Journal of Statistical Software

September 2014, Volume 60, Issue 6. http://www.jstatsoft.org/

Continuous Global Optimization in R

Katharine M. Mullen
University of California, Los Angeles

Abstract

This article surveys currently available implementations in R for continuous global optimization problems. A new R package **globalOptTests** is presented that provides a set of standard test problems for continuous global optimization based on C functions by Ali, Khompatraporn, and Zabinsky (2005). 48 of the objective functions contained in the package are used in empirical comparison of 18 R implementations in terms of the quality of the solutions found and speed.

Keywords: global optimization, constrained optimization, continuous optimization, R.

1. Introduction to global optimization

Global optimization is the process of finding the minimum of a function of n parameters, with the allowed parameter values possibly subject to constraints. In the absence of constraints (which are discussed in Section 1.1), the task may be formulated as

$$\underset{x}{\text{minimize}} \quad f(x) \tag{1}$$

where f is an objective function and the vector x represents the n parameters. If f is a function $\Re^n \to \Re$, so that elements x_i of the input vector x and the output value are real numbers, the global optimization problem is continuous.

Global optimization may be contrasted with local optimization. Local optimization finds local optima, which represent the best solution in a subset of the parameter space, not necessarily in the parameter space as a whole. A local optimum x^* may be defined as a point for which there exists some $\delta > 0$ such that for all points x such that $\|x - x^*\| \leq \delta$; $f(x^*) \leq f(x)$; in other words, a local optima x^* is a point at which the objective function $f(x^*)$ is less than or equal to $f(x)$ at all other points x in a certain neighborhood.

Convex optimization problems have as their solutions the optima of convex functions. Convex functions are continuous functions whose value at the midpoint of every interval in the domain

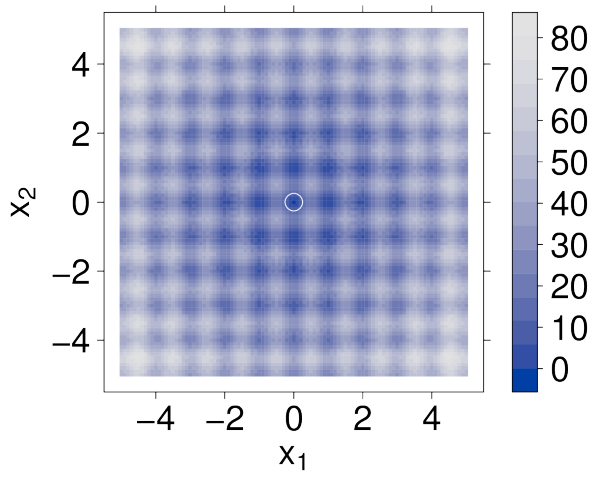


Figure 1: A contour plot of the two-dimensional Rastrigin function $f(x)$. The global minimum $f(x) = 0$ is at $(0, 0)$ and is marked with an open white circle. Local optima (shown as darker spots) are found throughout the parameter space at regularly spaced intervals.

does not exceed the mean of the values at the ends of the interval, i.e., f is convex in $[a, b]$ if for any two points x_1 and x_2 in $[a, b]$ and any λ where $0 < \lambda < 1$, $f[\lambda x_1 + (1 - \lambda)x_2] \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$ (Rudin 1976). Any local optimum of a convex function is also a global optimum. Statisticians are familiar with the convex optimization problem of minimizing the sum of the squared differences between a linear or nonlinear function and a vector of data, i.e., regression. Regression problems are such that you can start at any point in the parameter space, determine the gradient, take an appropriately-sized step in the direction that minimizes this gradient, and repeat until the gradient is vanishing, in this way determining a local (and global) solution (provided that the problem is well-conditioned). Such simple and computationally efficient methods fail on non-convex problems, in which finding a local optimum is no guarantee of finding a global solution.

Finding solutions to continuous global optimization problems is in some instances akin to finding a needle in a haystack. For instance, define the objective function $f : \Re^n \to \Re$ as zero everywhere except at a unique vector $x \in \Re^n$, where $f(x) = -1$. The problem has no structure that provides clues to when a guess x' is close to the solution x, and so no algorithm can efficiently and consistently discover the optimum.

However, most global optimization problems encountered in practice have features that can be exploited to render algorithms for their solution more efficient. For example, in many applications the objective function f is continuous, meaning that small changes in x translate to small changes in $f(x)$. Another common property is differentiability, i.e., at every point $x \in \Re^n$ the partial derivatives of $f(x)$ exist. Functions f that are differentiable allow the application of gradient-based algorithms, that is, methods that rely on forming the Jacobian matrix which represents the partial derivatives of a candidate solution x with respect to f. Note however, that continuity and differentiability make it easier to find local optima, but there may be many such optima, and finding the global optimum may remain difficult.

As an example of a global optimization problem, consider the minimization of the Rastrigin

function in $x \in \Re^D$

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

for $D = 2$, which is a common test for global optimization algorithms (Mullen, Ardia, Gil, Windover, and Cline 2011). As shown in Figure 1, the function has a global minimum where $f(x) = 0$ at the point $(0, 0)$. The function also has many local minima, and solutions returned by local optimization algorithms will not in general be globally optimal. However, the problem is continuous and differentiable, and the global optimum is reliably returned by many of the R implementations of global optimizers to be considered in this paper.

1.1. Constrained global optimization

The simplest and most common type of constraint on a global optimization is the box constraint, which sets a lower and upper bound on each element of the parameter vector. More generally, constrained global optimization problems seek a solution x that minimizes an objective function f such that x satisfies a set of inequality or equality constraints, often stated in the standard form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, i = 1, \ldots, p \\ & g_j(x) \leq 0, j = 1, \ldots, q \end{aligned} \tag{2}$$

where h and g are functions that may be nonlinear. Another important variety of constrained optimization are quadratic programming problems, which may be expressed in canonical form as

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) = \frac{1}{2} x^\top Q x + cx \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{3}$$

where Q is a symmetric $n \times n$ matrix, the vectors x and c have length n, and the constraints are defined by an $m \times n$ matrix A and a vector b of length m containing right-hand-side coefficients. If Q is zero, the quadratic programming problem is a linear programming problem, which can be solved by efficient polynomial time methods. Q is positive definite and the problem has a feasible solution, then the problem is convex, and there exists a unique global optimum, which also can be found with a polynomial time algorithm. However, if Q is not positive definite, all known algorithms for quadratic programming problems require in the worst case exponential time to solve; such cases must be addressed via global optimization methods. See, e.g., Nocedal and Wright (2006) and Jensen and Bard (2002) for elaboration.

None of the global optimization methods empirically compared in this paper allow explicit input of constraints other than box constraints. Constraints may however be implicitly set by use of penalty functions, so that the objective function returns a large or infinite value whenever the parameters are outside of the feasible region.

1.2. Paper outline

This paper surveys currently available methods for general-purpose continuous global optimization problems in the R language (R Core Team 2014) in Section 2. Section 3 presents the

R package **globalOptTests**, available from the Comprehensive R Archive Network (CRAN) at http://CRAN.R-project.org/package=globalOptTests, which collects 50 objective functions useful for benchmarking implementations of continuous global optimization algorithms (Ali *et al.* 2005). Studies to benchmark 18 R implementations in terms of the quality of solution found and run-time given a set budget of function evaluations using these objective functions are described in Section 4. Results are presented in Section 5. Section 6 contains conclusions and discussion.

For further introduction to continuous global optimization, see e.g., Horst, Pardalos, and Thoai (2000), Pardalos and Romeijn (2002), Neumaier (2013), Neumaier (2004), Floudas and Gounaris (2009), and Weise (2009). For further pointers to optimization methods in R, see the *Optimization and Mathematical Programming* task view on CRAN (Theussl 2014).

2. Implementations in R

The R language and environment provides an ever-increasing assortment of implementations of algorithms for the solution of continuous global optimization problems. The implementations may be roughly grouped into the following categories, though some implementations have the properties of more than one category.

- Annealing methods:
  - **stats** (R Core Team 2014): `optim(method = "SANN")`.
  - **GenSA** (Xiang, Gubian, Suomela, and Hoeng 2013).
- Evolutionary methods:
  - **rgenoud** (Mebane and Sekhon 2013).
  - **DEoptim** (Ardia, Mullen, Peterson, and Ulrich 2013), **RcppDE** (Eddelbuettel 2013).
  - **NMOF** (Schumann 2013): `DEopt` and `GAopt`.
  - **soma** (Clayden and based on the work of Ivan Zelinka 2011).
  - **Rmalschains** (Bergmeir, Molina, and Benítez 2014).
  - **cmaes** (Trautmann, Mersmann, and Arnu 2011).
  - **parma** (Ghalanos 2014): `cmaes`.
  - **BurStMisc** (Burns Statistics 2012): `genopt` function.
  - **GA** (Scrucca 2014).
  - **mcga** (Satman 2014).
  - **hydromad** (Andrews and Guillaume 2013): `SCEoptim`.
- Particle swarm optimization methods:
  - **ppso** (Francke 2012).
  - **pso** (Bendtsen 2012).
  - **hydroPSO** (Zambrano-Bigiarini 2013).
  - **NMOF**: `PSopt`.

- Branch and bound methods:
  - **nloptr** (Ypma 2014): StoGo.
- Deterministic methods:
  - **nloptr**: DIRECT.
- Other stochastic methods:
  - **nloptr**: CRS,
  - **NMOF**: `TAopt` and `LSopt`,
  - **nloptr**: MLSL.

All of these methods are heuristic, meaning that there is some likelihood but no guarantee that they find the optimal solution. Nonheuristic methods, in contrast, may maintain a provable upper and lower bound on the globally optimal objective value, and systematically explore the parameter space until a satisfactory value is found.

Note that we have excluded methods accessible by the package **rneos** (Pfaff 2012), which enables the user to pass optimization problems to the **NEOS** server (University of Wisconsin-Madison 2013; Czyzyk, Mesnier, and Moré 1998) and retrieve results within R; the linked-to packages are often commercial, closed-source software.

Implementations that are only useful for local optimization are also not considered here; the methods considered all are able to deal with test problems in which there are many (sub-optimal) local optima.

2.1. Annealing methods

Simulated annealing is a stochastic, heuristic method that starts at a randomized point x in the parameter space, and then evaluates a neighboring point x', usually chosen at random (Kirkpatrick, Gelatt, and Vecchi 1983). In its simplest form, if the value of the objective function is lesser at the new point, the new point is accepted, and the process is repeated. If the value at the new point is greater, however, the point is chosen with some acceptance probability P less than unity; this allows the currently best point considered by the algorithm to zero in on optima in f, but to escape local optima, since $P < 1$. P is reduced as the number of evaluations of f increases according to an annealing schedule, so that the probability of accepting a worse point x' decreases as the algorithm zeros in on a good solution.

stats *package,* `optim` *function,* `method = "SANN"`

A simulated annealing algorithm is included in the base R distribution in the function `optim` from the **stats** package; it is used by setting the `optim` argument `method` with `method = "SANN"`.

GenSA *package*

GenSA implements a generalized simulated annealing algorithm described in Xiang *et al.* (2013).

2.2. Evolutionary methods

Evolutionary methods are stochastic, heuristic optimization methods that have an analogue with the process of natural selection: the most fit members of a population survive into the next generation. Early examples of evolutionary methods include genetic algorithms (Holland 1975), which start with a population composed of candidate solutions (usually bit strings), and then apply logical operations of crossover and mutation to generate a new population. The objective function is evaluated for each string of the new population, and only the strings associated with the best objective function values survive to repeat the process.

rgenoud package

rgenoud (Mebane and Sekhon 2013) implements an algorithm that combines evolutionary search algorithms with derivative-based (Newton or quasi-Newton) methods. It may also be used for problems for which derivatives do not exist. Sekhon and Mebane (1998) describe the algorithm; the implementation is detailed in Mebane and Sekhon (2011). The package supports the use of multiple CPUs to perform parallel computations.

DEoptim package

DEoptim (Ardia *et al.* 2013) implements differential evolution (Price, Storn, and Lampinen 2006), a strategy similar to a genetic algorithm but designed for continuous optimization (i.e., optimization of real vectors). The package sees much use in quantitative finance applications (Ardia, Arango, and Gomez 2011a; Ardia, Boudt, Carl, Mullen, and Peterson 2011b). Mullen *et al.* (2011) describe the package, though in later versions parallelization options, options for mixed-integer problems, and the adaptive mutation strategy JADE (Zhang and Sanderson 2009) have been added. An implementation of **DEoptim** that uses C++ instead of C for computations has been implemented in the package **RcppDE** (Eddelbuettel 2013).

NMOF package: `DEopt` and `GAopt`

NMOF (Schumann 2013), a package associated with the book by Gilli, Maringer, and Schumann (2011), contains several implementations of global optimization algorithms, including the `DEopt` function for differential evolution and the `GAopt` function for optimization with genetic algorithms.

soma package

soma (Clayden and based on the work of Ivan Zelinka 2011) provides an R implementation of the self-organizing migrating algorithm (Zelinka 2004), which moves a set of individuals towards the best candidate solution x over the course of successive generations.

Rmalschains package

Rmalschains (Bergmeir *et al.* 2014) implements an algorithm for continuous optimization using local search chains (MA-LS-Chains) in R. **Rmalschains** attempts to obtain better performance (in terms of the precision of the solution and the number of evaluations required to reach it) by strategically applying local search (Molina, Lozano, García-Martínez, and Herrera 2010). The algorithm is an example of a memetic algorithm, which combines an evolutionary strategy

with a local optimization algorithm (often performed from the starting points represented by individual population members).

cmaes *package*

A covariance matrix adapting evolutionary strategy (CMA-ES, Hansen and Ostermeier 1996) is implemented in the package **cmaes** (Trautmann *et al.* 2011). Instead of picking new search points from a spherical normal distribution in the n-dimensional search space, an approximation for the covariance matrix of parameters informs the sampling used in generating successive generations.

parma *package*

An implementation of CMA-ES is also found in the package **parma** (Ghalanos 2014).

`genopt` *function from* ***BurStMisc***

The `genopt` function was originally published as a part of the book by Burns (1998); it is available via the **BurStMisc** package (Burns Statistics 2012). It is a simple function which works on real-valued parameter vectors and is one of the older global optimization methods developed for R.

GA *package*

GA (Scrucca 2014) allows optimization using genetic algorithms for both the real and integer parameter spaces.

mcga *package*

mcga (Satman 2014) is a package for optimization of real-valued functions via genetic algorithms.

nloptr *package: ISRES*

nloptr (see Section 2.4) provides an implementation of an evolutionary algorithm termed an improved stochastic ranking evolution strategy (ISRES, Runarsson and Yao 2005).

hydromad *package:* `SCEoptim`

The **hydromad** (Andrews and Guillaume 2013) contains an implementation of shuffled complex evolution in the function `SCEoptim`.

2.3. Particle swarm optimization methods

Particle swarm optimization is a stochastic, heuristic method introduced by Kennedy and Eberhart (1995). A set (swarm) of candidate solutions (particles) is moved through search space using formulas for position and velocity that depend on the state of the rest of the swarm.

pso package

pso (Bendtsen 2012) implements a particle swarm optimization algorithm.

ppso package

ppso (Francke 2012) implements particle swarm optimization along with dynamically dimensioned search algorithms (Tolson and Shoemaker 2007). Options are available for parallelization.

hydroPSO package

hydroPSO (Zambrano-Bigiarini 2013) implements a particle swarm optimization algorithm; its development was motivated by the need to fit environmental models, though it is a general-purpose optimizer.

NMOF package: `PSopt`

The **NMOF** package also contains a particle swarm optimization implementation.

2.4. Branch and bound methods

Branch and bound (Scholz 2012) is a systematic, nonheuristic method for solving optimization problems. It is the most widely used type of algorithm for solving difficult combinatorial optimization problems, though it is also often applied to continuous optimization. It often leads to exponential time complexities in the worst case. Most algorithms in this category apply a breadth-first search for the optimal solution, but not all nodes get expanded; a selection criterion determines which node to expand and when, and another criterion tells the algorithm when an optimal solution has been found. The search terminates when there are no unexplored parts of the parameter space remaining.

nloptr package: StoGo

nloptr (Ypma 2014) is an R interface to **NLopt** (Johnson 2013), a collection of open-source/freely available implementations for optimization. All methods in **nloptr** are possible to use via the package **nloptwrap** package (Borchers 2014), which was created to make the solvers easier to use.

The StoGo algorithm (Madsen, Zertchaninov, and Zilinskas 1998; Zertchaninov and Madsen 1998) accessible with **NLopt** divides the search space into smaller hyper-rectangles via a branch-and-bound technique, and searches these subspaces using a gradient-based local-search algorithm, optionally including some randomness. The method requires a gradient. When called via the `stogo` function from **nloptwrap**, a numeric gradient is used.

2.5. Deterministic methods

nloptr package: DIRECT

The DIRECT algorithm (Jones, Perttunen, and Stuckman 1993; Gablonsky and Kelley 2001) in **NLopt** for global optimization is a deterministic method based on division of the search

domain into smaller and smaller hyperrectangles. The **nloptr** package makes several different varieties of the algorithm available; the DIRECT_L method employs some randomization.

2.6. Other stochastic methods

This section collects stochastic algorithms that do not fall into the preceding categories.

nloptr *package: CRS*

nloptr (see Section 2.4) makes available an implementation of a controlled random search (CRS) algorithm (Kaelo and Ali 2006).

NMOF *package:* `TAopt` *and* `LSopt`

NMOF (see Section 2.2) contains an implementation of a threshold acceptance algorithm (Dueck and Scheuer 1990). It is similar in some ways to simulated annealing but requires the definition of a neighborhood function that generates new guesses for the optimal parameter vector from a currently accepted parameter vector. The function `LSopt` is similar.

nloptr *package: MLSL*

nloptr (see Section 2.4) makes available an implementation of a multi-level single-linkage (MLSL) algorithm (Kan and Timmer 1987), which performs a sequence of local searches from randomly chosen points.

3. Test problems

Given the many options currently available in R for the solution of continuous global optimization problems, which methods perform best? This question is difficult to answer in general, since the performance of any implementation is problem-dependent. An implementation may do well on a certain type of problem (e.g., problems with a parameter space of fewer than four dimensions, or problems in which there are only a handful of local optima) and fail miserably in others. Some insight into performance can be obtained by testing implementations on a wide variety of objective functions. Previous comparisons of the performance of implementations in R for solving continuous global optimization problems have been limited to a comparison of a very small number of implementations, as in Xiang *et al.* (2013), or on a very limited number of problems, as in Burns (2012a) and Burns (2012b).

A new package **globalOptTests** was created in order to collect an assortment of objective functions useful for testing implementations for continuous global optimization in R. The package makes accessible 50 objective functions first presented by Ali *et al.* (2005) as standard test problems for continuous global optimization written in C. The collection of C functions also appear on (and were downloaded from) a web page maintained by GAMS Development Corporation and GAMS Software GmbH (2013). The underlying C functions were changed in the R package to use a factor in Ackley's function that more commonly appears in the literature (0.2 instead of 0.02, thanks to a suggestion from Hans Werner Borchers). Also, Storn's Tchebychev problem in 9 and 17 dimensions was not included, since the global minima of the implementation of these functions does not appear to correspond to the value reported in Ali *et al.* (2005), the only function definition reference.

Name in package	Full name	Dimension	Global minimum
`Ackleys`	Ackley's problem	10	0
`AluffiPentini`	Aluffi-Pentini's problem	2	−0.3523
`BeckerLago`	Becker and Lago problem	2	0
`Bohachevsky1`	Bohachevsky 1 problem	2	0
`Bohachevsky2`	Bohachevsky 2 problem	2	0
`Branin`	Branin problem	2	0.3979
`Camel3`	Camel back three hump problem	2	0
`Camel6`	Camel back six hump problem	2	−1.0316
`CosMix2`	Cosine mixture problem	2	−0.2
`CosMix4`	Cosine mixture problem, $n = 4$	4	−0.4
`DekkersAarts`	Dekkers and Aarts problem	2	−24776.5183
`Easom`	Easom problem	2	−1
`EMichalewicz`	Epistatic Michalewicz problem	5	−4.6877
`Expo`	Exponential problem	10	−1
`GoldPrice`	Goldstein and Price problem	2	3
`Griewank`	Griewank problem	10	0
`Gulf`	Gulf research problem	3	0
`Hartman3`	Hartman 3 problem	3	−3.8628
`Hartman6`	Hartman 6 problem	6	−3.3224
`Hosaki`	Hosaki problem	2	−2.3458
`Kowalik`	Kowalik problem	4	0.0003
`LM1`	Levy and Montalvo 1 problem	3	0
`LM2n5`	Levy and Montalvo 2 problem	5	0
`LM2n10`	Levy and Montalvo 2 problem, $n = 10$	10	0
`McCormic`	McCormick problem	2	−1.9133
`MeyerRoth`	Meyer and Roth problem	3	$4.355 \times 10^{-}5$
`MieleCantrell`	Miele and Cantrell problem	4	0
`Modlangerman`	Modified Langerman problem	10	−0.965
`ModRosenbrock`	Modified Rosenbrock problem	2	0
`MultiGauss`	Multi-Gaussian problem	2	−1.297
`Neumaier2`	Neumaier 2 problem	4	0
`Neumaier3`	Neumaier 3 problem	10	−210
`Paviani`	Paviani's problem	10	−45.7784
`Periodic`	Periodic problem	2	0.9
`Powell`	Powell's quadratic problem	4	0
`PriceTransistor`	Price's transistor modelling problem	9	0
`Rastrigin`	Rastrigin problem	10	0
`Rosenbrock`	Rosenbrock problem	10	0
`Salomon`	Salomon problem	5	0
`Schaffer1`	Schaffer 1 problem	2	0
`Schaffer2`	Schaffer 2 problem	2	0.0012
`Schubert`	Shubert problem	2	−186.7309
`Schwefel`	Schwefel problem	10	−4189.8289
`Shekel5`	Shekel 5 problem	4	−10.1532
`Shekel7`	Shekel 7 problem	4	−10.4029
`Shekel10`	Shekel 10 problem	4	−10.5364
`Shekelfox5`	Shekel's foxholes problem	5	−10.4056
`Shekelfox10`	Shekel's foxholes problem, $n = 10$	10	−10.2087
`Wood`	Wood's problem	4	0
`Zeldasine10`	Sinusoidal problem	10	−3.5
`Zeldasine20`	Sinusoidal problem, $n = 20$	20	−3.5

Table 1: Functions included in **globalOptTests** and first collected in Ali *et al.* (2005).

The 50 objective functions in **globalOptTests** have parameter spaces between between 2 and 20 dimensions. All are relatively fast to evaluate (requiring less than one second using any common modern CPU) and are noise-free.

The objective functions are called via the function `goTest`. For example, to call the `"ModRosenbrock"` function, first the package is loaded with:

```
R> library("globalOptTests")
```

Then `"ModRosenbrock"` is evaluated at a given parameter vector, e.g., `c(0.4, 0.7)`, via the call:

```
R> goTest(par = c(0.4, 0.7), fnName = "ModRosenbrock")
```

To use the objective function in a call to an implementation of a given global optimization algorithm, for example the `optim` function with the `"SANN"` method, a call such as the following is made:

```
R> out <- optim(par = c(0.4, 0.7), fn = goTest, method = "SANN",
+    control = list(maxit = 10000), fnName = "ModRosenbrock")
```

For more complete information regarding the package's functionality, see the help pages of the package with the call:

```
R> help(package = "globalOptTests")
```

4. Empirical comparison

Eighteen R functions for continuous global optimization (given in Table 2) were applied to 48 objective functions from the package **globalOptTests**. The control settings of each implementation were adjusted to allow for a set number of function evaluations. Eleven of the 18 algorithms allowed explicit setting of the maximum number of function evaluations. Some of the 7 remaining implementations (e.g., `DEoptim`) allowed for setting the precise number of allowed evaluations indirectly; others, (e.g., `soma` and `cma_es`) stop according to convergence criteria automatically, and so may not use the total allotted budget of evaluations. The control settings of each implementation were not adjusted beyond tuning to control the number of function evaluations; undoubtedly, some of the implementations can be made to perform better by tuning additional control parameters.

The implementations chosen for comparison work "out-of-the-box", that is, using only a base R installation and R packages. Only functions that allow passing additional arguments to the objective function via a `...` argument or an environment argument were included (meaning that **ppso** was not considered). Packages which require tuning (e.g., to set a population size as in **mcga**) were not included. The implementation of the CMA-ES method found in the **cmaes** package was included, and not the implementation from **parma**; at the time of writing, the latter is often faster but also more likely to converge on values far from the solution, though the implementation in the **cmaes** package is more likely to terminate in an error.

Lower and upper bounds on parameter values were given as the vectors specified in the C code developed in Ali *et al.* (2005). These default bounds are possible to see using the package **globalOptTests** via calls such as

Function/method	Package	Stochastic?	Type
`GenSA`	**GenSA**	Yes	Annealing
`optim/method="SANN"`	**stats**	Yes	Annealing
`ga`	**GA**	Yes	Evolutionary
`genoud`	**genoud**	Yes	Evolutionary
`DEoptim`	**DEoptim**	Yes	Evolutionary
`soma`	**soma**	Yes	Evolutionary
`cma_es`	**cmaes**	Yes	Evolutionary
`malschains`	**Rmalschains**	Yes	Evolutionary
`SCEoptim`	**hydromad**	Yes	Evolutionary
`DEopt`	**NMOF**	Yes	Evolutionary
`nloptr/algorithm="NLOPT_GN_ISRES"`	**nloptr**	Yes	Evolutionary
`nloptr/algorithm="NLOPT_GN_DIRECT_L"`	**nloptr**	No	Other stochastic
`nloptr/algorithm="NLOPT_GN_CRS2_LM"`	**nloptr**	Yes	Other stochastic
`nloptr/algorithm="NLOPT_GD_STOGO_RAND"`	**nloptr**	Yes	Branch and bound
`nloptr/algorithm="NLOPT_GN_DIRECT"`	**nloptr**	No	Deterministic
`PSopt`	**NMOF**	Yes	Particle swarm
`hydroPSO`	**hydroPSO**	Yes	Particle swarm
`psoptim`	**pso**	Yes	Particle swarm

Table 2: Implementations for continuous global optimization compared here.

```
R> getDefaultBounds("ModRosenbrock")
```

Note that these bounds have been set to be asymmetric about the solution (e.g., if the global optimum is zero, the upper and lower bound associated with a given parameter might be set to -5 and 10, but not -5 and 5). If asymmetry in the bounds is not applied, certain algorithms have an advantage (e.g., the DIRECT method from **nloptr**). If required, a starting parameter vector was given by choosing values uniformly at random between these default lower and upper bounds before each call.

Comparison of implementations for continuous global optimization requires making choices regarding how to quantify performance. Criteria that may be interesting to examine include:

- whether the solution is ever found, even given unlimited time or function evaluations;
- number of function evaluations required to find the global optimum;
- time required to find the global optimum;
- time required to return a solution given a budget of evaluations of the objective function;
- quality of the solution found after a set number of function evaluations.

Here, both the nearness of solutions found to the global optimum ('accuracy') and the time required to return results given a set budget of function evaluations were examined.

For the 18 implementations, the solutions returned after a maximum of 10,000 function evaluations were collected. For each of the 48 objective functions examined, 100 calls to perform the optimization were made, using, if required by the implementation, different values for the starting parameter vector for each call. The budget of 10,000 function evaluations typically allowed each implementation tested to return a solution within a few seconds for the

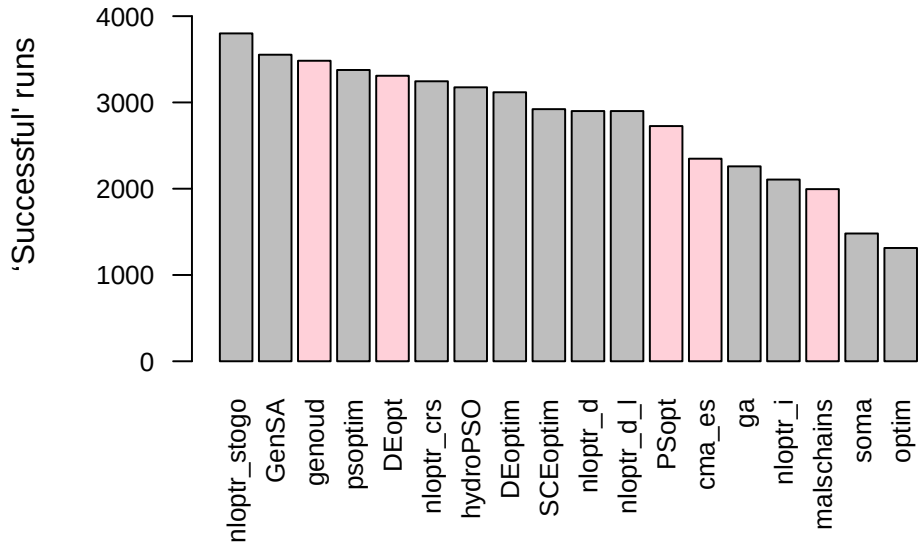


Figure 2: Tally of successes over all 100 runs for each of 48 objective functions (4800 total runs). A 'success' was defined as a solution less than 0.005 more than the minimum of the objective function between the default bounds. Implementations that returned an error as described in Table 3 are marked in red.

fast-to-evaluate objective functions contained in **globalOptTests**. Use of a budget of function evaluations as opposed to system time has the disadvantage of obscuring any inefficiencies in the implementations other than function evaluations, but has the important advantage of being independent of the particular system on which testing is performed. It also renders the study relatively fast to perform (compared, e.g., to giving each implementation an unlimited budget of evaluations with which to attempt to find the solution).

The methods were then compared using the time required to return a solution within a given budget of function evaluations. Timing on the 2-parameter BeckerLago, 4-parameter Kowalik, 10-parameter Rastrigin, and 20-parameter Zeldasine20 functions was measured using the elapsed time returned by the `system.time` function in R version 3.0.0 (running on a dedicated purpose 64-bit Linux distribution with a Intel Pentium Core 2 Duo 2133 MHz CPU). For each of the 18 implementations, 100 calls to perform the optimization were again made. However, in these tests each function was allowed a maximum of approximately 50,000 function evaluations. Options were used to eliminate printing to the screen or file, but otherwise the default settings were again applied.

For the 10-parameter Rastrigin function, the accuracy of the solutions obtained within the budget of 50,000 function evaluations was also examined.

5. Results

In the study that examined the accuracy of solutions found within 10,000 function evaluations, several implementations terminated with errors and did not return results for some runs. The `malschains` function in two cases returned zero as the objective function value along with a parameter vector associated with a non-zero objective function value; this did not result in

Function	Objective function	Runs affected	Summary of error/message
`cma_es`	`EMichalewicz`	2/100	`Inf` returned
`cma_es`	`Hartman3`	8/100	`Inf` returned
`cma_es`	`Hartman6`	25/100	`Inf` returned
`cma_es`	`Zeldasine10`	6/100	`Inf` returned
`cma_es`	`Zeldasine20`	26/100	`Inf` returned
`DEopt`	`Gulf`	100/100	`NA`'s not allowed
`DEopt`	`Paviani`	100/100	`NA`'s not allowed
`genoud`	`DekkersArts`	40/100	`NA`/NaN/`Inf` in foreign call
`genoud`	`Schwefel`	4/100	`NA`/`NaN`/`Inf` in foreign call
`malschains`	`Branin`	1/100	Erroneous fitness of 0
`malschains`	`GoldPrice`	1/100	Erroneous fitness of 0
`PSopt`	`Gulf`	98/100	`NA`'s not allowed
`PSopt`	`Paviani`	100/100	`NA`'s not allowed

Table 3: Problems encountered during selected runs of the accuracy study.

an explicit error but is clearly incorrect. These problems are summarized in Table 3.

Boxplots of the solutions found for the 48 test problems examined are presented in Appendix A. The global minimum of each objective function within the default bounds is shown in these plots as a red horizontal line. Study of these plots reveals that the performance in terms of the quality of the solution is quite heterogeneous among the 48 various objective functions; clear winners on some problems do very poorly on others.

Figure 2 is one way to summarize these results. Each implementation was considered to succeed if it returned a solution less than 0.005 more than the minimum of the objective function between the default bounds. This criterion of success was arbitrary but useful for illuminating the differences in the quality of the solutions found. The number of successes of each implementation was summed over all 100 runs taken on each of the 48 objective functions; the maximum possible number of successes was thus 4800.

Note however that the summary plot given in Figure 2 obscures interesting heterogeneity in performance, which can be studied in the plots in Appendix A. To take just one example, all implementations except `SCEoptim` and `PSopt` failed to consistently find the global minimum for the 2-parameter Easom problem.

For all of the 48 objective functions included in the study, at least one of the 18 implementations tested found the global minimum within the budget of 10,000 function evaluations during at least one run. However, more of the implementations would find the global optimum on many of the 48 test problems given a larger budget of function evaluations. To illustrate this, the solutions found for the 10-dimension Rastrigin problem were examined after increasing the budget of function evaluations to 50,000, from 10,000. Given a budget of 10,000 function evaluations, only the `stogo` method from **nloptr** (called via the package **nloptwrap**) found the global optimum for the Rastrigin problem, as shown in Appendix A. Given a budget of 50,000 function evaluations, however, **GenSA** also consistently finds the global minimum for this problem, as shown in Figure 3.

The good performance of `stogo` in terms of the quality of the solution found given 10,000 function evaluations is a bit misleading, since time tests show it to be the slowest implemen-

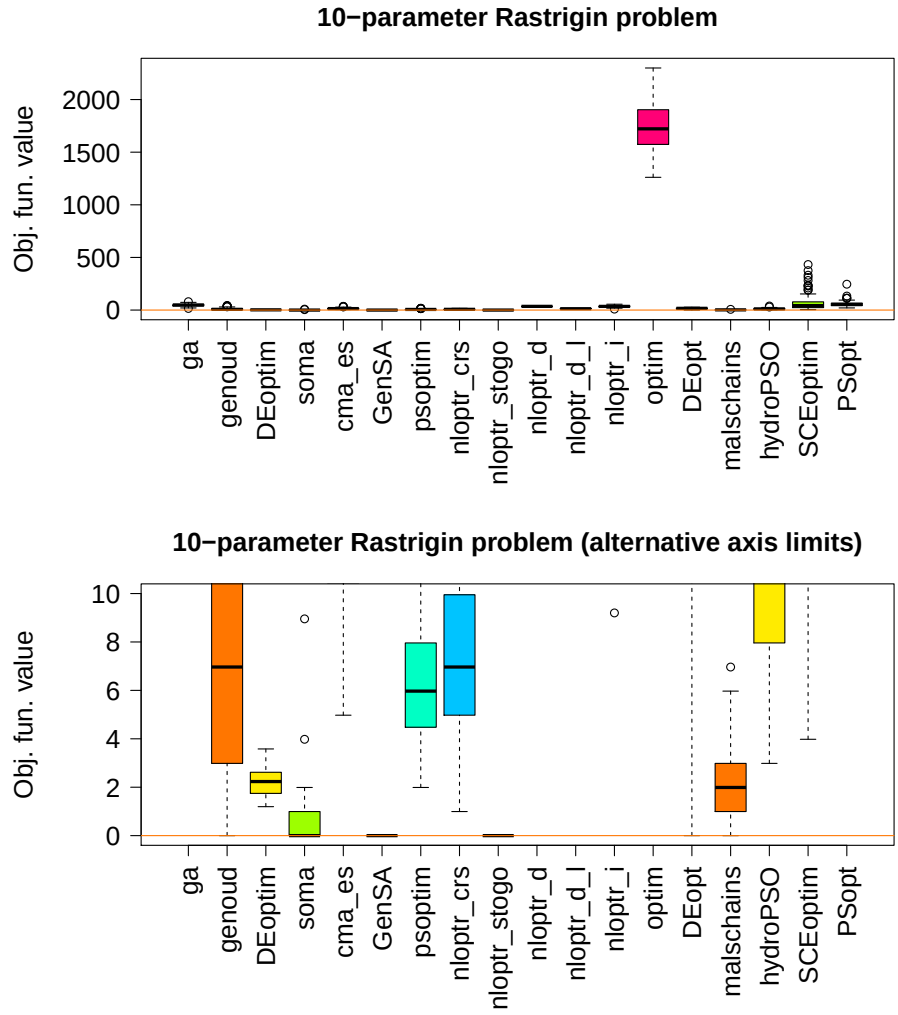


Figure 3: Boxplots of solutions returned for the 10-parameter Rastrigin problem, given a budget of 50,000 function evaluations. The global optimum of the function is marked with a horizontal red line. The axis limits chosen in the left plot include all results, whereas the axis limits in the right plot allow display of only the more accurate solutions.

tation tested by a factor of 10 on many problems. This is at least partially due to the fact that a numerical gradient is calculated (via the `nl.grad` function) when StoGo is called via **nloptwrap** and no analytic gradient is provided. If the user is able to supply an analytic gradient, results will be obtained faster. Else, the quality of the solutions `stogo` returns can often be matched (and in less time) by other implementations by increasing the number of function evaluations allowed.

Timing results for the four objective functions examined for speed are given in Appendix B. As is obvious from the comparison of these plots to the plots of the solutions returned by the various implementations in Appendix A, the implementations which return the most accurate estimates for the global minima of the objective functions tested are not the fastest. The **stogo** method is by a large factor the slowest implementation. The variability of the time required for most of the methods to return results is low (less than a second) for all methods except

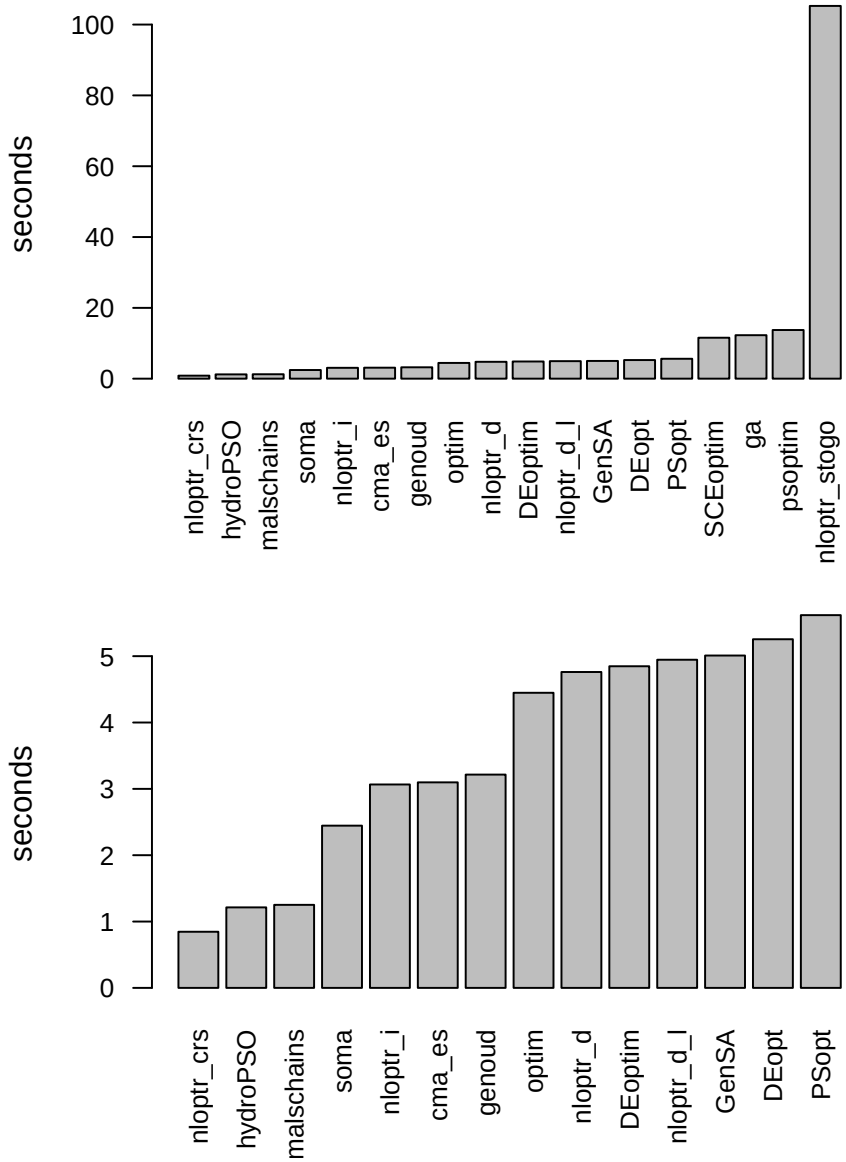


Figure 4: Summary of timing results given as the mean time calculated over all 100 runs and all four problems for which times were measured. The lower plot leaves out the four slowest methods, allowing closer inspection of the differences between the faster implementations.

`stogo`, `SCEoptim`, `ga` and `genoud`. The `malschains`, `nloptr_crs`, `hydroPSO` are among the faster implementations; `soma` is among the fastest methods for the higher dimensional (10 and 20 dimensions) objective functions tested. Interestingly, most of the implementations tested (with the exception of `stogo`) require approximately the same amount of time to return results for the 2-parameter problem as for the 20-parameter problem. A plot that summarizes the results in terms of the mean time calculated over all 100 runs and all four problems for which times were measured is given in Figure 4. Note that this summary plot obscures some important differences between performance on the individual problems, which can be seen in the plots in Appendix B.

6. Discussion, conclusions, and future work

This paper surveyed the wide variety of general-purpose methods for continuous global optimization that are currently available in R. Eighteen implementations were benchmarked on 48 objective functions collected in the new R package **globalOptTests**. The only implementation included in base R, the simulated annealing method in the `optim` function, had poor overall performance in terms of the solution quality; the user of global optimization methods in R should turn to contributed packages for better implementations.

In terms of accuracy of solutions found within 10,000 function evaluations, `stogo` (from package **nloptr** using **nloptwrap**), `genoud` from the **rgenoud** package and `GenSA` from the **GenSA** package were most capable of consistently returning a solution near the global minimum of each test function (where 'near' is taken to be within 0.005 of the global minimum). In terms of speed, `genoud` was the fastest of these three most accurate methods (though it did terminate in an error on a small number of problem instances, as described in Table 3). `GenSA` was slower than `genoud`, but not by a large factor, while `stogo` was comparatively very slow indeed, taking a factor of ten longer to return results. Note however that there was significant heterogeneity in results for both accuracy and speed among the 48 problems tested, as is evident in the plots in Appendices A and B. The reader can consult these plots for clues regarding which implementations may be promising for a given application. For instance, if only an approximate solution is needed and the objective function is very time-consuming to compute, a faster but less consistently accurate implementation may be a good choice.

Almost all of the 18 implementations tested have a host of control settings, the tuning of which may dramatically alter performance. The tests here represent how the implementations work without any such tuning; users should consult the help pages of each implementation for tips on how to optimally adjust control settings.

While the test objective functions included in **globalOptTests** have variety in certain senses, they are also all fast to evaluate and noise-free. The comparison studies here used a relatively generous budget of function evaluations; for problems that are time-consuming to evaluate, performance after fewer than 10,000 function evaluations may be of interest. The effects of noise in the objective function may also drastically alter the performance of the implementations, and should be examined in future studies.

An obvious way to extend this comparison would be to investigate how accurately and quickly the various implementations return estimates of the global optima in higher-dimensional parameter spaces. Many of the functions in **globalOptTests** can be evaluated in an arbitrary dimension (though note that the global optimum obtained via the function `getGlobalOpt` may not be correct if the dimension of the parameter vector is other than than given by `getProblemDimen`). The Rastrigin function, for instance, has a global optimum (at zero) that is independent of problem dimension; it can be evaluated in the 50-dimensional case as follows:

```
R> goTest(par = rep(1, 50), fnName = "Rastrigin", checkDim = FALSE)
```

For the 50-dimensional Rastrigin problem, at least the `GenSA` function consistently finds the global optimum when given a budget of 150,000 function evaluations.

An extension to consider the parallelization options which are available in some of the implementations considered here would also be of interest. Parallelization becomes especially critical when the objective function is expensive to evaluate.

The interested user or developer can easily extend these benchmarking studies; the **globalOptTests** package containing the objective functions is on CRAN, and the scripts used herein for empirical studies and plotting are available as supplementary information to this article.

Acknowledgments

Hans Werner Borchers provided many helpful comments on this work. Sincere thanks go to the two anonymous reviewers for their suggestions.

References

Ali MM, Khompatraporn C, Zabinsky ZB (2005). "A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems." *Journal of Global Optimization*, **31**, 635–672.

Andrews F, Guillaume J (2013). ***hydromad**: Hydrological Model Assessment and Development*. R package version 0.9-18, URL http://hydromad.catchment.org/.

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. URL http://journal.R-project.org/archive/2011-1/RJournal_2011-1_Ardia~et~al.pdf.

Ardia D, Mullen KM, Peterson BG, Ulrich J (2013). ***DEoptim**: Differential Evolution in R*. R package version 2.2-2, URL http://CRAN.R-project.org/package=DEoptim.

Bendtsen C (2012). ***pso**: Particle Swarm Optimization*. R package version 1.0.3, URL http://CRAN.R-project.org/package=pso.

Bergmeir C, Molina D, Benítez JM (2014). *Continuous Optimization Using Memetic Algorithms with Local Search Chains (MA-LS-Chains) in R*. R package version 0.2-2, URL http://CRAN.R-project.org/package=Rmalschains.

Borchers HW (2014). ***nloptwrap**: Wrapper for Package **nloptr***. R package version 0.5-7, URL http://CRAN.R-project.org/package=nloptwrap.

Burns P (1998). *S Poetry*. Burns Statistics. URL http://www.burns-stat.com/pages/Spoetry/Spoetry.pdf.

Burns P (2012a). "Another Comparison of Heuristic Optimizers." Published: 2012-08-20. Accessed: 2013-09-01, URL http://www.portfolioprobe.com/2012/08/20/another-comparison-of-heuristic-optimizers/.

Burns P (2012b). "A Comparison of Some Heuristic Optimization Methods." Published: 2012-07-23. Accessed: 2013-09-01, URL http://www.portfolioprobe.com/2012/07/23/a-comparison-of-some-heuristic-optimization-methods/.

Burns Statistics (2012). ***BurStMisc**: Burns Statistics Miscellaneous*. R package version 1.00, URL http://CRAN.R-project.org/package=BurStMisc.

Clayden J, based on the work of Ivan Zelinka (2011). ***soma**: General-Purpose Optimisation with the Self-Organising Migrating Algorithm*. R package version 1.1.0, URL http://CRAN.R-project.org/package=soma.

Czyzyk J, Mesnier MP, Moré JJ (1998). "The **NEOS** Server." *IEEE Computational Science Engineering*, **5**(3), 68 –75.

Dueck G, Scheuer T (1990). "Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing." *Journal of Computational Physics*, **90**(1), 161 – 175.

Eddelbuettel D (2013). ***RcppDE**: Global Optimization by Differential Evolution in C++*. R package version 0.1.2, URL http://CRAN.R-project.org/package=RcppDE.

Floudas CA, Gounaris CE (2009). "A Review of Recent Advances in Global Optimization." *Journal of Global Optimization*, **45**(1), 3–38.

Francke T (2012). ***ppso**: Particle Swarm Optimization and Dynamically Dimensioned Search, Optionally Using Parallel Computing Based on **Rmpi***. R package version 0.9-952, URL http://www.RForge.net/ppso/.

Gablonsky JM, Kelley CT (2001). "A Locally-Biased Form of the DIRECT Algorithm." *Journal of Global Optimization*, **21**, 27–37.

GAMS Development Corporation and GAMS Software GmbH (2013). "Selected Continuous Global Optimization Test Problems." Accessed: 2013-09-01, URL http://www.gamsworld.org/performance/selconglobal/selcongloballib.htm.

Ghalanos A (2014). ***parma**: Portfolio Allocation and Risk Management Applications*. R package version 1.5-1, URL http://CRAN.R-project.org/package=parma.

Gilli M, Maringer D, Schumann E (2011). *Numerical Methods and Optimization in Finance*. Academic Press. URL http://nmof.net/.

Hansen N, Ostermeier A (1996). "Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation." In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 312–317.

Holland JH (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press.

Horst R, Pardalos PM, Thoai NV (2000). *Introduction to Global Optimization*. Springer-Verlag.

Jensen PA, Bard JF (2002). *Operations Research Models and Methods*. John Wiley & Sons.

Johnson SG (2013). *The NLopt Nonlinear-Optimization Package, Version 2-3*. URL http://ab-initio.mit.edu/nlopt.

Jones DR, Perttunen CD, Stuckman BE (1993). "Lipschitzian Optimization without the Lipschitz Constant." *Journal of Optimization Theory and Applications*, **79**(1), 157–181.

Kaelo P, Ali M (2006). "Some Variants of the Controlled Random Search Algorithm for Global Optimization." *Journal of Optimization Theory and Applications*, **130**, 253–264.

Kan AR, Timmer G (1987). "Stochastic Global Optimization Methods Part I: Clustering Methods." *Mathematical Programming*, **39**, 27–56.

Kennedy J, Eberhart R (1995). "Particle Swarm Optimization." In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pp. 1942–1948.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983). "Optimization by Simulated Annealing." *Science*, **220**, 671–680.

Madsen K, Zertchaninov S, Zilinskas A (1998). *Global Optimization Using Branch-and-Bound*. Report in the `stogo` subdirectory of the **NLopt** source code, URL http://ab-initio.mit.edu/nlopt/nlopt-2.3.tar.gz.

Mebane WR, Sekhon JS (2011). "Genetic Optimization Using Derivatives: The **rgenoud** Package for R." *Journal of Statistical Software*, **42**(11), 1–26.

Mebane WR, Sekhon JS (2013). ***rgenoud**: R version of GENetic Optimization Using Derivatives*. R package version 5.7-12, URL http://CRAN.R-project.org/package=rgenoud.

Molina D, Lozano M, García-Martínez C, Herrera F (2010). "Memetic Algorithms for Continuous Optimisation based on Local Search Chains." *Evolutionary Computation*, **18**(1), 27–63.

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/.

Neumaier A (2004). "Complete Search in Continuous Global Optimization and Constraint Satisfaction." In A Iserles (ed.), *Acta Numerica 2004*. Cambridge University Press. URL http://www.mat.univie.ac.at/~neum/ms/glopt03.pdf.

Neumaier A (2013). "Global Optimization." Accessed: 2013-04-07, URL http://www.mat.univie.ac.at/~neum/glopt.html.

Nocedal J, Wright SJ (2006). *Numerical Optimization*. 2nd edition. Springer-Verlag.

Pardalos PM, Romeijn HE (2002). *Handbook of Global Optimization Volume 2*. Springer-Verlag.

Pfaff B (2012). ***rneos**: XML-RPC Interface to **NEOS***. R package version 0.2-7, URL http://CRAN.R-project.org/package=rneos.

Price KV, Storn RM, Lampinen JA (2006). *Differential Evolution – A Practical Approach to Global Optimization*. Natural Computing. Springer-Verlag.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Rudin W (1976). *Principles of Mathematical Analysis*. International Series in Pure and Applied Mathematics, 3rd edition. McGraw-Hill International.

Runarsson TP, Yao X (2005). "Search Biases in Constrained Evolutionary Optimization." *IEEE Transactions on Systems, Man, and Cybernetics C*, **35**(2), 233 –243.

Satman MH (2014). ***mcga**: Machine Coded Genetic Algorithms for Real-Valued Optimization Problems*. R package version 2.0.9, URL http://CRAN.R-project.org/package=mcga.

Scholz D (2012). *Deterministic Global Optimization: Geometric Branch-and-Bound Methods and Their Applications*. 1st edition. Springer-Verlag.

Schumann E (2013). ***NMOF**: Numerical Methods and Optimization in Finance*. R package version 0.28-2, URL http://CRAN.R-project.org/package=NMOF.

Scrucca L (2014). ***GA**: Genetic Algorithms*. R package version 2.1, URL http://CRAN.R-project.org/package=GA.

Sekhon JS, Mebane WR (1998). "Genetic Optimization Using Derivatives: Theory and Application to Nonlinear Models." *Political Analysis*, **7**, 189–213.

Theussl S (2014). *CRAN Task View: Optimization and Mathematical Programming*. Version 2014-08-08, URL http://CRAN.R-project.org/view=Optimization.

Tolson BA, Shoemaker CA (2007). "Dynamically Dimensioned Search Algorithm for Computationally Efficient Watershed Model Calibration." *Water Resources Research*, **43**(1), W01413.

Trautmann H, Mersmann O, Arnu D (2011). ***cmaes**: Covariance Matrix Adapting Evolutionary Strategy*. R package version 1.0-11, URL http://CRAN.R-project.org/package=cmaes.

University of Wisconsin-Madison (2013). *The **NEOS** Server*. Accessed: 2013-09-01, URL http://neos-guide.org/.

Weise T (2009). *Global Optimization Algorithms – Theory and Application*. Thomas Weise. URL http://www.it-weise.de/projects/book.pdf.

Xiang Y, Gubian S, Suomela B, Hoeng J (2013). "Generalized Simulated Annealing for Efficient Global Optimization: The **GenSA** Package for R." *The R Journal*, **5**(1), 13–28. URL http://journal.R-project.org/archive/2013-1/xiang-gubian-suomela-etal.pdf.

Ypma J (2014). ***nloptr**: R Interface to **NLopt***. R package version 1.04, URL http://CRAN.R-project.org/package=nloptr.

Zambrano-Bigiarini M (2013). ***hydroPSO**: Particle Swarm Optimisation, with Focus on Environmental Models*. R package version 0.3-3, URL http://CRAN.R-project.org/package=hydroPSO.

Zelinka I (2004). "SOMA – Self Organizing Migrating Algorithm." In G Onwubolu, BV Babu (eds.), *New Optimization Techniques in Engineering*. Springer-Verlag.

Zertchaninov S, Madsen K (1998). "A C++ Programme for Global Optimization." *Technical Report IMM-REP-1998-04*, Department of Mathematical Modelling, Technical University of Denmark. Report in the `stogo` subdirectory of the **NLopt** source code, URL `http://ab-initio.mit.edu/nlopt/nlopt-2.3.tar.gz`.

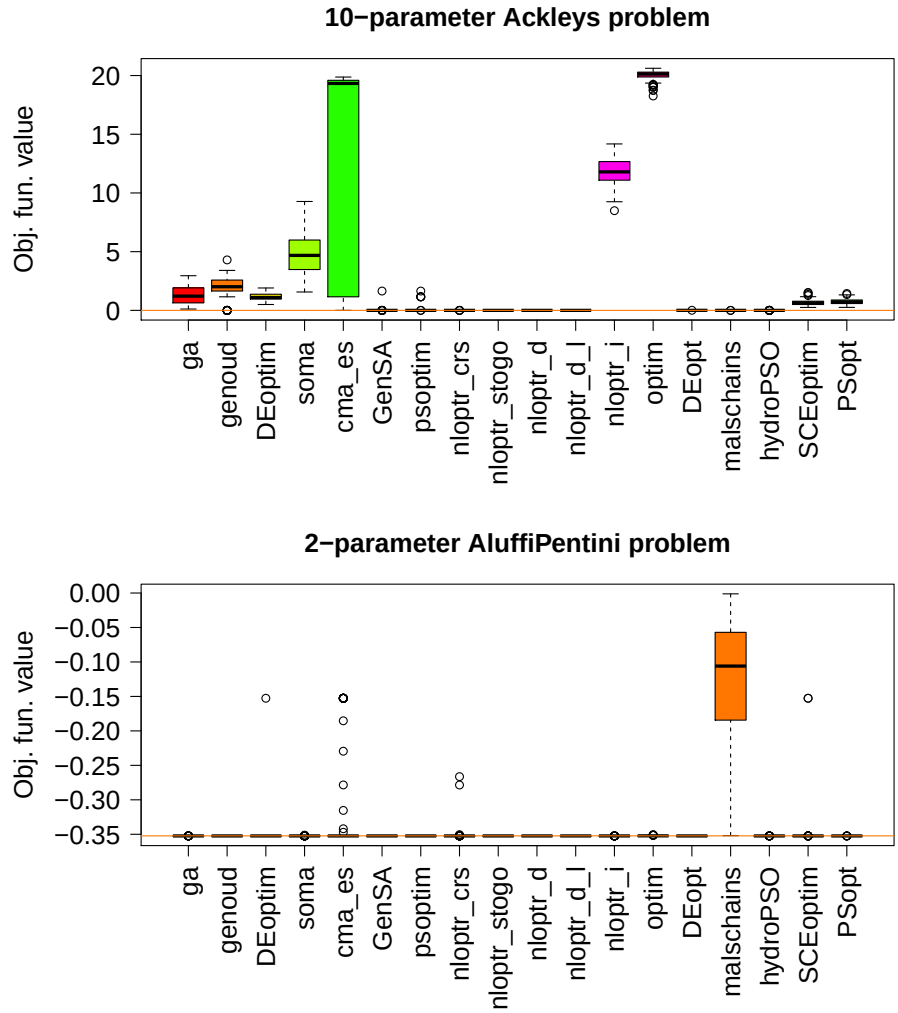
Zhang J, Sanderson AC (2009). "JADE: Adaptive Differential Evolution With Optional External Archive." *Evolutionary Computation, IEEE Transactions on*, **13**(5), 945 –958.

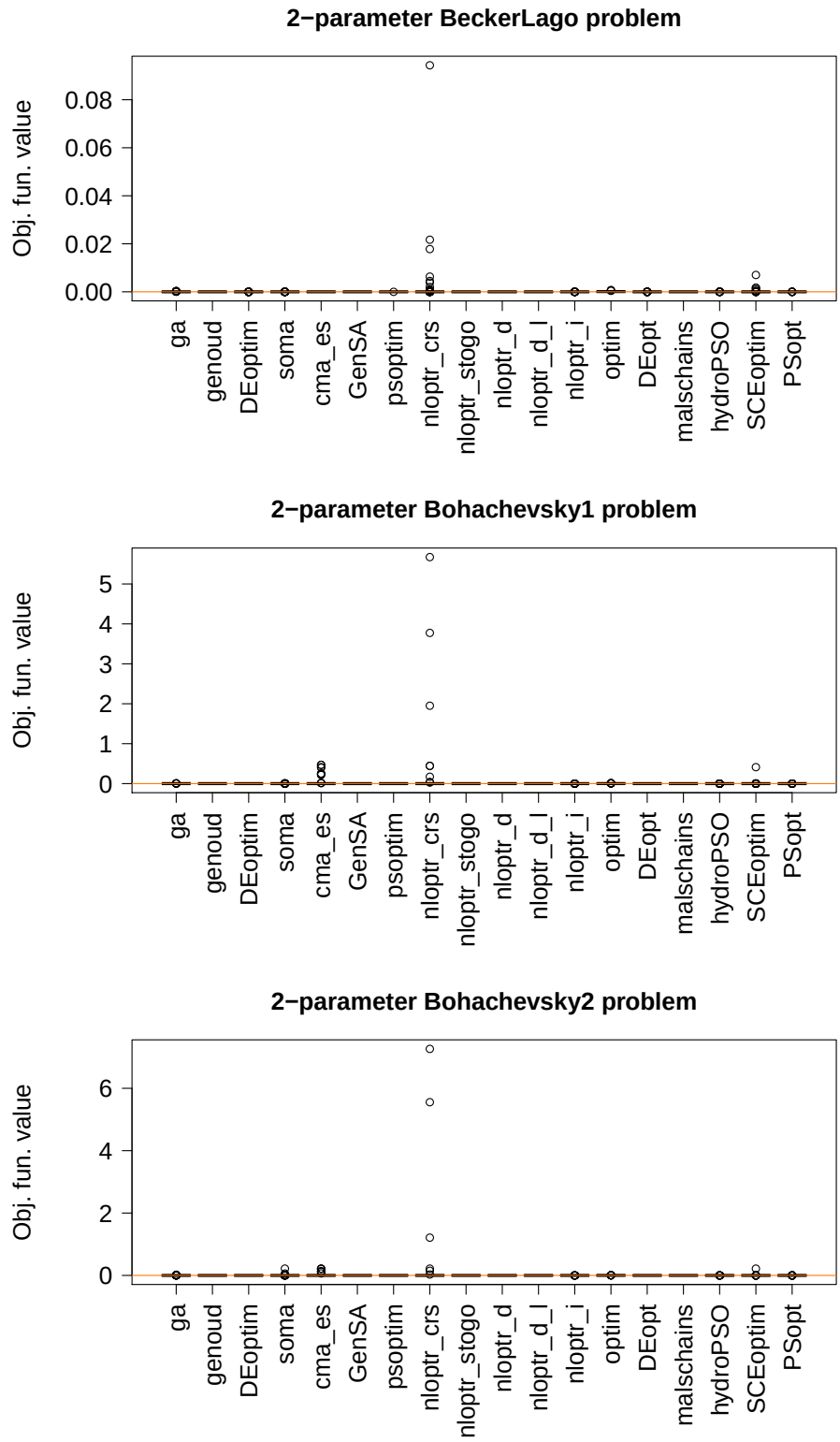
A. Boxplots of solutions

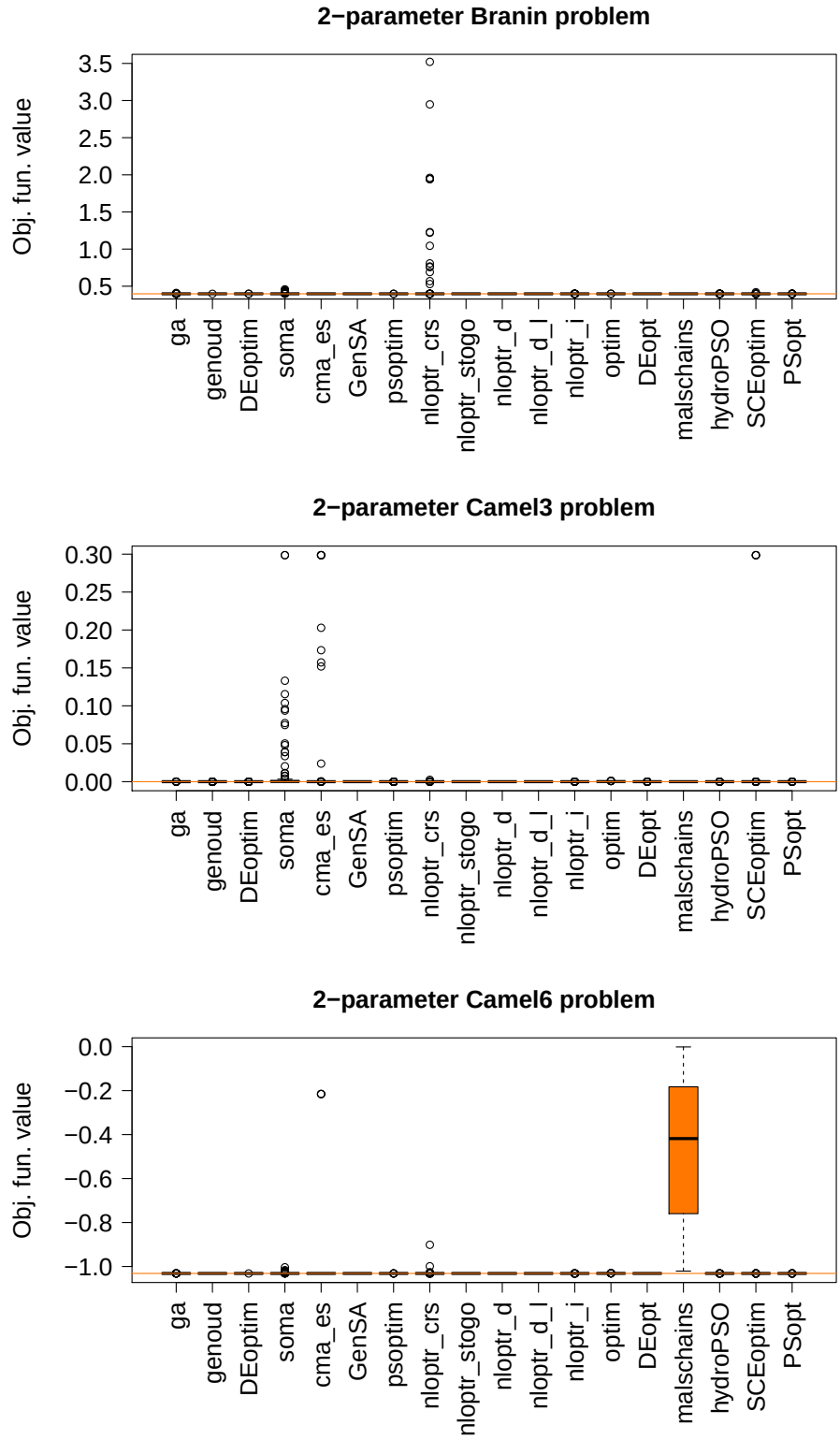
This appendix contains boxplots of solutions returned by the 18 implementations for 48 objective functions. The global optimum of the functions (within the parameter bounds) is demarcated with a horizontal red line in all plots. The lower and upper bounds on parameter values do not appear to have had an effect in the `DEopt` technique, in that parameter vectors outside of the bounds were sometimes returned by this function (e.g., see results for the `Schwefel` function).

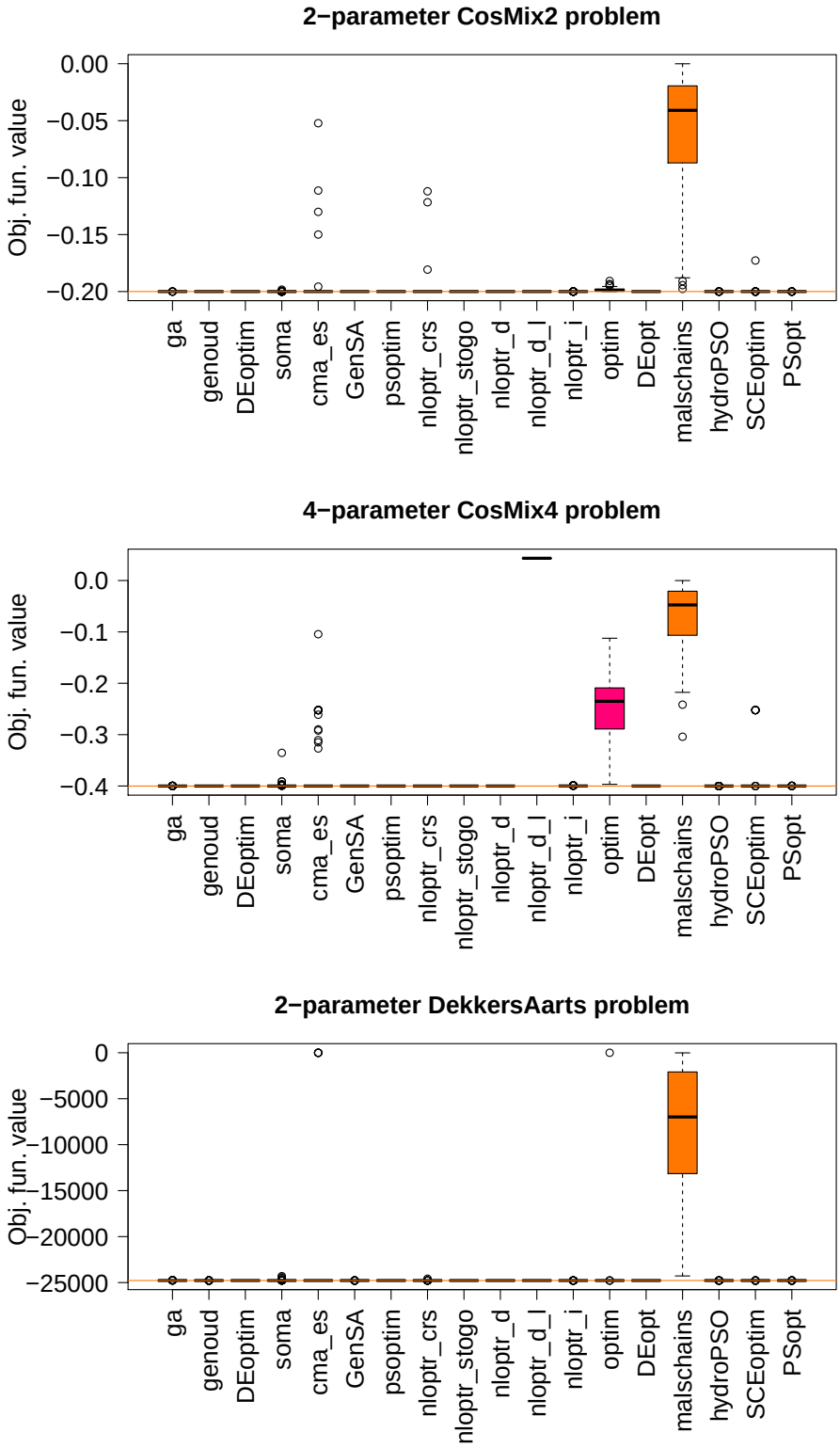
For the purposes of presentation, the y-limits of some plots do not encompass all solutions returned. The outlying values not shown are as follows: For the `LM2n5` problem, 2 values $>$ 10 returned by `optim`; for the `PriceTransistor` problem, 2 values $>$ 1000 returned by `optim`. The erroneous values of zero returned by `malschains` in one case each for the `Branin` and `GoldPrice` functions were also not shown.

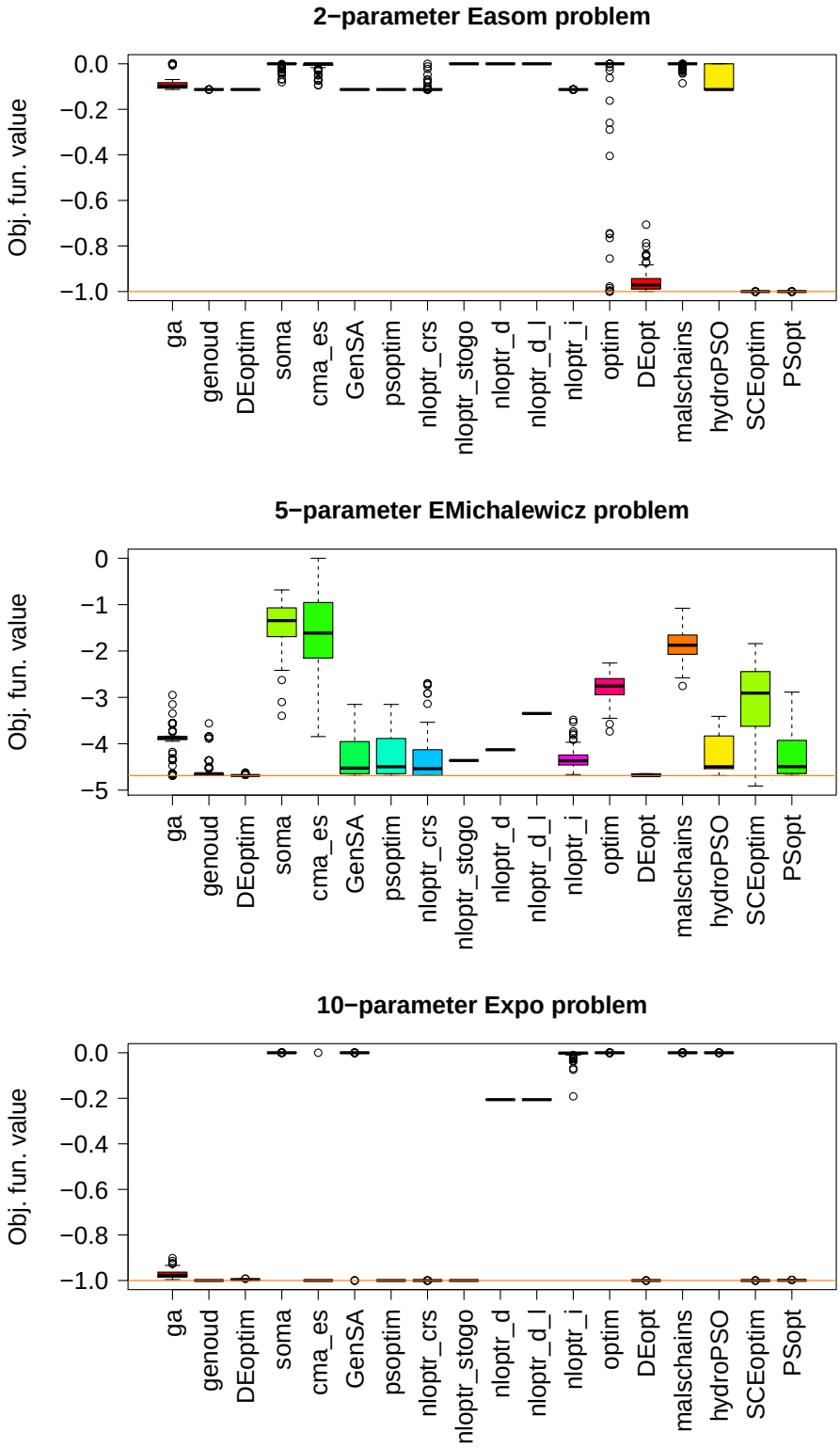
The data described in these plots (along with scripts to generate and plot the results) is included as supplementary information to this paper.

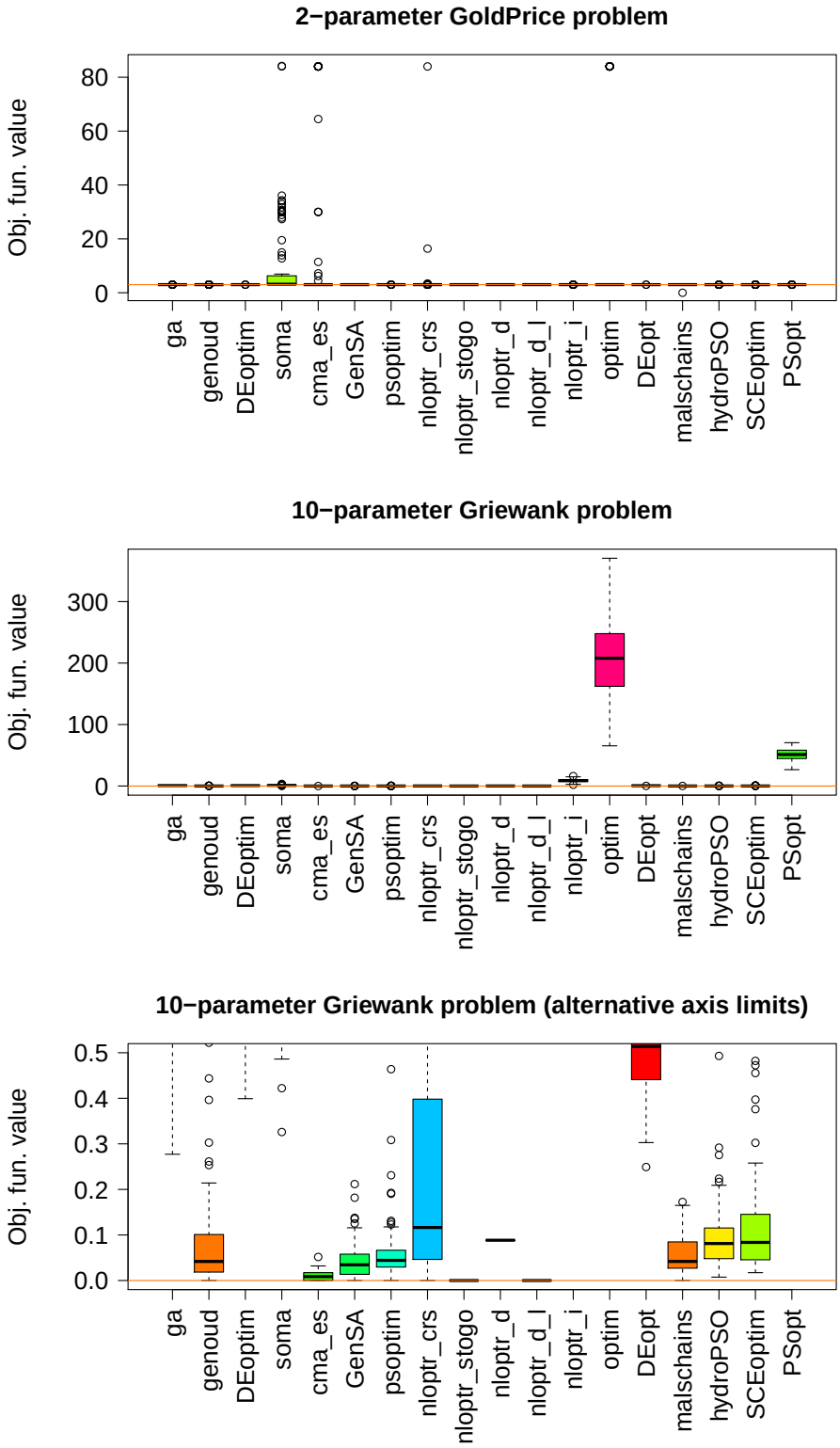


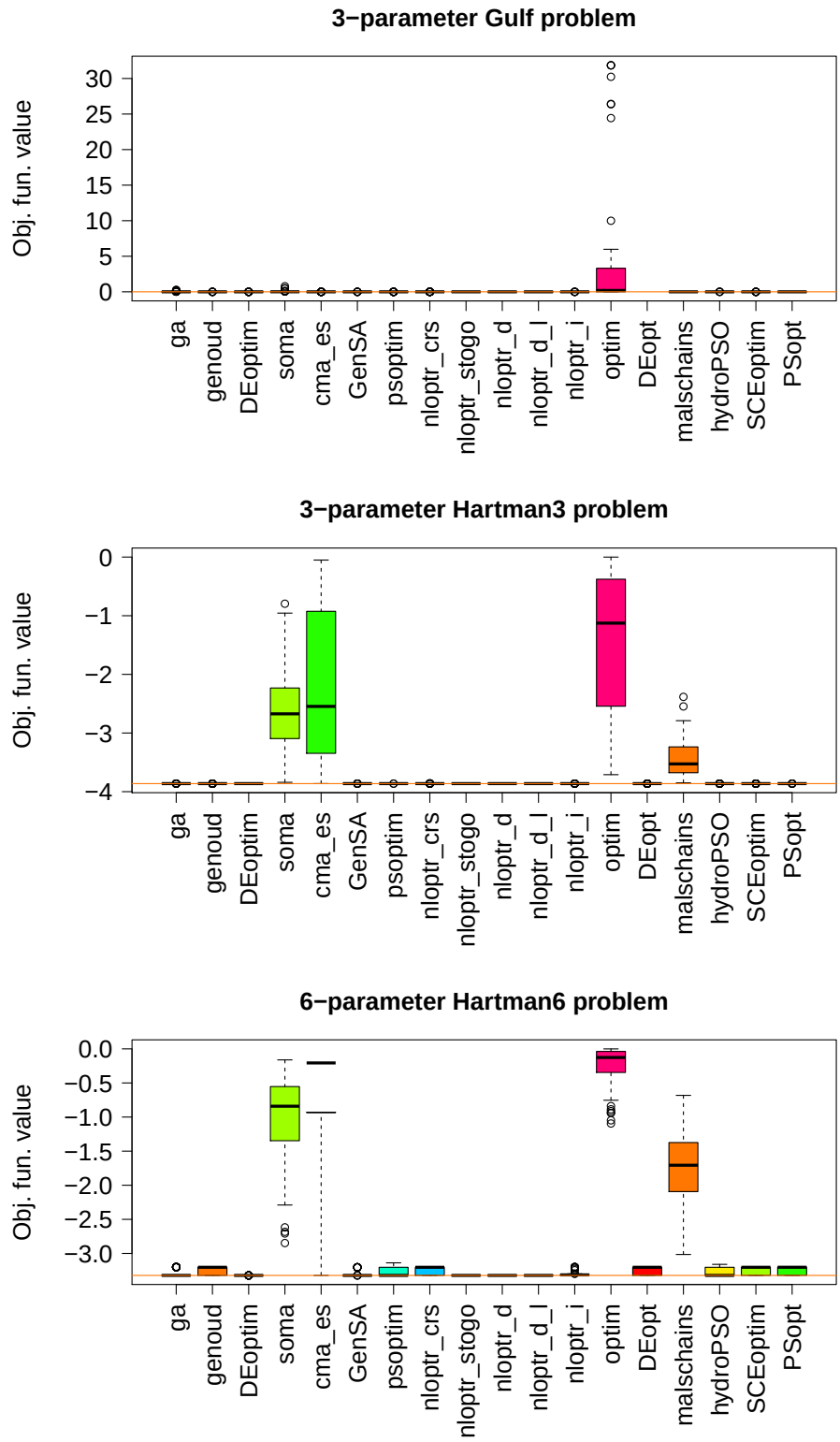

2−parameter BeckerLago problem
Obj. fun. value
0.00
0.02
0.04
0.06
0.08
2−parameter Bohachevsky1 problem
Obj. fun. value
0
1
2
3
4
5
2−parameter Bohachevsky2 problem
Obj. fun. value
0
2
4
6
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

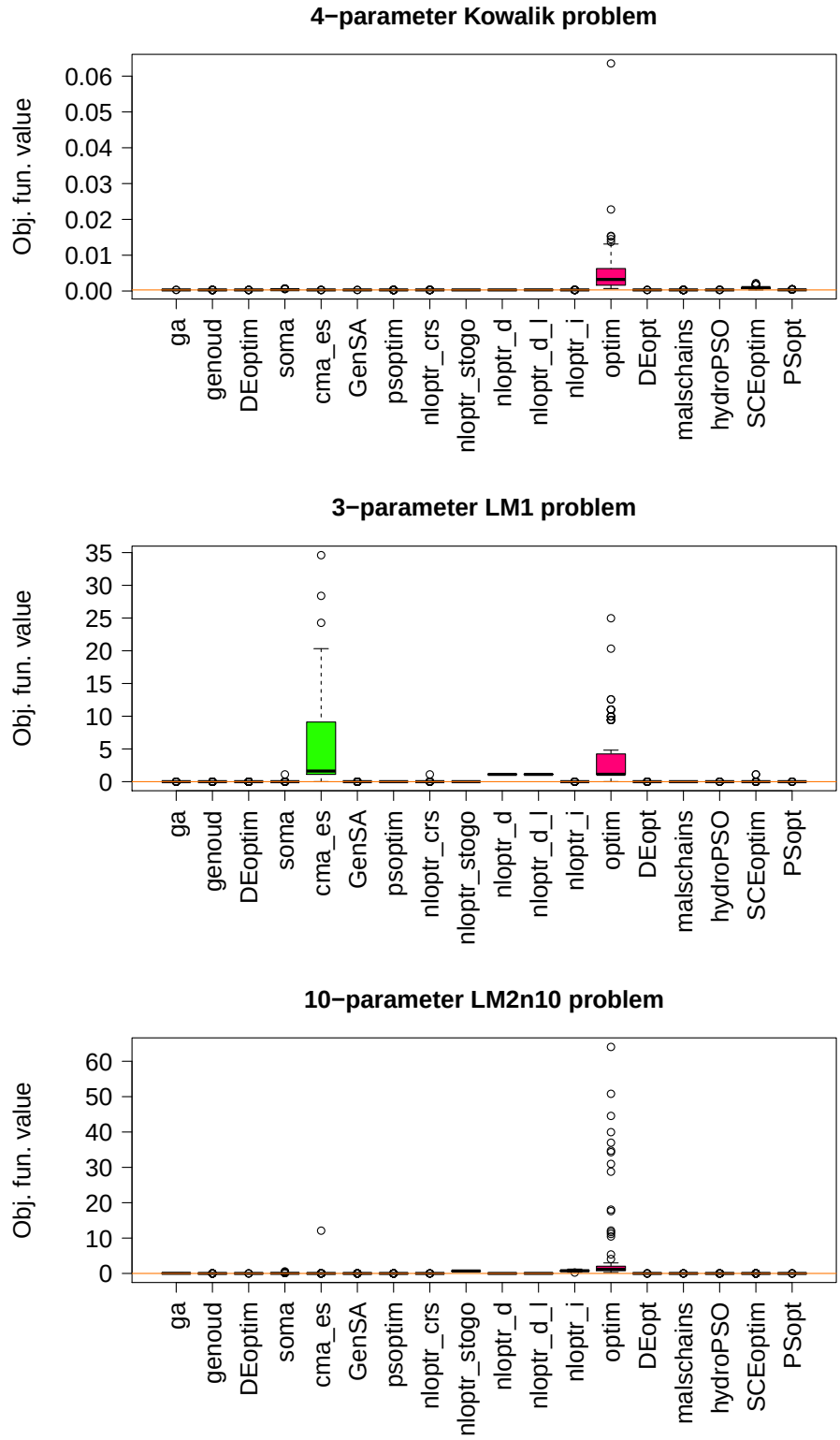

2−parameter Branin problem
Obj. fun. value
3.5
3.0
2.5
2.0
1.5
1.0
0.5
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
2−parameter Camel3 problem
Obj. fun. value
0.30
0.25
0.20
0.15
0.10
0.05
0.00
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
2−parameter Camel6 problem
Obj. fun. value
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

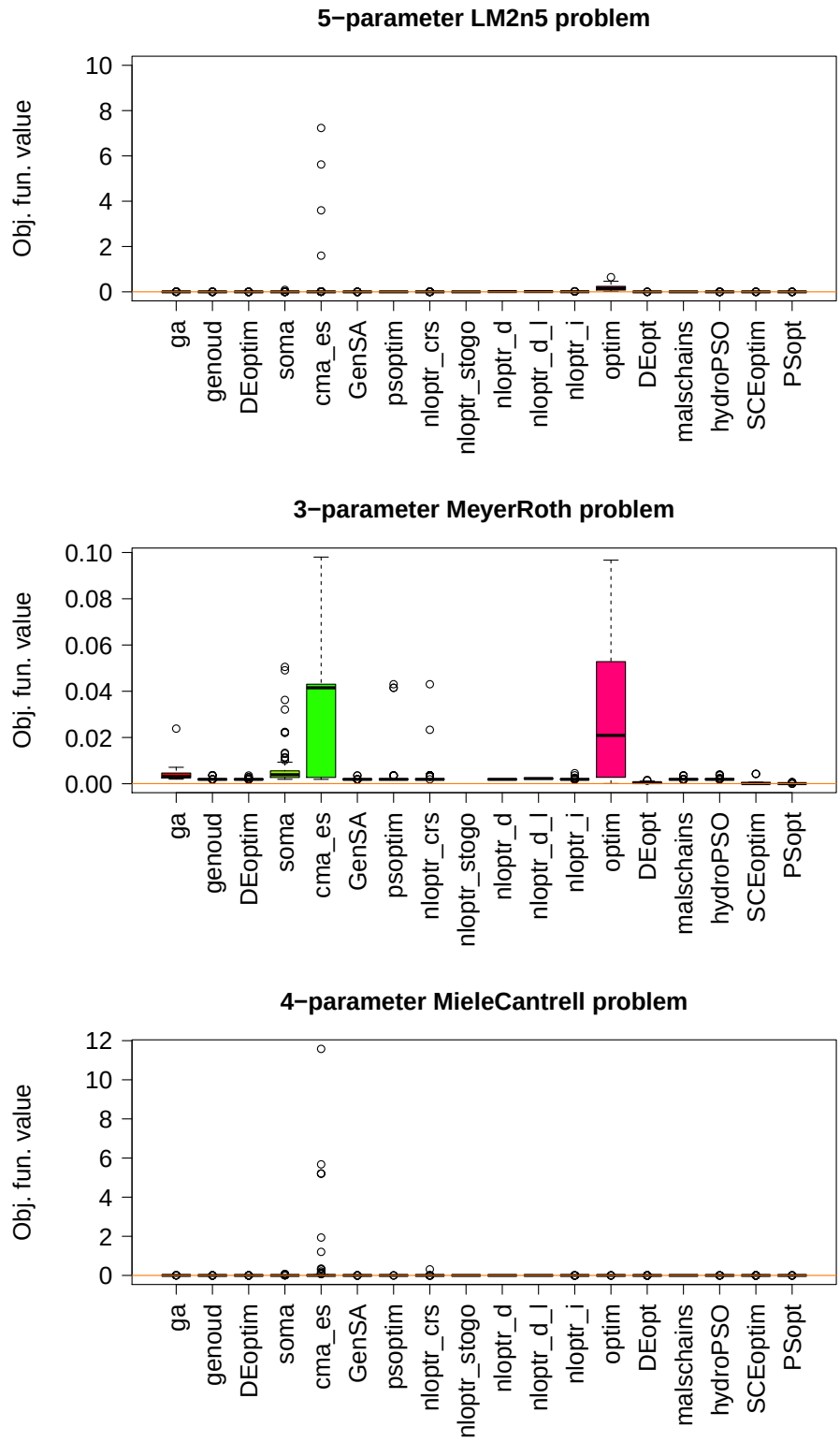

2−parameter CosMix2 problem
Obj. fun. value
0.00
−0.05
−0.10
−0.15
−0.20
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
4−parameter CosMix4 problem
Obj. fun. value
0.0
−0.1
−0.2
−0.3
−0.4
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
2−parameter DekkersAarts problem
Obj. fun. value
0
−5000
−10000
−15000
−20000
−25000
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

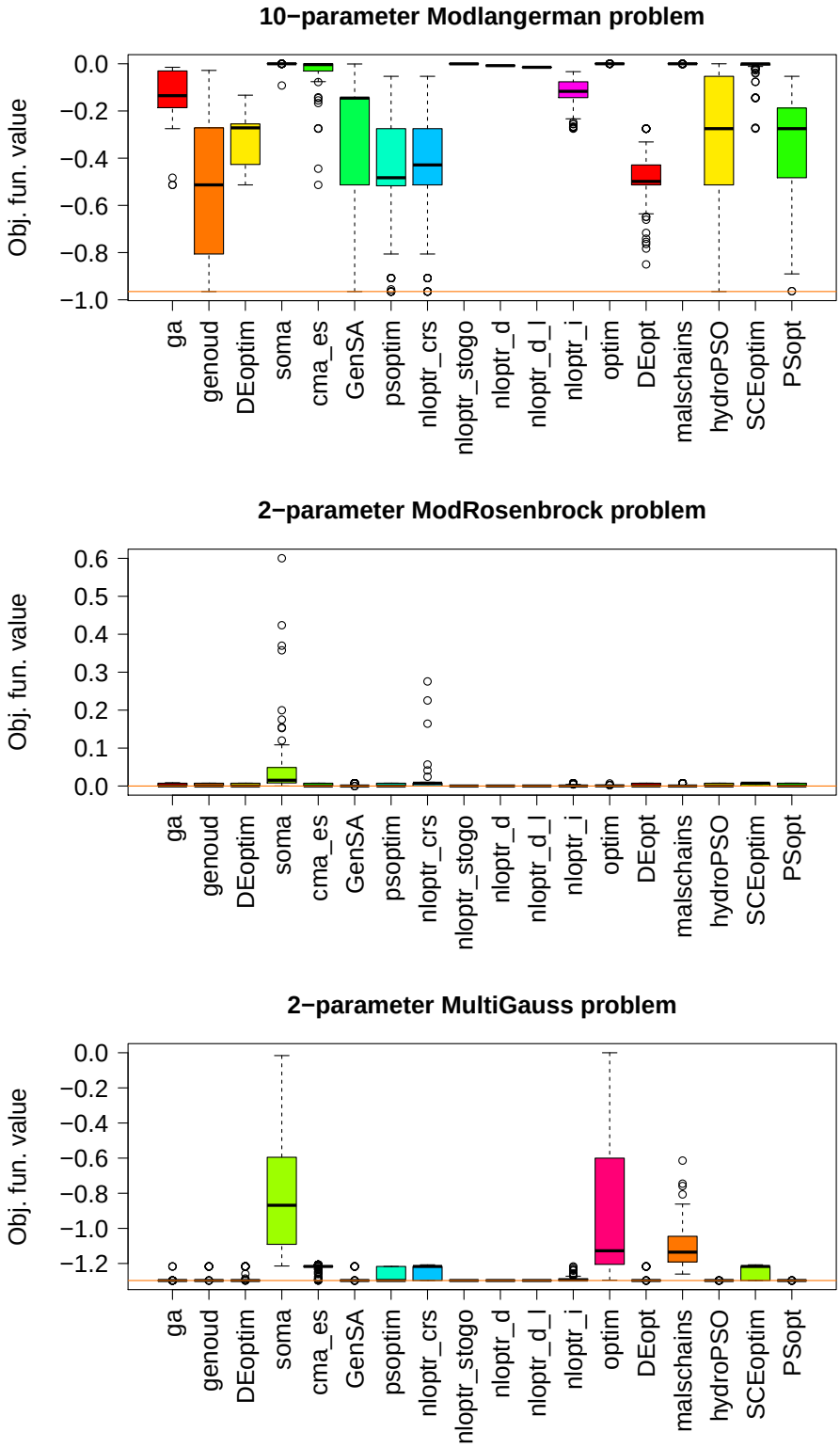

2−parameter Easom problem
Obj. fun. value
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
5−parameter EMichalewicz problem
Obj. fun. value
0
−1
−2
−3
−4
−5
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter Expo problem
Obj. fun. value
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

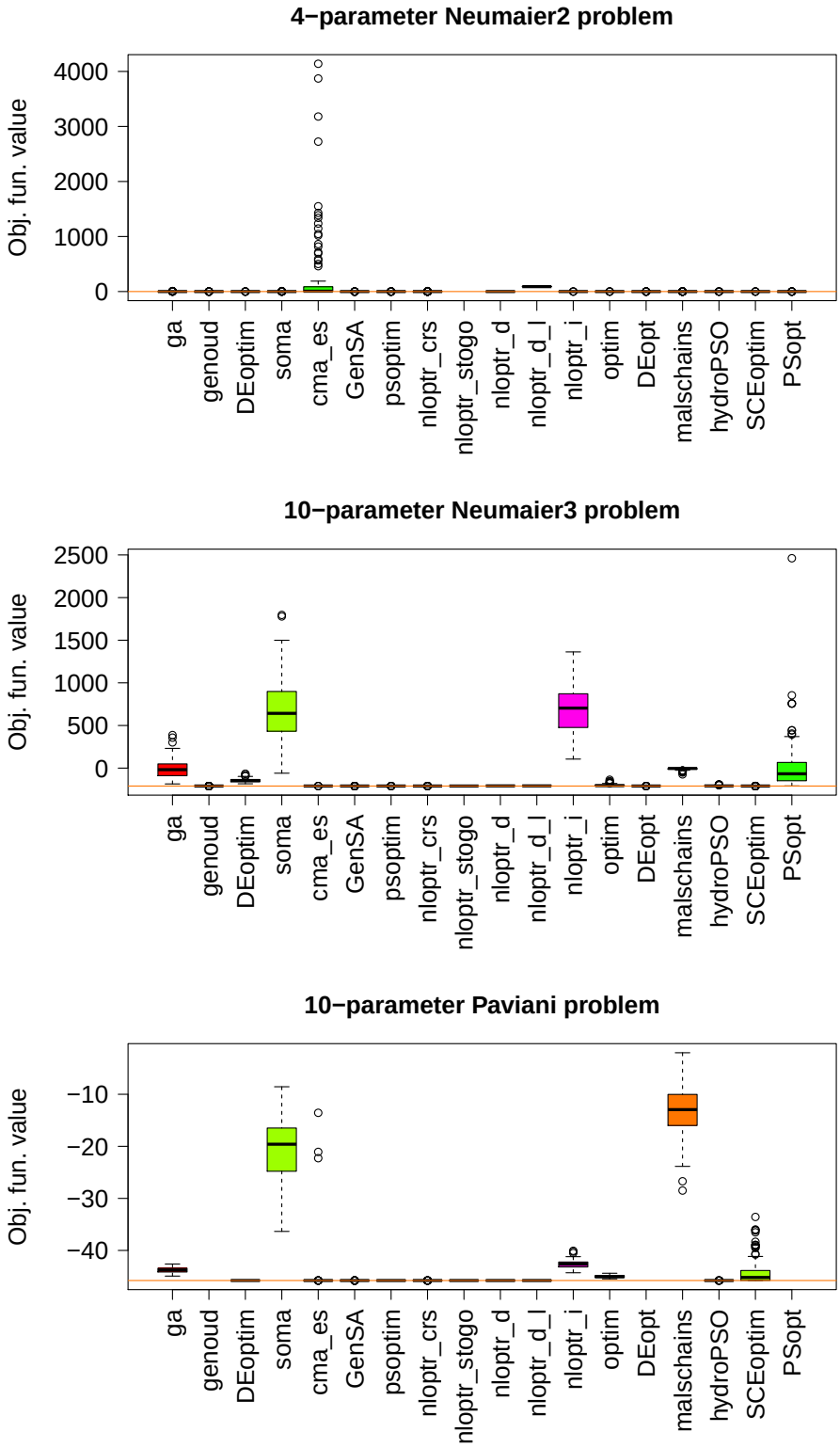

2−parameter GoldPrice problem
Obj. fun. value
0
20
40
60
80
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter Griewank problem
Obj. fun. value
0
100
200
300
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter Griewank problem (alternative axis limits)
Obj. fun. value
0.0
0.1
0.2
0.3
0.4
0.5
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

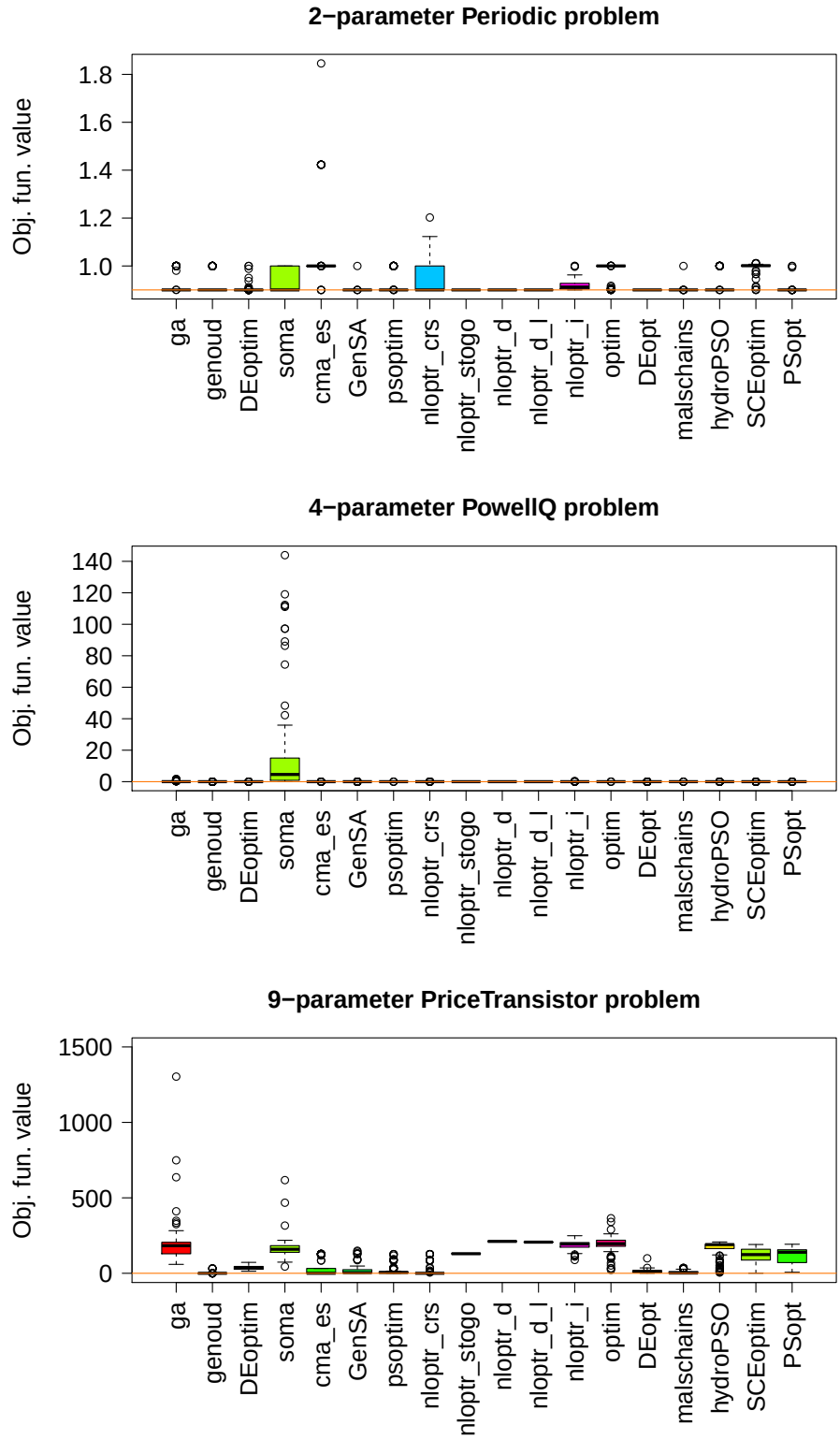

3−parameter Gulf problem
Obj. fun. value
30
25
20
15
10
5
0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
3−parameter Hartman3 problem
Obj. fun. value
0
−1
−2
−3
−4
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
6−parameter Hartman6 problem
Obj. fun. value
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

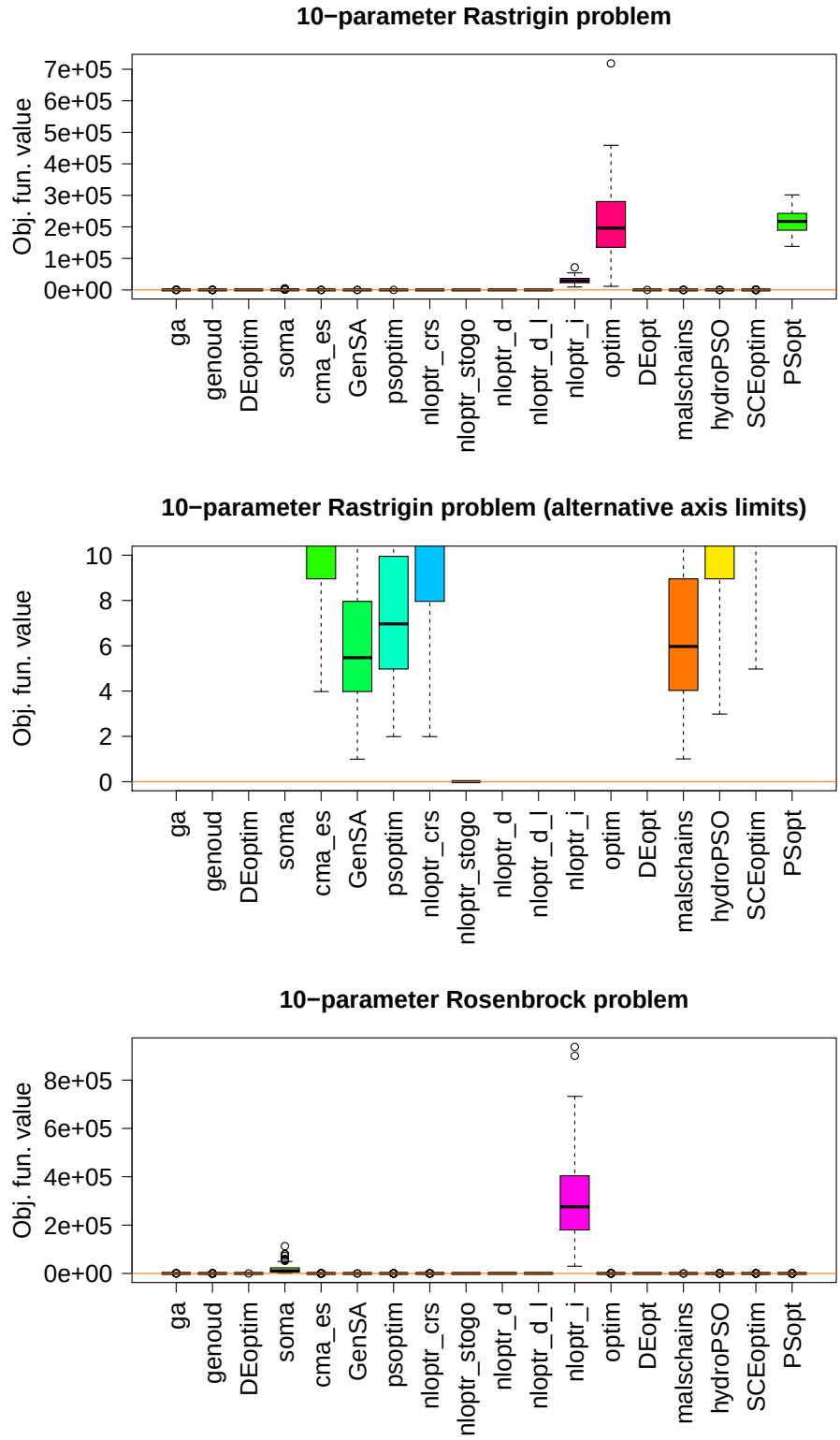

4−parameter Kowalik problem
Obj. fun. value
0.06
0.05
0.04
0.03
0.02
0.01
0.00
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
3−parameter LM1 problem
Obj. fun. value
35
30
25
20
15
10
5
0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter LM2n10 problem
Obj. fun. value
60
50
40
30
20
10
0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

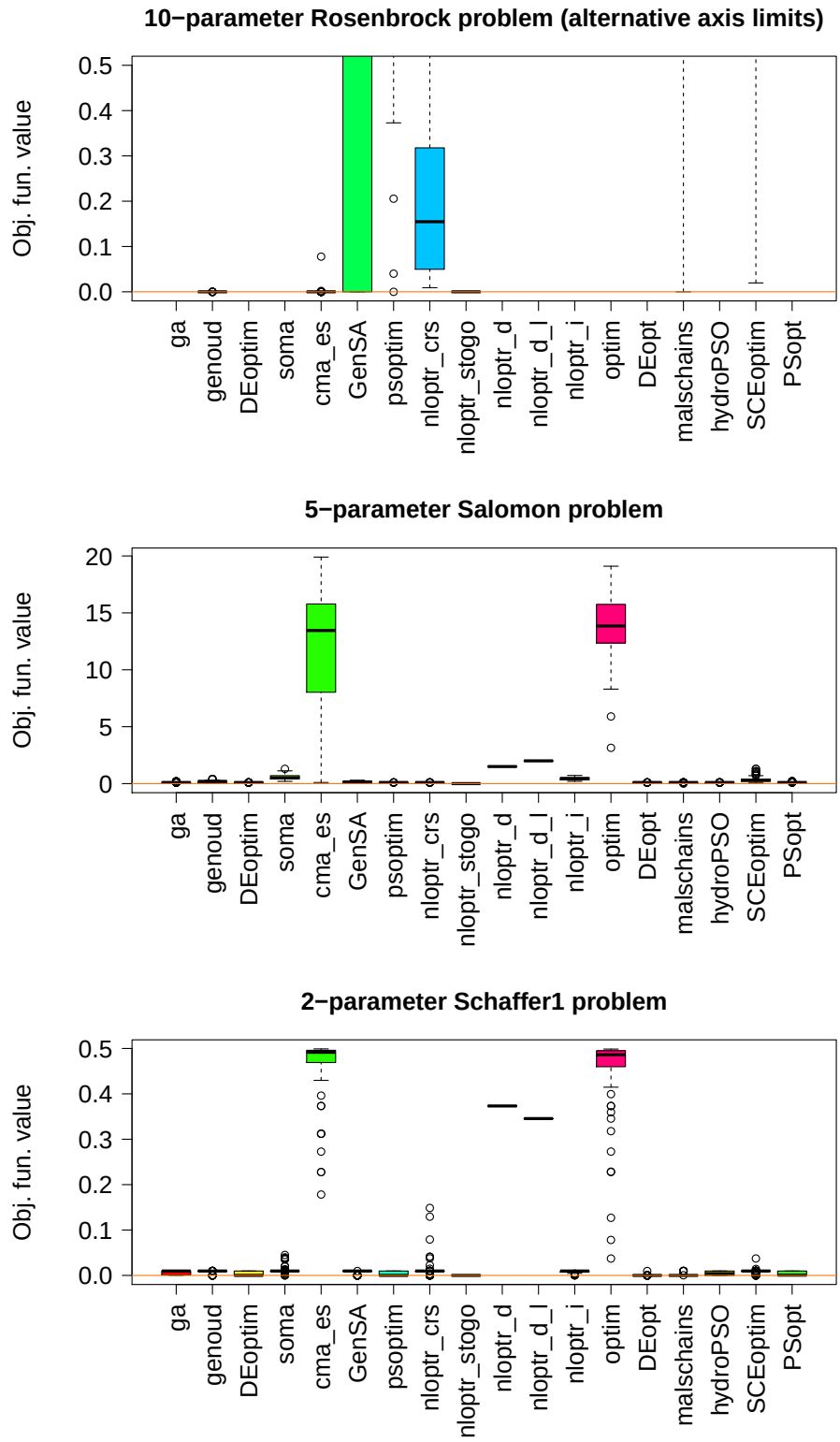

5−parameter LM2n5 problem
Obj. fun. value
0
2
4
6
8
10
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
3−parameter MeyerRoth problem
Obj. fun. value
0.00
0.02
0.04
0.06
0.08
0.10
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
4−parameter MieleCantrell problem
Obj. fun. value
0
2
4
6
8
10
12
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

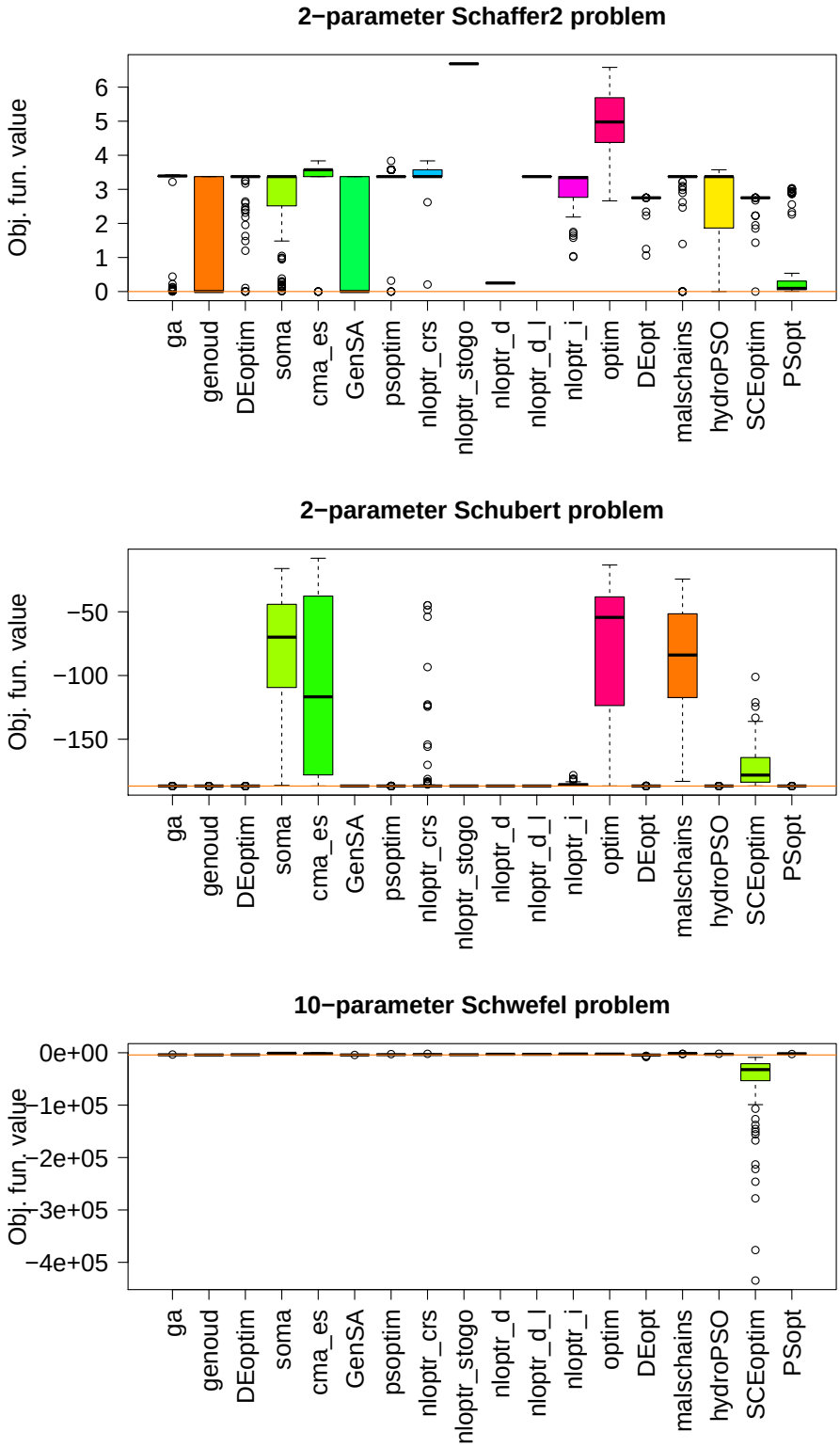

10−parameter Modlangerman problem
Obj. fun. value
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
2−parameter ModRosenbrock problem
Obj. fun. value
0.6
0.5
0.4
0.3
0.2
0.1
0.0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
2−parameter MultiGauss problem
Obj. fun. value
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

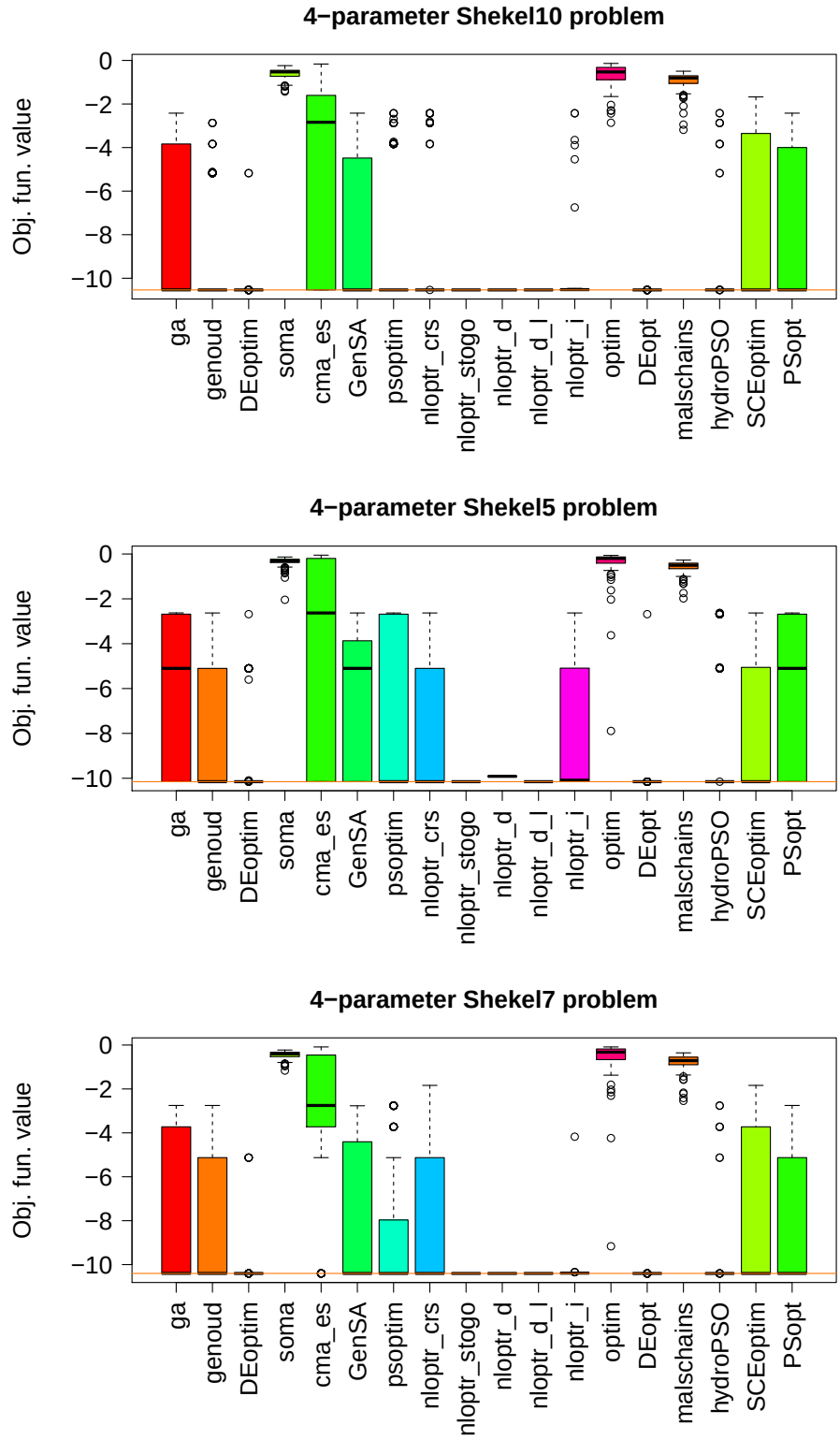

4−parameter Neumaier2 problem
Obj. fun. value
4000
3000
2000
1000
0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter Neumaier3 problem
Obj. fun. value
2500
2000
1500
1000
500
0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter Paviani problem
Obj. fun. value
−10
−20
−30
−40
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

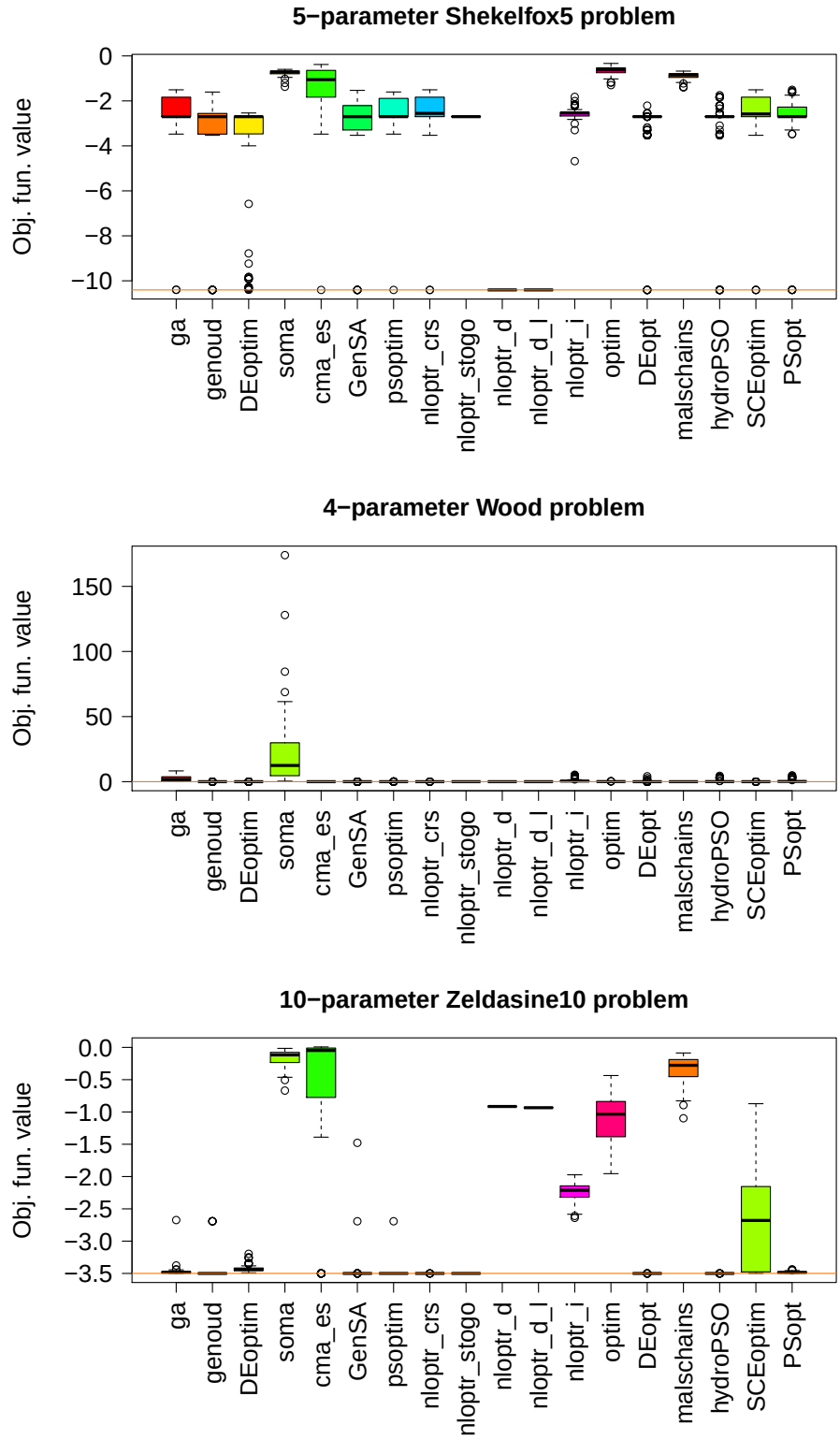

2−parameter Periodic problem
Obj. fun. value
1.8
1.6
1.4
1.2
1.0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
4−parameter PowellQ problem
Obj. fun. value
140
120
100
80
60
40
20
0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
9−parameter PriceTransistor problem
Obj. fun. value
1500
1000
500
0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

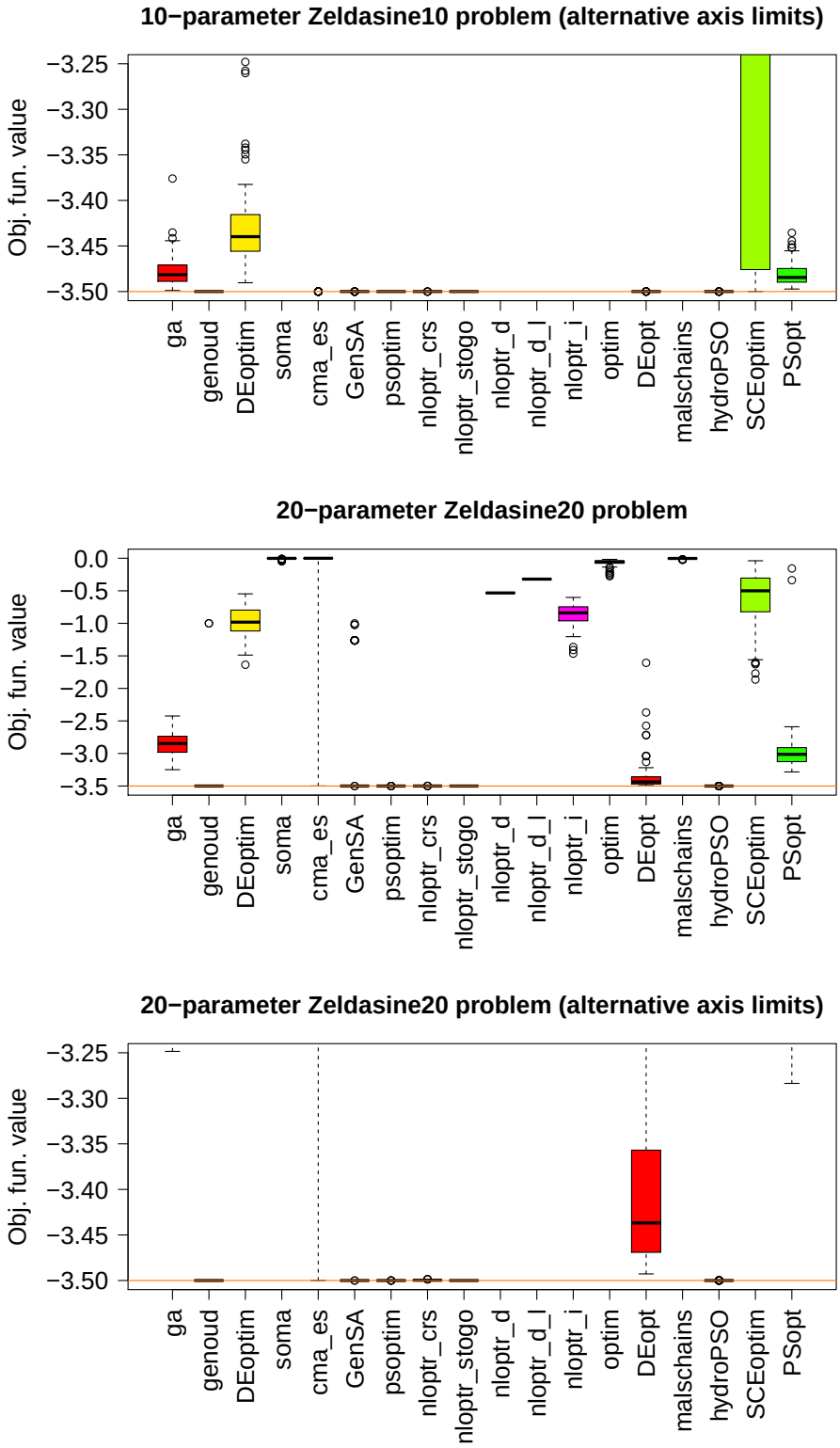

10−parameter Rastrigin problem
Obj. fun. value
7e+05
6e+05
5e+05
4e+05
3e+05
2e+05
1e+05
0e+00
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter Rastrigin problem (alternative axis limits)
Obj. fun. value
10
8
6
4
2
0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter Rosenbrock problem
Obj. fun. value
8e+05
6e+05
4e+05
2e+05
0e+00
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt


10−parameter Rosenbrock problem (alternative axis limits)
Obj. fun. value
0.0
0.1
0.2
0.3
0.4
0.5
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
5−parameter Salomon problem
Obj. fun. value
0
5
10
15
20
2−parameter Schaffer1 problem
Obj. fun. value
0.0
0.1
0.2
0.3
0.4
0.5


2−parameter Schaffer2 problem
Obj. fun. value
0
1
2
3
4
5
6
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
2−parameter Schubert problem
Obj. fun. value
−50
−100
−150
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter Schwefel problem
Obj. fun. value
0e+00
−1e+05
−2e+05
−3e+05
−4e+05
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

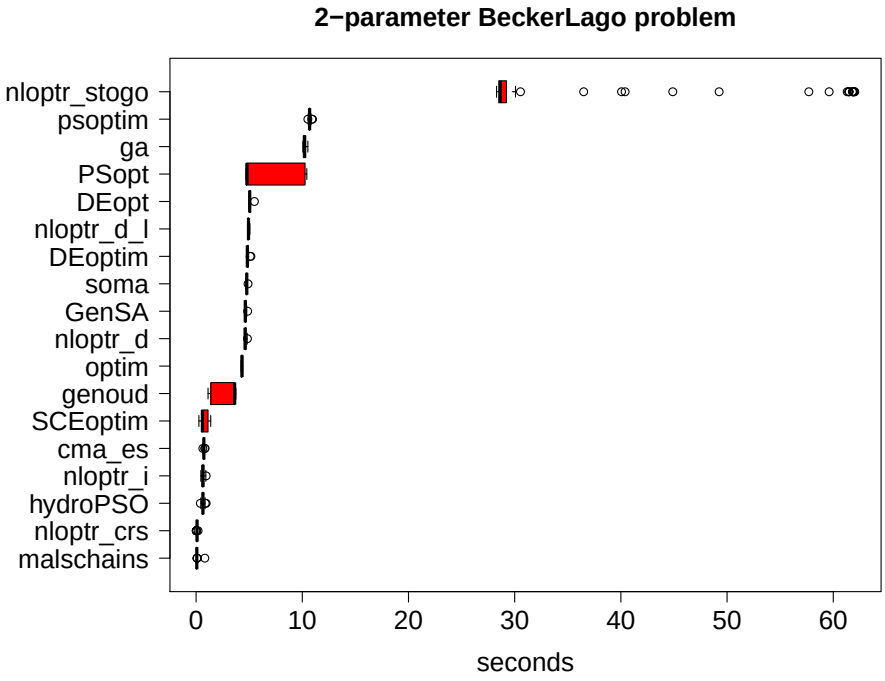

4−parameter Shekel10 problem
4−parameter Shekel5 problem
4−parameter Shekel7 problem
Obj. fun. value
0
−2
−4
−6
−8
−10
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

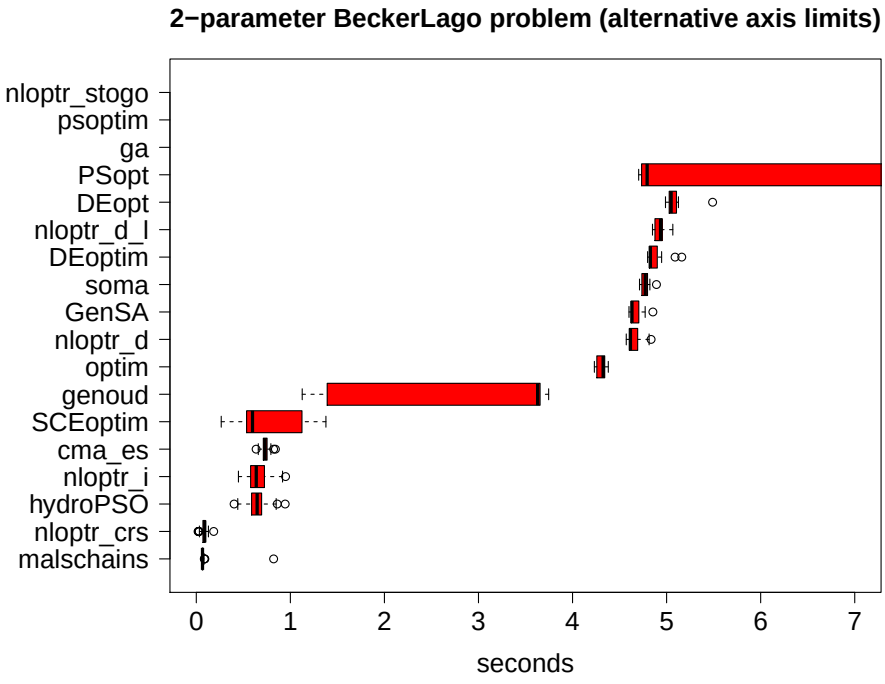

5−parameter Shekelfox5 problem
Obj. fun. value
0
−2
−4
−6
−8
−10
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
4−parameter Wood problem
Obj. fun. value
150
100
50
0
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
10−parameter Zeldasine10 problem
Obj. fun. value
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

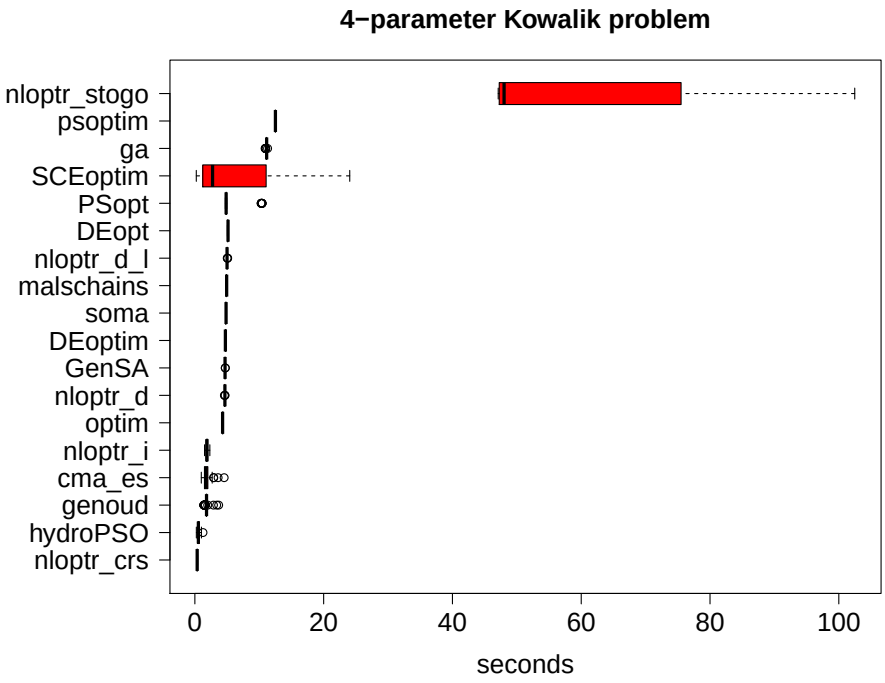

10−parameter Zeldasine10 problem (alternative axis limits)
Obj. fun. value
−3.25
−3.30
−3.35
−3.40
−3.45
−3.50
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
20−parameter Zeldasine20 problem
Obj. fun. value
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt
20−parameter Zeldasine20 problem (alternative axis limits)
Obj. fun. value
−3.25
−3.30
−3.35
−3.40
−3.45
−3.50
ga
genoud
DEoptim
soma
cma_es
GenSA
psoptim
nloptr_crs
nloptr_stogo
nloptr_d
nloptr_d_l
nloptr_i
optim
DEopt
malschains
hydroPSO
SCEoptim
PSopt

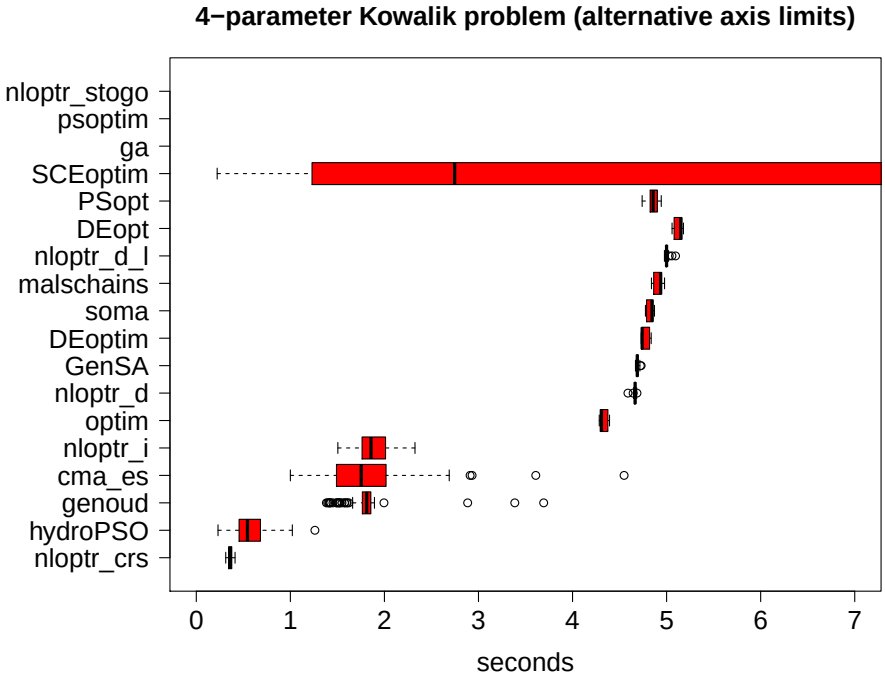
B. Timing results

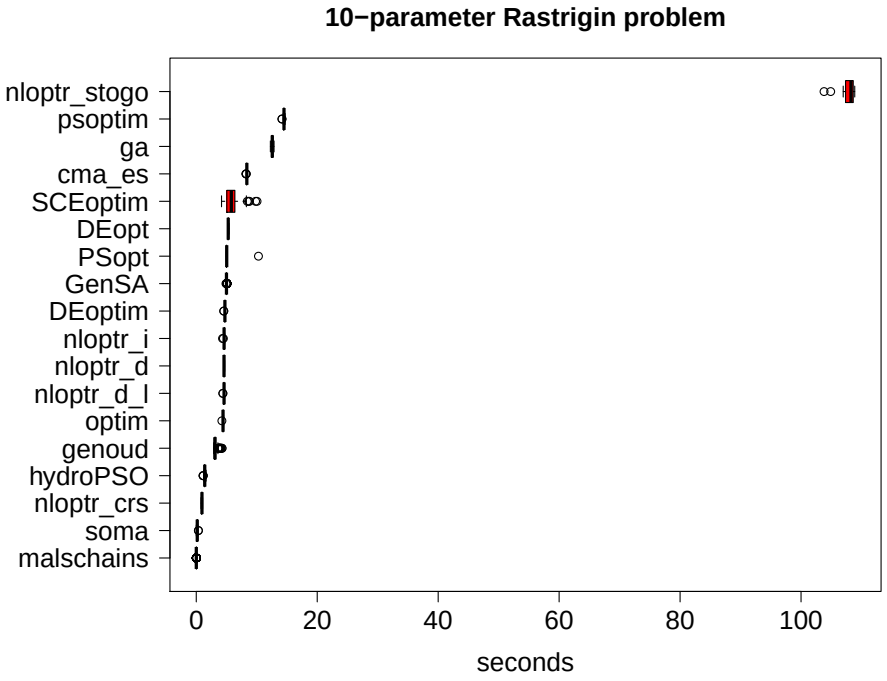
This appendix contains boxplots of time required to return a solution within a given budget of function evaluations as described in Section 4. The data described in these plots (along with scripts to generate and plot the results) is included as supplementary information to this paper.

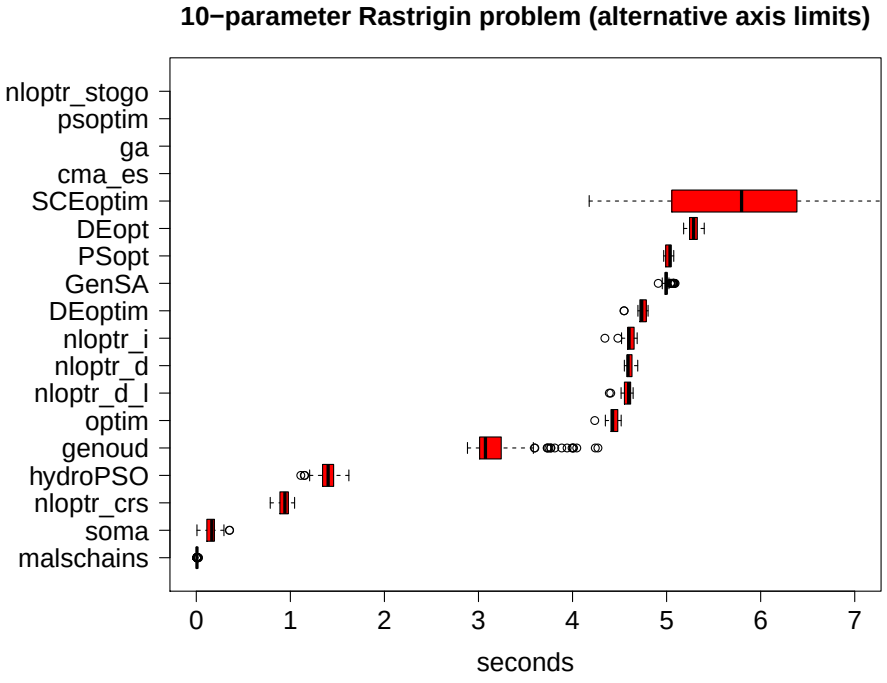


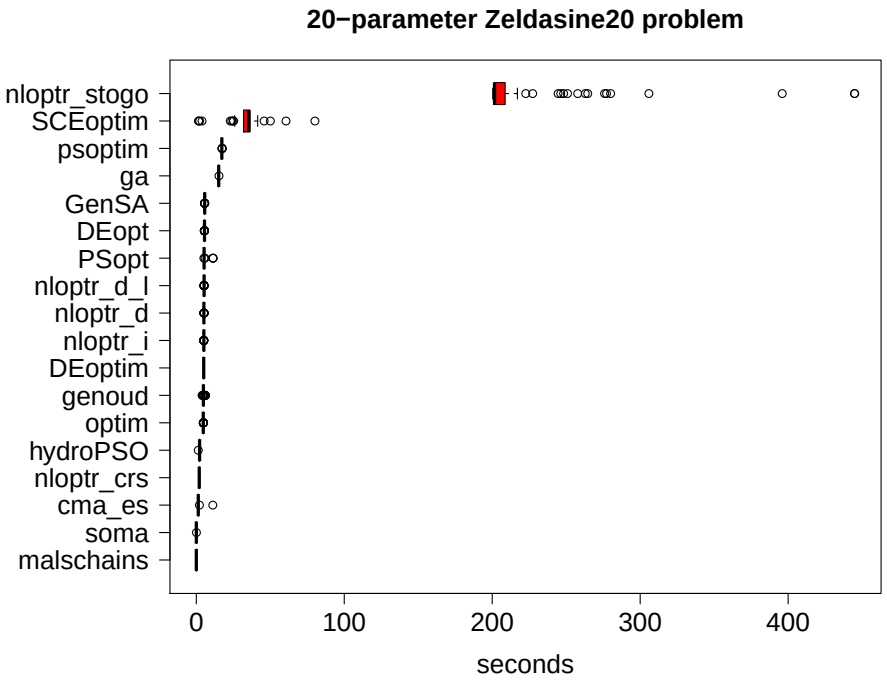


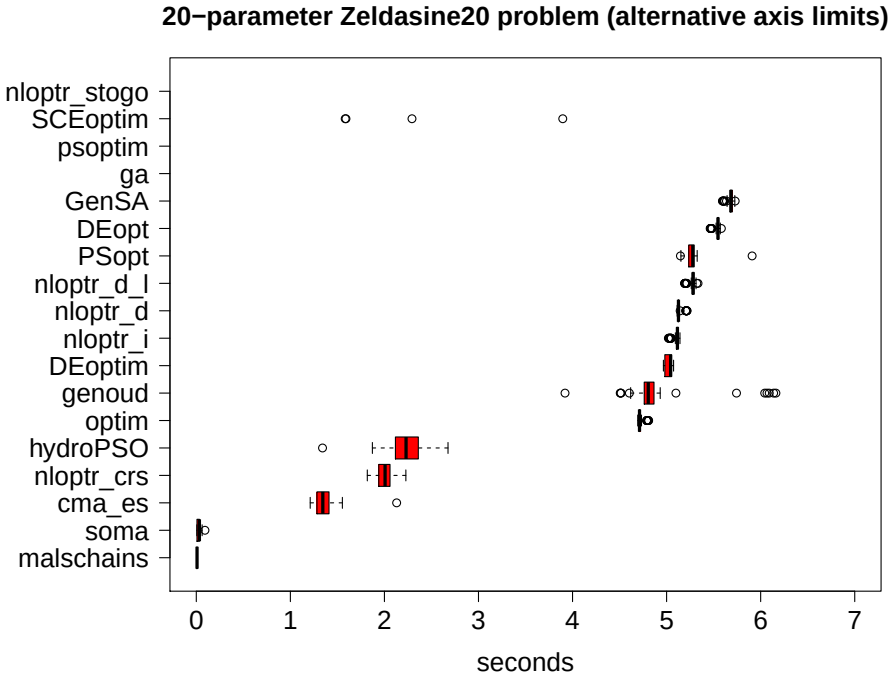

4−parameter Kowalik problem
nloptr_stogo
psoptim
ga
SCEoptim
PSopt
DEopt
nloptr_d_l
malschains
soma
DEoptim
GenSA
nloptr_d
optim
nloptr_i
cma_es
genoud
hydroPSO
nloptr_crs
0
20
40
60
80
100
seconds


4−parameter Kowalik problem (alternative axis limits)
nloptr_stogo
psoptim
ga
SCEoptim
PSopt
DEopt
nloptr_d_l
malschains
soma
DEoptim
GenSA
nloptr_d
optim
nloptr_i
cma_es
genoud
hydroPSO
nloptr_crs
0
1
2
3
4
5
6
7
seconds


10−parameter Rastrigin problem
nloptr_stogo
psoptim
ga
cma_es
SCEoptim
DEopt
PSopt
GenSA
DEoptim
nloptr_i
nloptr_d
nloptr_d_l
optim
genoud
hydroPSO
nloptr_crs
soma
malschains
0
20
40
60
80
100
seconds


10−parameter Rastrigin problem (alternative axis limits)
nloptr_stogo
psoptim
ga
cma_es
SCEoptim
DEopt
PSopt
GenSA
DEoptim
nloptr_i
nloptr_d
nloptr_d_l
optim
genoud
hydroPSO
nloptr_crs
soma
malschains
0
1
2
3
4
5
6
7
seconds


20−parameter Zeldasine20 problem
nloptr_stogo
SCEoptim
psoptim
ga
GenSA
DEopt
PSopt
nloptr_d_l
nloptr_d
nloptr_i
DEoptim
genoud
optim
hydroPSO
nloptr_crs
cma_es
soma
malschains
0
100
200
300
400
seconds


20−parameter Zeldasine20 problem (alternative axis limits)
nloptr_stogo
SCEoptim
psoptim
ga
GenSA
DEopt
PSopt
nloptr_d_l
nloptr_d
nloptr_i
DEoptim
genoud
optim
hydroPSO
nloptr_crs
cma_es
soma
malschains
0
1
2
3
4
5
6
7
seconds

Affiliation:

Katharine M. Mullen
Department of Statistics
University of California, Los Angeles
8125 Math Sciences Bldg.
Los Angeles, CA 90095-1554, United States of America
E-mail: katharine.mullen@stat.ucla.edu
URL: http://www.stat.ucla.edu/~katharine.mullen/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 60, Issue 6 *Submitted:* 2012-12-26
September 2014 *Accepted:* 2014-08-05