smoof: Single- and Multi-Objective Optimization Test Functions
by Jakob Bossek

Abstract Benchmarking algorithms for optimization problems usually is carried out by running the algorithms under consideration on a diverse set of benchmark or test functions. A vast variety of test functions was proposed by researchers and is being used for investigations in the literature. The smoof package implements a large set of test functions and test function generators for both the single- and multi-objective case in continuous optimization and provides functions to easily create own test functions. Moreover, the package offers some additional helper methods, which can be used in the context of optimization.

Introduction

The task of global optimization is to find the best solution \( x = (x_1, \ldots, x_n)^T \in X \) according to a set of objective functions \( F = \{f_1, \ldots, f_m\} \). If input vectors as well as output values of the objective functions are real-valued, i.e., \( f_i : \mathbb{R}^n \to \mathbb{R} \) the optimization problem is called continuous. Otherwise, i.e., if there is at least one non-continuous parameter, the problem is termed mixed. For \( m = 1 \), the optimization problem is termed single-objective and the goal is to minimize a single objective \( f \), i.e.,

\[
x^* = \arg \min_{x \in X} f(x).
\]

Clearly, talking about minimization problems is no restriction: we can maximize \( f \) by minimizing \( -f \). Based on the structure of the search space, there may be multiple or even infinitely many global optima, i.e., \( x^* \in X^* \subseteq X \). We are faced with a multi-objective optimization problem if there are at least two objective functions. In this case as a rule no global optimum exists since the objectives are usually conflicting and there is just a partial order on the search space; for sure \((1,4)^T \leq (3,7)^T \) makes sense, but \((1,4)^T \) and \((3,2)^T \) are not comparable. In the field of multi-objective optimization we are thus interested in a set \( PS = \{ x \in X | \exists \tilde{x} \in X : f(\tilde{x}) \preceq f(x) \} \subseteq X \) of optimal trade-off solutions termed the Pareto-optimal set, where \( \preceq \) defines the dominance relation. A point \( x \in X \) dominates another point \( \tilde{x} \in X \), i.e., \( x \preceq \tilde{x} \) if

\[
\forall i \in \{1, \ldots, m\} : f_i(x) \leq f_i(\tilde{x})
\]

and

\[
\exists i \in \{1, \ldots, m\} : f_i(x) < f_i(\tilde{x}).
\]

Hence, all trade-off solutions \( x^* \in PS \) are non-dominated. The image of the Pareto-set \( PF = f(PS) = (f_1(PS), \ldots, f_m(PS)) \) is the Pareto-front in the objective space. See Coello et al. (2006) for a thorough introduction to multi-objective optimization.

There exists a plethora of optimization algorithms for single-objective continuous optimization in R (see the CRAN Task View on Optimization and Mathematical Programming (Theussl and Borchers) for a growing list of available implementations and packages). Mullen (2014) gives a comprehensive review of continuous single-objective global optimization in R. In contrast there are just a few packages, e.g., emoa (Mersmann, 2012), mco (Mersmann, 2014), ecr (Bossek, 2017a), with methods suited to tackle continuous multi-objective optimization problems. These packages focus on evolutionary multi-objective algorithms (EMOA), which are very successful in approximating the Pareto-optimal set.

Benchmarking optimization algorithms

In order to investigate the performance of optimization algorithms or for comparing of different algorithmic optimization methods in both the single- and multi-objective case a commonly accepted approach is to test on a large set of artificial test or benchmark functions. Artificial test functions exhibit different characteristics that pose various difficulties for optimization algorithms, e.g., multimodal functions with more than one local optimum aim to test the algorithms’ ability to escape from local optima. Scalable functions can be used to access the performance of an algorithm while increasing the dimensionality of the decision space. In the context of multi-objective problems the geometry of the Pareto-optimal front (convex, concave, . . .) as well as the degree of multimodality are important...
characteristics for potential benchmarking problems. An overview of single-objective test function characteristics can be found in Jamil and Yang (2013). A thorough discussion of multi-objective problem characteristics is given by Huband et al. (2006). Kerschke and Dagefoerde (2015) recently published an R package with methods suited to quantify/estimate characteristics of unknown optimization functions at hand. Since the optimization community mainly focuses on purely continuous optimization, benchmarking test sets lack functions with discrete or mixed parameter spaces.

Related work

Several packages make benchmark functions available in R. The packages cec2005benchmark (Gonzalez-Fernandez and Soto, 2015) and cec2013 (Zambrano-Bigiarini and Gonzalez-Fernandez, 2015) are simple wrappers for the C implementations of the benchmark functions for the corresponding CEC 2005/2013 Special Session on Real-Parameter Optimization and thus very limited. The globalOptTests (Mullen, 2014) package interfaces 50 continuous single-objective functions. Finally the soobench (Mersmann and Bischl, 2012) package contains some single-objective benchmark functions and in addition several useful methods for visualization and logging.

Contribution

The package smoof (Bossek, 2017b) contains generators for a large and diverse set of both single-objective and multi-objective optimization test functions. Single-objective functions are taken from the comprehensive survey by Jamil and Yang (2013) and black-box optimization competitions (Hansen et al., 2009; Gonzalez-Fernandez and Soto, 2015). Moreover, a flexible function generator introduced by Wessing (2015) is interfaced. Multi-objective test functions are taken from Deb et al. (2002); Zitzler et al. (2000) and Zhang et al. (2009). In the current version – version 1.4 in the moment of writing – there are 99 continuous test function generators available (72 single-objective, 24 multi-objective, and 3 function family generators). Discontinuous functions (2) and functions with mixed parameters spaces (1) are underrepresented at the moment. This is due to the optimization community mainly focusing on continuous functions with numeric-only parameter spaces as stated above. However, we plan to extend this category in upcoming releases.

Both single- and multi-objective smoof functions share a common and extensible interface, which allows to easily add new test functions. Finally, the package contains additional helper methods which facilitate logging in the context of optimization.

Installation

The smoof package is available on CRAN, the Comprehensive R Archive Network, in version 1.4. To download, install and load the current release, just type the code below in your current R session.

```r
> install.packages("smoof")
> library(smoof)
```

If you are interested in toying around with new features take a look at the public repository at GitHub (https://github.com/jakobbossek/smoof). This is also the place to complain about problems and missing features / test functions; just drop some lines to the issue tracker.

Diving into the smoof package

In this section we first explain the internal structure of a test function in the smoof package. Later we take a look on how to create objective functions, the predefined function generators and visualization. Finally, we present additional helper methods which may facilitate optimization in R.

Anatomy of smoof functions

The functions makeSingleObjectiveFunction and makeMultiObjectiveFunction respectively can be used to create objective functions. Both functions return a regular R function with its characteristic properties appended in terms of attributes. The properties are listed and described in detail below.

- **name**  The function name. Mainly used for plots and console output.
- **id**  Optional short name. May be useful to index lists of functions.
- **description**  Optional description of the function. Default is the empty string.
fn  The actual implementation of the function. This must be a function of a single argument x.

has.simple.signature Logical value indicating whether the function fn expects a simple vector of values or a named list. This parameter defaults to TRUE and should be set to FALSE, if the function depends on a mixed parameter space, i.e., there are both numeric and factor parameters.

par.set The set of function parameters of fn. smoof makes use of the ParamHelpers (Bischl et al., 2016) package to define parameters.

noisy Is the function noisy? Default is FALSE.

minimize Logical value(s) indicating which objectives are to be minimized (TRUE) or maximized (FALSE) respectively. For single objective functions a single logical value is expected. For multi-objective test functions a logical vector with n.objectives components must be provided. Default is to minimize all objectives.

vectorized Does the function accept a matrix of parameter values? Default is FALSE.

constraint.fn Optional function which returns a logical vector indicating which non-box-constraints are violated.

tags A character vector of tags. A tag is a kind of label describing a property of the test function, e.g., multimodal or separable. Call the getAvailableTags function for a list of possible tags and see (Jamil and Yang, 2013) for a description of these. By default, there are no tags associated with the test function.

global.opt.params If the global optimum is known, it can be passed here as a vector, matrix, list or data.frame.

global.opt.value The function value of the global.opt.params argument.

n.objectives The number of objectives.

Since there exists no global optimum in multi-objective optimization, the arguments global.opt.params and global.opt.value are exclusive to the single-objective function generator. Moreover, tagging is possible for the single-objective case only until now. In contrast, the property n.objectives is set to 1 internally for single-objective functions and is thus no parameter of makeSingleObjectiveFunction.

Creating smoof functions

The best way to describe how to create an objective function in smoof is via example. Assume we want to add the two-dimensional Sphere function

\[ f : \mathbb{R}^2 \to \mathbb{R}, x \mapsto x_1^2 + x_2^2 \] with \( x_1, x_2 \in [-10, 10] \)

to our set of test functions. The unique global optimum is located at \( x^* = (0, 0)^T \) with a function value of \( f(x^*) = 0 \). The code below is sufficient to create the Sphere function with smoof.

```r
> fn <- makeSingleObjectiveFunction(
>   name = "2D-Sphere",
>   fn = function(x) x[1]^2 + x[2]^2,
>   par.set = makeNumericParamSet(
>     len = 2L, id = "x",
>     lower = c(-10, -10), upper = c(10, 10),
>     vector = TRUE
>   ),
>   tags = "unimodal",
>   global.opt.param = c(0, 0),
>   global.opt.value = 0
> )
> print(fn)
Single-objective function
Name: 2D-Sphere
Description: no description
Tags:
Noisy: FALSE
Minimize: TRUE
Constraints: TRUE
Number of parameters: 2

<table>
<thead>
<tr>
<th>Type</th>
<th>len</th>
<th>Def</th>
<th>Constr</th>
<th>Req</th>
<th>Tunable</th>
<th>Trafo</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>numeric</td>
<td>vector</td>
<td>-</td>
<td>-10,-10 to 10,10</td>
<td>TRUE</td>
<td>-</td>
</tr>
</tbody>
</table>
```

The R Journal Vol. 9/1, June 2017 ISSN 2073-4859
Global optimum objective value of 0.0000 at
x1 x2
1 0 0

Here we pass the mandatory arguments name, the actual function definition fn and a parameter set par.set. We state, that the function expects a single numeric vector parameter of length two where each component should satisfy the box constraints \((x_1, x_2 \in [-10, 10])\). Moreover we let the function know its own optimal parameters and the corresponding value via the optional arguments global.opt.param and global.opt.value. The remaining arguments fall back to their default values described above.

As another example we construct a mixed parameter space function with one numeric and one discrete parameter, where the latter can take the three values \(a, b, c\) respectively. The function is basically a shifted single-objective Sphere function, where the shift in the objective space depends on the discrete value. Since the function is not purely continuous, we need to pass the calling entity a named list to the function and thus has.simple.signature is set to FALSE.

```
> fn2 <- makeSingleObjectiveFunction(
>   name = "Shifted-Sphere",
>   fn = function(x) {
>     shift = which(x$disc == letters[1:3]) * 2
>     return(x$num^2 + shift)
>   },
>   par.set = makeParamSet(
>     makeNumericParam("num", lower = -5, upper = 5),
>     makeDiscreteParam("disc", values = letters[1:3])
>   ),
>   has.simple.signature = FALSE
> )
> print(fn2)
Single-objective function
Name: Shifted-Sphere
Description: no description
Tags:
Noisy: FALSE
Minimize: TRUE
Constraints: TRUE
Number of parameters: 2
<table>
<thead>
<tr>
<th>Type</th>
<th>len</th>
<th>Def</th>
<th>Constr</th>
<th>Req</th>
<th>Tunable</th>
<th>Trafo</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>numeric</td>
<td>-</td>
<td>-5 to 5</td>
<td>TRUE</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>disc</td>
<td>discrete</td>
<td>-</td>
<td>a,b,c</td>
<td>TRUE</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

> fn2(list(num = 3, disc = "c"))
[1] 15
```

Visualization

There are multiple methods for the visualization of 1D or 2D smoof functions. The generic plot method draws a contour plot or level plot or a combination of both. The following code produces the graphics depicted in Figure 1 (left).

```
> plot(fn, render.contours = TRUE, render.levels = TRUE)
```

Here the argument render.levels achieves the heatmap effect, whereas render.contours activates the contour lines. Moreover, numeric 2D functions can be visualized as a 3D graphics by means of the plot3D function (see Fig. 1 (right)).

```
> plot3D(fn, contour = TRUE)
```

If you prefer the visually appealing graphics of ggplot2 (Wickham, 2009) you can make use of autoplot, which returns a ggplot object. The returned ggplot object can be easily modified with additional geometric objects, statistical transformations and layers. For instance, let us visualize the mixed parameter function fn2 which was introduced in the previous subsection. Here we activate ggplot2 facetting via use.facets = TRUE, flip the default facet direction and adapt the limits of the objective axis by hand. Figure 2 shows the resulting plot.
library(ggplot2)
pl <- autoplot(fn2, use.facets = TRUE) # basic call
pl + ylim(c(0, 35)) + facet_grid(. ~ disc) # (one column per discrete value)

In particular, due to the possibility to subsequently modify the ggplot objects returned by the autoplot function it can be used effectively in other packages to, e. g., visualize an optimization process. For instance the ecr package makes extensive use of smoof functions and the ggplot2 plots.

**Getter methods**

Accessing the describing attributes of a smoof function is essential and can be simply realized by attr("attrName", fn) or alternatively via a set of helper functions. The latter approach is highly recommended. By way of example the following listing shows just a few of the available helpers.

```r
> getGlobalOptimum(fn)$param
x1 x2
1 0 0
> getGlobalOptimum(fn)$value
[1] 0
> getGlobalOptimum(fn)$is.minimum
[1] TRUE
> getNumberOfParameters(fn)
[1] 2
```
Extending `smoof` with custom objective functions is nice to have, but the main benefit in using this package is the large set of preimplemented functions typically used in the optimization literature. At the moment of writing there are in total 72 single-objective functions and 24 multi-objective function generators available. Moreover, there are interfaces to some more specialized benchmark sets and problem generators which will be mentioned in the next section.

**Generators for single-objective test functions**

To apply some optimization routines to say the Sphere function you do not need to define it by hand. Instead you can just call the corresponding `generator` function, which has the form `makeFUNFunction` where `FUN` may be replaced with one of the function names. Hence, the Sphere function can be generated by calling `makeSphereFunction(dimensions = 2L)`, where the integer `dimensions` argument defines the dimension of the search space for scalable objective functions, i.e., functions which are defined for arbitrary parameter space dimensions $n \geq 2$. All 72 single-objective functions with their associated tags are listed in Table 1. The tags are based on the test function survey in (Jamil and Yang, 2013). Six functions with very different landscapes are visualized in Figure 3.

![Figure 3: Two-dimensional test functions Matyas (top left), Rastrigin (top middle), Himmelblau (top right), Zettl (bottom left), Three Hump Camel (bottom middle) and Generalized Drop Wave (bottom right).](image)

Beside these functions there exist two additional single-objective generators, which interface special test function sets or function generators.

**BBOB** The 24 Black-Box Optimization Benchmark (BBOB) 2009 (Hansen et al., 2009) functions can be created with the `makeBBOBFunction(fid, iid, dimension)` generator, where `fid` $\in \{1, \ldots, 24\}$ is the function identifier, `iid` is the instance identifier and `dimension` the familiar argument for specifying the parameter space dimension.

**MPM2** The problem generator `multiple peaks model 2` (Wessing, 2015) is accessible via the function `makeMPM2Function`. This problem generator produces multimodal problem instances by combining several randomly distributed peaks (Wessing, 2015). The number of peaks can be set via the `n.peaks` argument. Further arguments are the problem dimension, an initial seed for the random numbers generator and the topology, which accepts the values ‘random’ or ‘funnel’ respectively. For details see the technical report of the multiple peaks model 2 Wessing (2015).
<table>
<thead>
<tr>
<th>Function</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ackley</td>
<td>continuous, multimodal, differentiable, non-separable, scalable</td>
</tr>
<tr>
<td>Adjamia</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Alpine N. 1</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Alpine N. 2</td>
<td>continuous, differentiable, separable, scalable, multimodal</td>
</tr>
<tr>
<td>Aluffi-Pentini</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Bartels Conn</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Beale</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Bent-Cigar</td>
<td>continuous, differentiable, non-separable, scalable, unimodal</td>
</tr>
<tr>
<td>Bird</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>BileShpere</td>
<td>multi-objective</td>
</tr>
<tr>
<td>Bohachevsky N. 1</td>
<td>continuous, differentiable, separable, scalable, multimodal</td>
</tr>
<tr>
<td>Booth</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>BraninRCOS</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Brent</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Brown</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Bukin N. 2</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Bukin N. 4</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Bukin N. 6</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Carrom Table</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Chichinadze</td>
<td>continuous, differentiable, non-separable, scalable, multimodal</td>
</tr>
<tr>
<td>Chung Reynolds</td>
<td>unimodal, continuous, differentiable, scalable</td>
</tr>
<tr>
<td>Complex</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Cosine Mixture</td>
<td>discontinuous, non-differentiable, scalable, multimodal</td>
</tr>
<tr>
<td>Cross-In-Tray</td>
<td>continuous, non-separable, scalable, multimodal</td>
</tr>
<tr>
<td>Cube</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Deckkers-Aarts</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Deflected Corrugated Spring</td>
<td>continuous, differentiable, non-separable, multimodal</td>
</tr>
<tr>
<td>Dixon-Price</td>
<td>continuous, differentiable, non-separable, scalable, unimodal</td>
</tr>
<tr>
<td>Double-Sum</td>
<td>continuous, differentiable, non-separable, scrutable, multimodal</td>
</tr>
<tr>
<td>Eason</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Egg Crate</td>
<td>continuous, differentiable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Egg Holder</td>
<td>continuous, differentiable, non-scalable, multimodal</td>
</tr>
<tr>
<td>El-Attar-Vidyasagar-Dutta</td>
<td>continuous, differentiable, non-separable, unimodal</td>
</tr>
<tr>
<td>Englva</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Exponential</td>
<td>continuous, differentiable, non-separable, scalable</td>
</tr>
<tr>
<td>Freudenstein Roth</td>
<td>continuous, differentiable, non-separable, none scalable, multimodal</td>
</tr>
<tr>
<td>Generalized Drop-Wave</td>
<td>multimodal, non-separable, continuous, differentiable, scalable</td>
</tr>
<tr>
<td>Giunta</td>
<td>continuous, differentiable, separable, multimodal</td>
</tr>
<tr>
<td>Goldstein-Price</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Griewank</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Hansen</td>
<td>continuous, differentiable, non-separable, scalable, multimodal</td>
</tr>
<tr>
<td>Himmelblau</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Holder Table N. 1</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Holder Table N. 2</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Hosaki</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Hyper-Ellipsoid</td>
<td>unimodal, convex, continuous, scalable</td>
</tr>
<tr>
<td>Jennrich-Sampson</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Judge</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Keane</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Kearfott</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Leon</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Mathias</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>McCormick</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Michalewicz</td>
<td>continuous, multimodal, scalable</td>
</tr>
<tr>
<td>Periodic</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Double-Sum</td>
<td>continuous, differentiable, separable, scalable, unimodal</td>
</tr>
<tr>
<td>Price N. 1</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Price N. 2</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Price N. 4</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>multimodal, continuous, separable, scalable</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Schaffer N. 2</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
<tr>
<td>Schaffer N. 4</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Schwefel</td>
<td>continuous, multimodal, scalable</td>
</tr>
<tr>
<td>Shubert</td>
<td>continuous, differentiable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Six-Hump Camel Back</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Sphere</td>
<td>unimodal, separable, convex, continuous, differentiable, scalable</td>
</tr>
<tr>
<td>Strybinski-Tang</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Sum of Different Squares</td>
<td>unimodal, continuous, scalable</td>
</tr>
<tr>
<td>Swiler2014</td>
<td>discontinuous, mixed, multimodal</td>
</tr>
<tr>
<td>Three-Hump Camel</td>
<td>continuous, differentiable, non-separable, non-scalable, multimodal</td>
</tr>
<tr>
<td>Trećkuni</td>
<td>continuous, differentiable, separable, scalable, unimodal</td>
</tr>
<tr>
<td>Zettl</td>
<td>continuous, differentiable, non-separable, non-scalable, unimodal</td>
</tr>
</tbody>
</table>

Table 1: All single objective functions currently available in smoof with their corresponding tags.
Generators for multi-objective test functions

Evolutionary algorithms play a crucial role in solving multi-objective optimization tasks. The relative performance of multi-objective evolutionary algorithms (MOEAs) is, as in the single-objective case, mainly studied experimentally by systematic comparison of performance indicators on test instances. In the past decades several test sets for multi-objective optimization were proposed mainly by the evolutionary computation community. The smoof package offers generators for the DTLZ function family by Deb et al. (Deb et al., 2002), the ZDT function family by Zitzler et al. (Zitzler et al., 2000) and the multi-objective optimization test instances UF1, . . . , UF10 of the CEC 2009 special session and competition (Zhang et al., 2009).

The DTLZ generators are named makeDTLZXFunction with X = 1, . . ., 7. All DTLZ generators need the search space dimension n (argument dimensions) and the objective space dimension p (argument n.objectives) with n ≥ p to be passed. DTLZ4 may be passed an additional argument alpha with default value 100, which is recommended by Deb et al. (2002). The following lines of code generate the DTLZ2 function and visualize its Pareto-front by running the NSGA-II EMOA implemented in the mco package with a population size of 100 for 100 generations (see Figure 4).

```r
> fn = makeDTLZ2Function(dimensions = 2L, n.objectives = 2L)
> visualizeParetoOptimalFront(fn, show.only.front = TRUE)
```

ZDT and UF functions can be generated in a similar manner by utilizing makeZDTXFunction with X = 1, . . ., 5 or makeUFFunction.

![Figure 4: Visualization of the approximated Pareto-front for the DTLZ2 function with two-dimensional search and objective space.](image)

Optimization helpers

In this section we present some additional helper methods which are available in smoof.

Filtering and building of test sets

In a benchmark study we most often need not just a single test function, but a set of test functions with certain properties. Say we want to benchmark an algorithm on all multimodal smoof functions. Instead of scouring the smoof documentation for suitable test functions we can make use of the filterFunctionsByTags helper function. This function has only a single mandatory argument, namely a character vector of tags. Hence, to get an overview of all multimodal functions we can write the following:

```r
> fn.names <- filterFunctionsByTags(tags = "multimodal")
> head(fn.names)
[1] "Ackley"  "Adjiman"  "Alpine N. 1"  "Alpine N. 2"  "Bartels Conn"
[6] "Bird"
```
The above shows there are 46 multimodal functions. The next step is to generate the actual smoof functions. We could do this by hand, but this would be tedious work. Instead we utilize the makeFunctionsByName helper function which comes in useful in combination with filtering. It can be passed a vector of generator names (like the ones returned by filterFunctionsByTags) and additional arguments which are passed down to the generators itself. E. g., to initialize all two-dimensional multimodal functions, we can apply the following function call.

```r
> fns <- makeFunctionsByName(fn.names, dimensions = 2)
> all(sapply(fns, isSmoofFunction))
[1] TRUE
```

The `smoof` package ships with some handy wrappers. These are functions which expect a `smoof` function and possibly some further arguments and return a wrapped `smoof` function, which behaves as the original and does some secret logging additionally. We can wrap a `smoof` function within a counting wrapper (function `addCountingWrapper`) to count the number of function evaluations. This is of particular interest, if we compare stochastic optimization algorithms and the implementations under consideration do not return the number of function evaluations carried out during the optimization process. Moreover, we might want to log each function value along the optimization process. This can be achieved by means of a logging wrapper. The corresponding function is `addLoggingWrapper`. By default it logs just the test function values (argument `logg.y` is `TRUE` by default). Optionally the logging of the decision space might be activated by setting the `logg.x` argument to `TRUE`. The following listing illustrates both wrappers by examplary optimizing the Branin RCOS function with the Nelder-Mead Simplex algorithm.

```r
> set.seed(123)
> fn <- makeBraninFunction()
> fn <- addCountingWrapper(fn)
> fn <- addLoggingWrapper(fn, logg.x = TRUE, logg.y = TRUE)
> par.set <- getParamSet(fn)
> lower <- getLower(par.set); upper = getUpper(par.set)
> res <- optim(c(0, 0), fn = fn, method = "Nelder-Mead")
> res$counts[1L] == getNumberOfEvaluations(fn)
[1] TRUE
> head(getLoggedValues(fn, compact = TRUE))
  x1  x2  y1
1 1 0.00 0.00 55.60
2 0.10 0.00 53.68
3 0.00 0.10 54.41
4 0.10 0.10 52.53
5 0.15 0.15 51.01
6 0.25 0.05 50.22
```
Conclusion and future work

Benchmarking optimization algorithms on a set of artificial test functions with well-known characteristics is an established means of evaluating performance in the optimization community. This article introduces the R package smoof, which contains a large collection of continuous test functions for the single-objective as well as the multi-objective case. Besides a set of helper functions is introduced which allows users to log in detail the progress of the optimization algorithm(s) studied. Future work will lay focus on implementing more continuous test functions, introducing test functions with mixed parameter spaces and provide reference Pareto-sets and Pareto-Fronts for the multi-objective functions. Furthermore the reduction of evaluation time by rewriting existing functions in C(++) is planned.

Jakob Bossek  
PhD Student  
Department of Information Systems  
University of Münster  
Germany  
bossek@wi.uni-muenster.de

Bibliography


