of the material is extracted by fitting a parametric model to the measurements.

In the supplementary information we provide the full description of a model function used by **DEoptim** to obtain physically realistic parameter estimates from the data shown in Figure 3. This model is based on the Parratt recursion (Parratt 1954), which as Als-Nielsen and McMorrow (2001) describe in detail, is often used to model each of the layers in the multi-layered materials. For the data here, the Parratt recursion is used to describe reflection and transmission of X-rays from two thin layers of Pt (with each layer having a possibly distinct thickness density, and roughness at the interface) atop an infinitely thick layer of SiO_2. A schematic description of the model for this multilayered material is depicted in Figure 4.

The free parameters of the applied Parratt recursion model are the thicknesses d_1 and d_2 of each Pt layer, the density den_1 and den_2 of each Pt layer, terms r_1, r_2 and r_3 descriptive of the roughness of the interfaces between layers and at the surface, a parameter b describing a linear background, and a multiplicative scaling parameter m. The model function can be understood qualitatively by considering the the case of a single layer on a substrate. For this case the position of the abrupt drop-off in scattered intensity after the initial plateau is determined by the density of the layer. The period of the subsequent oscillation fringes is determined by the thickness of the layer, whereas the point at which the oscillations fade is determined by the roughness of the layer. The Parratt recursion modifies this description by taking into account scattering and transmission from each layer.

The objective function R to minimize is formulated as the sum of the squared differences between the log of the data and the log of the Parratt recursion model function. The surface

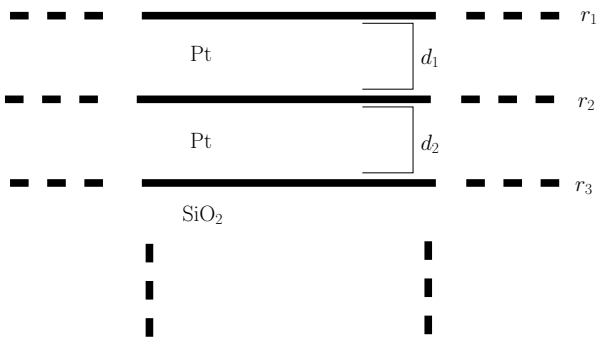

Figure 4: Schematic description of two layers of Pt on a substrate of SiO_2. A Parratt recursion model representing this structure will be fit to the XRR measurements, with free parameters including the thickness of the Pt layers (d_1 and d_1), and terms (r_1, r_2, and r_3) describing the roughness of the interfaces between layers.

comprised of objective function values in the 9-dimensional parameter space contains many local minima. Discovery of parameter estimates that represent a qualitatively good fit requires a global optimization algorithm such as DE. Treatment of global optimization problems such as these have been successfully addressed for many years in the XRR community using DE as, e.g., Wormington, Panaccione, Matney, and Bowen (1999), Taylor, Wall, Loxley, Wormington, and Lafford (2001), and Bowen and Tanner (2006) describe. Special purpose programs, e.g., the *GenX* program developed by Björck and Andersson (2007) and the *MOTOFIT* program developed by Nelson (2006), have been implemented for XRR model fitting problems of this form.

The XRR measurements shown in Figure 3 are included in **DEoptim** as the dataset `xrrData`, with the vector of data to be fit represented by the vector `counts`. We have encoded the objective function R as the function `rss`. Using knowledge of the physical system underlying the measurements in order to set plausible lower and upper bounds on the parameters to optimize, and to set fixed values for `beta`, `wavelength`, and `delta`, objective function is minimized with the call

```
R> parrattFit <- DEoptim(lower = c(d_1 = 5.5e-10, d_2 = 1.5e-08,
+                          r_1 = 2.1e-10, r_2 = 5.0e-12, r_3 = 2.2e-10,
+                          den_1 = 10, den_2 = 10, b = 40, m = .90e7),
+                        upper = c(d_1 = 5.5e-09, d_2 = 1.5e-07,
+                          r_1 = 2.1e-09, r_2 = 5.0e-11, r_3 = 2.2e-09,
+                          den_1 = 25, den_2 = 25, b = 55,  m = 1.1e7),
+                      fn = rss, theta_r = theta_r, delta = delta,
+                      beta = beta, wavelength = wavelength, data = counts,
+                      control = list(itermax = 1500, NP = 90))
```

Table 2 gives parameter estimates arrived at via the above call, along with the associated

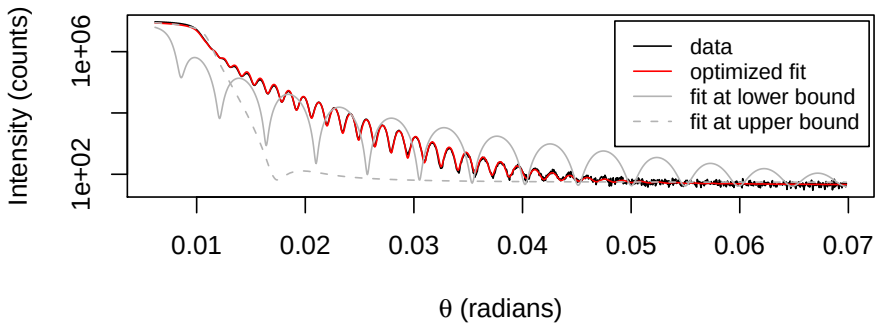

Figure 5: XRR measurements (black) of Pt layers on SiO_2 substrate with model fit (red). For comparison, the model has also been evaluated at the lower and upper bounds on the parameters used in the call to `DEoptim` (solid and dashed grey, respectively).

	d_1	d_2	r_1 / nm	r_2	r_3	den_1 / g·cm^{-3}	den_2	b / counts	m
`lower`	0.55	15.0	0.21	0.005	0.22	10.0	10.0	40.0	0.90e7
`upper`	5.50	150.0	2.10	0.050	2.20	25.0	25.0	55.0	1.20e7
`bestmem`	0.918	46.4	0.76	0.0073	0.69	23	22	43.0	1.0e7

Table 2: Parameter estimates (`bestmem`) and lower and upper bounds associated with the call to **DEoptim** that results in the fit of Parratt recursion model to XRR data shown in Figure 5. Parameters d_1 and d_2 represent the thickness of the Pt layers, parameters r_1, r_2, r_3 describe the roughness of the interfaces between layers, and parameters den_1 and den_2 represent the density of the Pt layers. Parameter b represents an additive background term, and parameter m represents a multiplicative scaling factor for the intensity. Estimates are reported to two significant figures, except for t_1 and t_2, which are reported to three.

lower and upper bounds. The resulting fit of the model to the data is shown in Figure 5. The thickness, density and roughness estimates are plausible from physical first principles, and well-approximate estimates obtained via examination of the material with other measurement methods. As shown in Figure 5, the model fit also captures the qualitative features of the dataset well. The robustness of the estimates has been furthermore validated via initialization of DE using a variety of starting populations; the estimates presented in Table 2 reliably represent the best results obtained.

The function `rss` encoding the objective function can easily be customized to the dataset at hand, allowing, for instance, inclusion of more or fewer free parameters. Note that in this example the population size `NP` was set to 90 since in practice it has been observed that convergence to the global optimum is facilitated if `NP` is at least ten times the number of parameters being optimized (Price *et al.* 2006).

5. Application II: Log-returns of the Swiss Market Index

In this section, we address in some detail the problem of estimating the parameters of MS-GARCH models, which are GARCH models subject to structural changes in the parameters. In the MSGARCH framework, a hidden Markov sequence $\{s_t\}$ with discrete state space $\{1, \ldots, K\}$ allows discrete changes in the model parameters.

MSGARCH models have received much attention in recent years as they provide an explanation for the high persistence in volatility observed in single-regime GARCH models (see e.g., Lamoureux and Lastrapes 1990). Furthermore, these models allow for a sudden change in the (unconditional) volatility level which may lead to significant improvements in volatility forecasts (Dueker 1997). These features make the models attractive for various applications in financial modeling, such as risk management.

While MSGARCH models are attractive for the description of a variety of phenomena, we face practical difficulties when attempting to fit their parameters to data. The maximization of the likelihood function is a constrained optimization problem since some (or all) of the model parameters must be positive to ensure a positive conditional variance. It is also common to require that the covariance stationarity condition holds; this leads to additional complicated non-linear inequality constraints which render the optimization procedure cumbersome. Optimization results are often sensitive to the choice of starting values. Finally, convergence is hard to achieve if the true parameter values are close to the boundary of the parameter space and if the underlying process is nearly non-stationary. For these reasons, a robust optimizer is required. DE offers an adequate approach to finding the maximum likelihood parameter estimates in this framework.

In order to illustrate the robustness of `DEoptim` compared to traditional estimation techniques, we consider the asymmetric MSGARCH model investigated in Ardia (2008, chapter 7). The author illustrated the poor performance of traditional local optimizers when estimating such sophisticated models. Only computationally demanding MCMC techniques were able to provide meaningful results.

A K-regime Markov-switching asymmetric GARCH(1,1) model with Student-t innovations for the log-returns $\{y_t\}$ may be written as

$$\begin{aligned} y_t &= \varepsilon_t \sqrt{\tfrac{\nu-2}{\nu}}\, \sigma_{s_t,t} \quad t = 1, \ldots, T \\ \varepsilon_t &\overset{i.i.d.}{\sim} \mathcal{S}(0, 1, \nu) \\ \sigma^2_{i,t} &= \omega_i + \left(\alpha_i^+ 1_{\{y_{t-1} \geq 0\}} + \alpha_i^- 1_{\{y_{t-1} < 0\}}\right) y^2_{t-1} + \beta_i\, \sigma^2_{i,t-1}\,, \end{aligned} \tag{2}$$

where $\omega_i > 0$, $\alpha_i^+, \alpha_i^-, \beta_i \geq 0$ $(i = 1, \ldots, K)$ and $\nu > 2$. The restriction on the degrees of freedom parameter ensures that the conditional variance $\sigma^2_{i,t}$ remains finite; the restrictions on the GARCH parameters guarantee its positivity. $1_{\{\cdot\}}$ denotes the indicator function which is equal to one if the constraint holds and zero otherwise. The sequence $\{s_t\}$ is assumed to be a stationary, irreducible Markov process with discrete state space $\{1, \ldots, K\}$ and transition matrix $P = [p_{ij}]$ where $p_{ij} = \mathsf{P}(s_{t+1} = j \,|\, s_t = i)$. Finally, $\mathcal{S}(0, 1, \nu)$ denotes the standard Student-t density with ν degrees of freedom and $\sqrt{(\nu-2)/\nu}$ is a scaling factor which ensures that the conditional variance is given by $\sigma^2_{s_t,t}$.

Model specification (2) allows reproduction of the so-called *volatility clustering* observed in financial returns, i.e., the fact that large changes tend to be followed by large changes (of either

sign) and small changes tend to be followed by small changes. Moreover, it allows for sudden changes in the unconditional variance of the process; in the ith regime, the unconditional variance is

$$\frac{\omega_i}{1 - (\alpha_i^+ + \alpha_i^-)/2 - \beta_i},$$

provided that $(\alpha_i^+ + \alpha_i^-)/2 + \beta_i < 1$ (i.e., the process is covariance stationary). Finally, it allows determination of whether or not an asymmetric response, referred to as the leverage effect in the financial literature, is present (i.e., $\alpha_i^- > \alpha_i^+$ for at least one i) and is different between the regimes (i.e., $\alpha_i^- \neq \alpha_{i'}^-$).

The use of a Student-t instead of a Normal distribution is quite popular in standard single-regime GARCH literature. For Markov-switching models, a Student-t distribution might be seen as superfluous since the switching regime can account for large unconditional kurtosis in the data. However, as empirically observed by Klaassen (2002), allowing for Student-t innovations within regimes can enhance the stability of the states and emphasizes the conditional variance's behavior instead of outliers.

To illustrate the utility of **DEoptim**, we fit a two-regime ($K = 2$) asymmetric MSGARCH model to daily log-returns of the Swiss Market Index (SMI). The sample period is from November 12, 1990, to October 20, 2000, for a total of 2500 observations and the log-returns are expressed in percent. The data set was downloaded from `http://www.finance.yahoo.com` and is available if **DEoptim** is loaded using the command `data(SMI)`. Note that the two-regime specification is used for illustrative purposes only; checking for possible model misspecification is beyond the scope of the present paper.

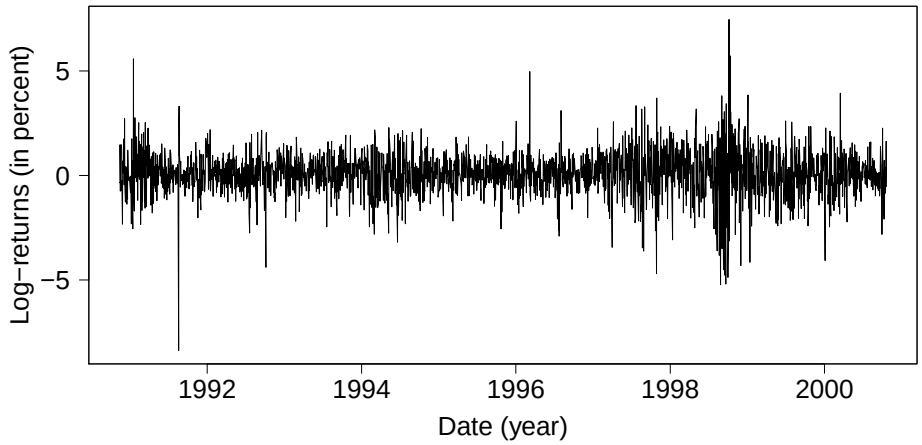

Figure 6: SMI daily log-returns.

In addition to the positivity constraints on the model parameters, we require covariance stationarity to hold in the two regimes, i.e., $(\alpha_i^+ + \alpha_i^-)/2 + \beta_i < 1$ for $i = 1, 2$. We also require the transition probabilities p_{11} and p_{22} of the state variable to lie within the $[0, 1]$ interval. The constraints on the domain are set using the arguments `lower` and `upper` of `DEoptim` while the covariance stationarity constraints are tested within the objective function which then returns `Inf` if not satisfied. The eleven parameters are regrouped into the vector $\theta = (\omega_1, \omega_2, \alpha_1^+, \alpha_2^+, \alpha_1^-.\alpha_2^-, \beta_1, \beta_2, p_{11}, p_{22}, \nu)$ for notational purposes.

In order to find the maximum likelihood estimator $\hat{\theta}_{\text{mle}}$ we minimize the negative value of the natural logarithm of the likelihood function (NLL) corresponding to the model (2) with $K = 2$. The likelihood function of the MSGARCH model is obtained by applying the filtering approach of Hamilton (1989). The objective function is implemented in C to speed up the optimization procedure. `DEoptim` is run with the default parameters except that we set `itermax = 400` and `NP = 110`.

For comparison purposes, the objective function is also optimized using various other functions available in R. More specifically, we use the function `optim` with all methods available and the function `nlminb`, employing the default values of the control parameters of all the methods (except for `optim` with `method = SANN` where we set `itermax = 1e5`). Estimation results are reported in Table 3. All methods except `DEoptim` converged to local minima (various starting values were used, all leading to local optima); NLL is clearly lower than the value obtained by DE. Also, notice that the parameter estimates are different between the optimization methods. In the case `optim` with `method = "L-BFGS-B"` the algorithm did not converge (since in this approach the objective function must return finite values).

Method	NLL	$\hat{\omega}_1$	$\hat{\omega}_2$	$\hat{\alpha}_1^+$	$\hat{\alpha}_2^+$	$\hat{\alpha}_1^-$	$\hat{\alpha}_2^-$	$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{p}_{11}$	$\hat{p}_{22}$	$\hat{\nu}$
`optim`[1]	3384.0	0.004	0.094	0.024	0.055	0.045	0.421	0.963	0.745	0.366	0.387	5.37
`optim`[2]	3380.6	0.042	0.042	0.041	0.041	0.165	0.165	0.862	0.862	0.500	0.500	8.41
`optim`[3]	3387.7	0.044	0.044	0.046	0.046	0.177	0.177	0.864	0.864	0.500	0.500	5.33
`optim`[4]	n.c.	-	-	-	-	-	-	-	-	-	-	-
`optim`[5]	3381.1	0.211	0.037	0.205	0.026	0.231	0.184	0.192	0.878	0.916	0.998	5.10
`nlminb`	3374.3	0.000	0.093	0.024	0.033	0.047	0.301	0.964	0.758	0.297	0.555	9.12
`DEoptim`	3353.6	0.332	0.190	0.003	0.007	0.275	0.228	0.262	0.786	0.995	0.997	9.40

Table 3: Optimization results of the asymmetric MSGARCH model estimation. NLL: negative log-likelihood function at optimum. n.c.: non-convergence of the algorithm. `optim`: output of the function `optim` with `method`: [1]`"Nelder-Mead"`, [2]`"BFGS"`, [3]`"CG"`, [4]`"L-BFGS-B"`, [5]`"SANN"` with control parameter `itermax = 1e5`. `DEoptim`: DE optimization with control parameters `NP = 110` and `itermax = 400`. Starting values for `optim` and `nlminb` were set to $\omega_1 = 0.1$, $\omega_2 = 0.1$, $\alpha_1^+ = 0.05$, $\alpha_2^+ = 0.05$, $\alpha_1^- = 0.05$, $\alpha_2^- = 0.05$, $\beta_1 = 0.8$, $\beta_2 = 0.8$, $p_{11} = 0.5$, $p_{22} = 0.5$ and $\nu = 5$. Lower boundaries were set to 0.0 (2.0 for ν) and upper boundaries to 1.0 (50 for ν).

The model parameters estimated by `DEoptim` clearly indicate two different regimes for the conditional variance process. More precisely, the values of $\hat{\omega}_i$ and $\hat{\beta}_i$ are far apart between the regimes. We note the presence of leverage effect in both regimes (i.e., $\hat{\alpha}_i^+ < \hat{\alpha}_i^-$), with similar levels. The unconditional variance of the first regime is 0.554, about four times smaller than in the second regime; we will therefore refer to regime one as the low-volatility regime and to regime two as the high-volatility regime. The estimated transition probabilities $\hat{p}_{11}$ and $\hat{p}_{22}$ are respectively 0.995 and 0.997 indicating infrequent mixing between states. Finally, the estimated degrees of freedom parameter suggests heavy tails for the distribution of the conditional log-returns.

In Figure 7 we display the filtered probabilities of the high-volatility state implied by the model parameters estimated with `DEoptim` (in blue), `nlminb` (in green) and `optim` with `method = "SANN"` (in red) together with the log-returns (in small circles). The parameters obtained with `DEoptim` lead to a clear separation of regimes in the filtering probabilities. The beginning of year 1991 is associated with the high-volatility state. Then, from the second half of 1991 to

1997, the returns are clearly associated with the low-volatility regime, with the exception of 1994. From 1997 to 2000, the model remains in the high-volatility regime with a transition during the second semester 2000 to the low-volatility state. On the contrary, the two other methods do not capture any clear regime separation in the state variable's behavior.

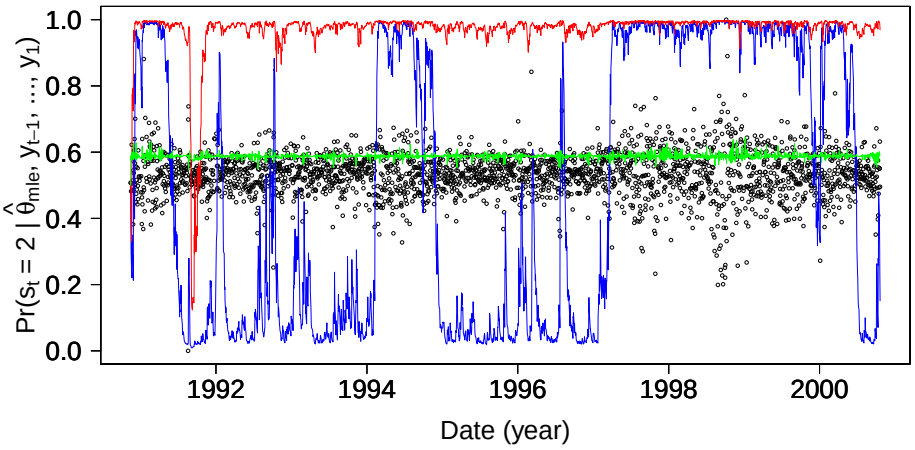

Figure 7: Filtered probabilities of the high-volatility state. Estimations obtained by `DEoptim` are given in blue, `nlminb` in green and `optim` with `method = "SANN"` in red. Small circles depict SMI log-returns.

6. Summary and conclusions

Differential evolution is a heuristic evolutionary method for global optimization that is effective on many problems of interest in science in technology. In implementing the package **DEoptim** we have made this algorithm possible to easily apply in the R language and environment. As Section 3 details, we have also made available many variations on the classical DE strategy. These variations as well as the classical strategy are due to Price, Storn and Lampinen, and we have referred the interested reader to their textbook (Price *et al.* 2006) on DE for details.

We have described herein the use of the package for fitting the Parratt recursion models for X-ray reflectometry and an MSGARCH model for the log-returns of the Swiss Market Index. These case studies showcase the power of the DE algorithm underlying **DEoptim**. We hope that readers will find the package to be a valuable tool for optimization. Finally, if you use R or **DEoptim**, please cite the software in publications.

Computational details

The results in this paper were obtained using R 2.10.0 (R Development Core Team 2009) with the packages `DEoptim` version 2.0-4 (Ardia and Mullen 2009). Computations were performed on a Genuine Intel dual core CPU T2400 1.83Ghz processor and on a quad core Intel Xeon Processor E5410.

DEoptim relies on repeated evaluation of the objective function in order to move the popula-

tion 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.

Acknowledgements

Many **DEoptim** users have sent us comments that helped improve the package. We would like to thank in particular Hans Werner Borchers, Eugene Demidenko, Tarmo Leinonen, Soren Macbeth, Dorothée Pages, Brian Peterson, and Joshua Ulrich. We would also like to thank Juan David Ospina Arango, who inspired us to present the Rastrigin function as an example. Finally, we would like to thank Rainer Storn for his advocacy of DE and making his code publicly available, which was a great help to us in the implementation of **DEoptim**.

References

Als-Nielsen J, McMorrow D (2001). *Elements of Modern X-ray Physics*. Wiley.

Ardia D (2008). *Financial Risk Management with Bayesian Estimation of GARCH Models: Theory and Applications*, volume 612 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, Germany. ISBN 978-3-540-78656-6. doi:10.1007/978-3-540-78657-3. URL http://www.springer.com/economics/econometrics/book/978-3-540-78656-6.

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

Björck M, Andersson G (2007). "*GenX*: An Extensible X-ray Reflectivity Refinement Program Utilizing Differential Evolution." *Journal of Applied Crystallography*, **40**(6), 1174–1178.

Börner J, Higgins SI, Kantelhardt J, Scheiter S (2007). "Rainfall or Price Variability: What Determines Rangeland Management Decisions? A Simulation-Optimization Approach to South African Savanas." *Agricultural Economics*, **37**(2–3), 189–200. doi:10.1111/j.1574-0862.2007.00265.x.

Bowen DK, Tanner BK (2006). *X-Ray Metrology in Semiconductor Manufacturing*. CRC.

Cao R, Vilar JM, Devia A (2009). "Modelling Consumer Credit Risk via Survival Analysis." *Statistics & Operations Research Transactions*, **33**(1), 3–30.

Dueker MJ (1997). "Markov Switching in GARCH Processes and Mean-Reverting Stock-Market Volatility." *Journal of Business & Economic Statistics*, **15**(1), 26–34.

Hamilton JD (1989). "A New Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle." *Econometrica*, **57**(2), 357–384.

Higgins SI, Kantelhardt J, Scheiter S, Boerner J (2007). "Sustainable Management of Extensively Managed Savanna Rangelands." *Ecological Economics*, **62**(1), 102–114. doi:10.1016/j.ecolecon.2006.05.019.

Holland JH (1975). *Adaptation in Natural Artificial Systems*. University of Michigan Press, Ann Arbor.

Klaassen F (2002). "Improving GARCH Volatility Forecasts with Regime-Switching GARCH." *Empirical Economics*, **27**(2), 363–394. doi:10.1007/s001810100100.

Lamoureux CG, Lastrapes WD (1990). "Persistence in Variance, Structural Change, and the GARCH Model." *Journal of Business & Economic Statistics*, **8**(2), 225–243.

Mitchell M (1998). *An Introduction to Genetic Algorithms*. The MIT Press.

Nelson A (2006). "Co-refinement of multiple-contrast neutron/X-ray reflectivity data using *MOTOFIT*." *Journal of Applied Crystallography*, **39**(2), 273–276. doi:10.1107/S0021889806005073. URL http://dx.doi.org/10.1107/S0021889806005073.

Opsina Arango JD (2009). *Estimacion de un Modelo de Difusion con Saltos con Distribucion de Error Generalizada Asimetrica usando Algorithmos Evolutivos*. Master's thesis, Universidad Nacional de Colombia.

Parratt LG (1954). "Surface Studies of Solids by Total Reflection of X-Rays." *Physical Review*, **95**(2), 359–369.

Price KV, Storn RM, Lampinen JA (2006). *Differential Evolution: A Practical Approach to Global Optimization*. Springer-Verlag, Berlin, Germany. ISBN 3540209506.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Storn R, Price K (1997). "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces." *Journal of Global Optimization*, **11**(4), 341–359. ISSN 0925-5001.

Taylor M, Wall J, Loxley N, Wormington M, Lafford T (2001). "High resolution X-ray Diffraction Using a High Brilliance Source, with Rapid Data Analysis by Auto-fitting." *Materials Science and Engineering B*, **80**(1-3), 95 – 98. ISSN 0921-5107.

Wormington M, Panaccione C, Matney KM, Bowen DK (1999). "Characterization of Structures from X-ray Scattering Data Using Genetic Algorithms." *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, **357**(1761), 2827–2848. doi:10.1098/rsta.1999.0469.

Affiliation:

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

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