DEoptim: An R Package for Global Optimization by Differential Evolution

David Ardia aeris CAPITAL AG
Katharine M. Mullen NIST
Joshua Ulrich FOSS Trading
Brian G. Peterson DV Trading

Abstract

This introduction to the R package **DEoptim** is an abreviated version of the manuscript Mullen *et al.* (2011), published in the *Journal of Statistical Software.* **DEoptim** implements the Differential Evolution algorithm for 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. Moreover, the package is self-contained and does not depend on any other packages.

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), and are often referred to as *evolutionary algorithms*. The genetic algorithm is one such method, and was invented by John Holland in the 1960s (Holland 1975). Genetic algorithms apply logical operations, usually on bit strings of fixed or variable length, in order to perform crossover, mutation, and selection on a population. Over the course of successive generations, the members of the population are more likely to represent a minimum of an objective function. Genetic algorithms have proven themselves to be useful heuristic methods for global optimization, in particular for combinatorial optimization problems. Evolution strategies are another variety of evolutionary algorithm, in which members of the population are represented with floating point numbers, and the population is transformed over successive generations using arithmetic operations. See Price *et al.* (2006, Section 1.2.3) for a detailed overview of evolutionary algorithms.

In the 1990s Rainer Storn and Kenneth Price developed an evolution strategy they termed *Differential Evolution* (DE) (Storn and Price 1997). 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 *et al.* (2006) catalogue.

The **DEoptim** (Ardia *et al.* 2011c) 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 2011). **DEoptim** has been published on the Comprehensive R Archive Network and is available at `http://cran.r-project.org/web/packages/DEoptim/`. The **DEoptim** project is hosted on R-forge at `https://r-forge.`

r-project.org/projects/deoptim/. Since becoming publicly available it has been used by a variety of authors, e.g., Börner *et al.* (2007), Higgins *et al.* (2007), Opsina Arango (2009), Ardia *et al.* (2011b) and Ardia *et al.* (2011a) to solve optimization problems arising in diverse domains. Interested readers are referred to Mullen *et al.* (2011) for a longer version of the present vignette.

The recently released **RcppDE** (Eddelbuettel 2010) package ports the C code in **DEoptim** to C++. The documentation claims **RcppDE** is more efficient than the C-based implementation in **DEoptim** version 2.0-7, but most of the increased efficiency is attributable to a non-trivial difference in the functionality: **RcppDE** does not pass `"..."` to the objective function. **DEoptim** version 2.0-8 incorporates several language-independent improvements made in **RcppDE** and compares favorably to a patched version of **RcppDE** that passes `"..."`.

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.

In the remainder of this vignette 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 serves as a user manual.

1.1. An introductory example

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

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

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")
```

The `DEoptim` function of the package **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,

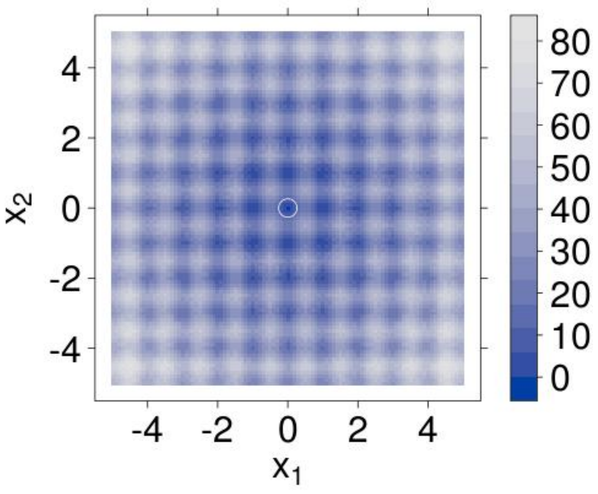

Figure 1: A contour plot of the two-dimensional Rastrigin function $f(x)$. The global minimum $f(x) = 0$ is at $(0, 0)$ and is marked with an open white circle.

`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 (`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 (50 being the default value of `NP`), 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 Rastrigin problem more efficiently than the simulated annealing method found in the R function `optim`.

We note that as the dimensionality of the Rastrigin problem increases, **DEoptim** may not be able to find the global minimum in the default number of generations. Heuristics to help ensure that the global minimum is found include re-running the problem with a larger population size (value of `NP`), and increasing the maximum allowed number of generations.

1.2. Problems suitable for DE

Differential Evolution does not require derivatives of the objective function. It is therefore useful in situations in which the objective function is stochastic, noisy, or difficult to differentiate. DE, however, may be inefficient on smooth functions, where derivative-based methods generally are most efficient.

In the example below, a generalized Rosenbrock function is considered. This function is

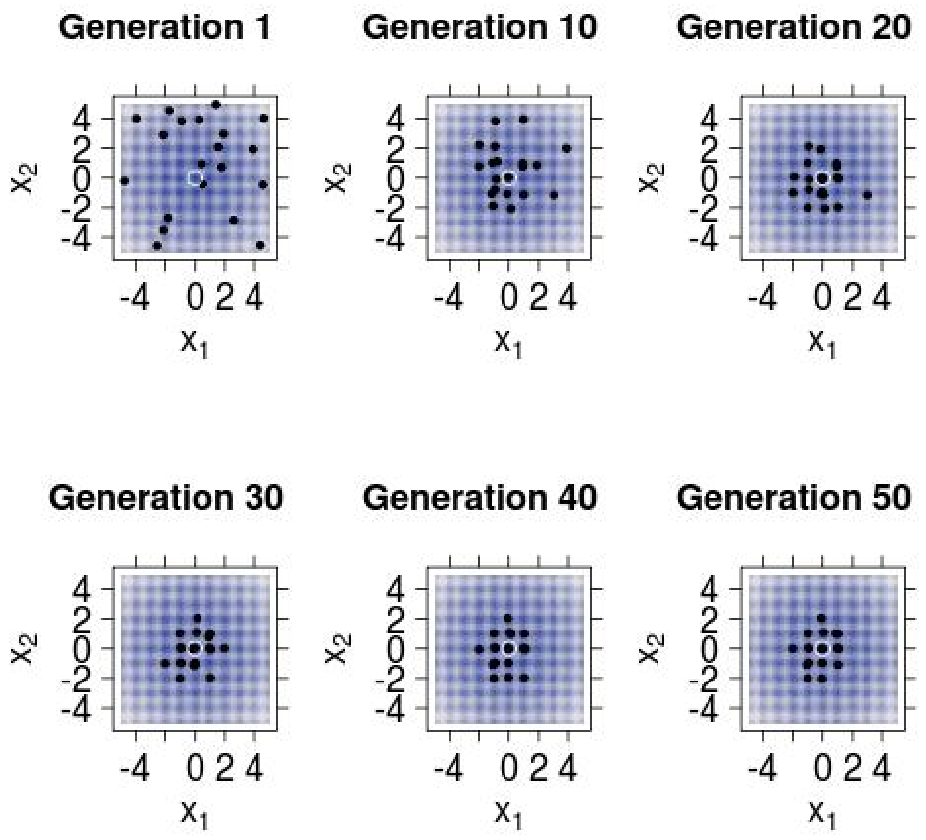

Figure 2: The population associated with various generations of a call to `DEoptim` as it searches for the minimum of the Rastrigin function (marked with an open white circle). The minimum is consistently determined within 200 generations using the default settings of `DEoptim`.

differentiable and has a single local minimum. It is often more efficient to apply methods other than DE to optimization of such functions. Functions that are smooth but have many local minima, however, may still be good candidates for optimization with DE, since alternative algorithms for local, as opposed to global, optimization may converge to a sub-optimal solution.

A generalized Rosenbrock function is possible to represent in R as

```
R> genrose.f <- function(x){
+ n <- length(x)
+ fval <- 1.0 + sum (100 * (x[1:(n-1)]^2 - x[2:n])^2 + (x[2:n] - 1)^2)
+ return(fval)
+ }
```

This function has a global minimum at 1, which **DEoptim** finds for $n = 10$ with a call like:

```
R> n <- 10
```

```
R> ans <- DEoptim(fn=genrose.f, lower=rep(-5, n), upper=rep(5, n),
+                 control=list(NP=100, itermax=4000,trace=FALSE))
```

The minimum can be determined with far fewer function evalutations with a gradient-based method such as "BFGS" (Nash 1990), e.g., with the call

```
R> ans1 <- optim(par=runif(10,-5,5), fn=genrose.f, method="BFGS",
+                control=list(maxit=4000))
R>
```

Note further that users interested in exact reproduction of results should set the seed of their random number generator before calling **DEoptim**. DE is a randomized algorithm, and the results may vary between runs.

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 further elaboration. 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. In order generate a new parameter vector, DE disturbs an old parameter vector with the scaled difference of two randomly selected parameter vectors.

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 $x_{i,g}$, where i indexes the vectors that make up the population and g indexes generation. This is accomplished using *differential mutation* of the population members. A trial 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

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

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 $\mathcal{U}(0, 1)$. 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. Mutation is applied in this way to each member of the population.

If an element v_j of the parameter vector is found to violate the bounds after mutation and crossover, it is reset, where here and throughout we use j to index into a parameter vector. In the implementation of **DEoptim**, if $v_j > \text{upper}_j$, it is reset as $v_j \doteq \text{upper}_j - rand \cdot (\text{upper}_j - \text{lower}_j)$, and if $v_j < \text{lower}_j$, it is reset as $v_j \doteq \text{lower}_j + rand \cdot (\text{upper}_j - \text{lower}_j)$. This ensures that candidate population members found to violate the bounds are set some random amount away from them, in such a way that the bounds are guaranteed to be satisfied. 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.

The algorithm stops after some set number of generations, or after the objective function value associated with the best member has been reduced below some set threshold, or if it is unable to reduce the best member by a certain value over over set number of iterations.

Variations on this theme are possible, some of which are described in the following section. Values of *NP* and *CR* that have been found to be most effective for a variety of problems are described in Price *et al.* (2006, Section 2). 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 by David Ardia. Early versions were written in pure R. Since version 2.0-0 (published to CRAN in 2009 by Katharine Mullen) the package has relied on an 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.
- `...`: allows the user to pass additional arguments to the function `fn`.

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

$$v_{i,g} \doteq \mathrm{old}_{i,g} + \mathrm{F} \cdot (\mathrm{best}_g - \mathrm{old}_{i,g}) + \mathrm{F} \cdot (x_{r1,g} - x_{r2,g})$$

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

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

is used, where jitter is defined as $0.0001 \cdot \mathit{rand} + F$.

- `4`: DE / rand / 1 / bin with per vector dither. In place of the classical DE mutation given in (1), the expression

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

is used, where dither is calculated as dither $\doteq F + \mathit{rand} \cdot (1 - F)$.

- `5`: DE / rand / 1 / bin with per generation dither. The strategy described for `4` is used, but dither is only determined once per-generation.
- `6`: DE / current-to-p-best / 1. The top $(100 * p)$ percent best solutions are used in the mutation, where p is defined in $(0, 1]$.
- any value not above: variation to DE / rand / 1 / bin: either-or algorithm. In the case that $\mathit{rand} < 0.5$, the classical strategy described for `1` is used. Otherwise, the expression

$$v_{i,g} \doteq x_{r0,g} + 0.5 \cdot (F + 1.0) \cdot (x_{r1,g} + x_{r2,g} - 2 \cdot x_{r0,g})$$

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 `0.5`.

- `F`: Stepsize from interval [0,2]. The default value is `0.8`.

- `trace`: Positive integer or logical value indicating whether printing of progress occurs at each iteration. The default value is `TRUE`. If a positive integer is specified, printing occurs every `trace` iterations.

- `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 convergence. Defaults to `NULL`, so that the initial population is generated randomly within the lower and upper boundaries.

- `storepopfrom`: From which generation should the following intermediate populations be stored in memory. Default to `itermax + 1`, i.e., no intermediate population is stored.

- `storepopfreq`: The frequency with which populations are stored. 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 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.

A note on recommended settings: We have set the default values to the methods recommended by Price *et al.* (2006) as starting points. We use `strategy = 2` by default; the user should consider trying as alternatives `strategy = 6` and `strategy = 1`, though the best method will be highly problem-dependent. Generally, the user should set the lower and upper bounds to exploit the full allowable numerical range, i.e., if a parameter is allowed to exhibit values in the range [-1, 1] it is typically a good idea to pick the initial values from this range instead of unnecessarily restricting diversity. Increasing the value for `NP` will mean greater likelihood of finding the minimum, but run-time will be longer.

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.

Disclaimer

The views expressed in this vignette are the sole responsibility of the authors and do not necessarily reflect those of NIST and aeris CAPITAL AG.

References

Ardia D, Arango JO, Gomez NG (2011a). "Jump-Diffusion Calibration using Differential Evolution." *Wilmott Magazine*, **55**, 76–79.

Ardia D, Boudt K, Carl P, Mullen KM, Peterson BG (2011b). "Differential Evolution with DEoptim: An Application to Non-Convex Portfolio Optimization." *The R Journal*, **3**(1), 27–34.

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

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.

Eddelbuettel D (2010). ***RcppDE**: Global optimization by differential evolution in C++*. R package version 0.1.0, URL http://CRAN.R-project.org/package=RcppDE.

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

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

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

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/.

Nash JC (1990). *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. Adam Hilger.

Opsina Arango J (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.

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

R Development Core Team (2011). *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.

Affiliation:

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

Joshua Ulrich
FOSS Trading
URL: http://fosstrading.com

Brian G. Peterson
DV Trading
E-mail: brian@braverock.com