Mathematical Programming Computation manuscript No.
(will be inserted by the editor)

RBFOpt: an open-source library for black-box optimization with costly function evaluations

Alberto Costa · Giacomo Nannicini

Received: date / Accepted: date

Abstract We consider the problem of optimizing an unknown function given as an oracle over a mixed-integer set. We assume that the oracle is expensive to evaluate, so that estimating partial derivatives by finite differences is impractical. In the literature, this is typically called a *black-box* optimization problem with costly evaluation. Our approach is based on the Radial Basis Function method originally proposed by Gutmann (2001), which builds and iteratively refines a surrogate model of the unknown objective function. The two main methodological contributions of this paper are an approach to exploit a noisy but less expensive oracle to accelerate convergence to the optimum of the exact oracle, and the introduction of an automatic model selection phase during the optimization process. Numerical experiments show that these contributions significantly improve the performance of the algorithm on a test set of continuous and mixed-integer nonlinear unconstrained problems taken from the literature. Our implementation is open-source and free for non-commercial academic use.

Keywords Black-box optimization · Derivative free optimization · Global optimization · Radial basis function · Open-source software · Mixed-integer nonlinear programming

A. Costa
Singapore University of Technology and Design
E-mail: costa@sutd.edu.sg

G. Nannicini
Singapore University of Technology and Design
E-mail: nannicini@sutd.edu.sg

1 Introduction

In this paper, we address a problem cast in the following form:

$$\left.\begin{aligned} \min\ & f(x) \\ & x \in [x^L, x^U] \\ & x \in \mathbb{Z}^q \times \mathbb{R}^{n-q}, \end{aligned}\right\} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $x^L, x^U \in \mathbb{R}^n$ are vectors of lower and upper bounds on the decision variables, and $q \leq n$. We assume that f is continuous with respect to all variables, even though some of the variables are restricted to take only on integer values. Furthermore, we assume that the analytical expression for f is unknown and function values are only available through an oracle that is expensive to evaluate, e.g. a time-consuming simulation. In the literature, this is typically called a *black-box* optimization problem with costly evaluation.

This problem class finds many applications. Our work originated from a project in architectural design where we faced the following problem (see also [9]). During the design phase, a building can be described by a parametric model and the parameters are the decision variables, that can be continuous or discrete. Lighting and heating simulation software can be used to study energy profiles of buildings, simulating sun exposure over a prescribed period of time. This information can be used to determine a performance measure, i.e. an objective function, but the analytical expression is not available due to the complexity of the simulations. The goal is to optimize this function to find a good parameterization of the parametric model of the building. However, each run of the simulation software usually takes considerable time: up to several hours. Thus, we want to optimize the objective function within a small budget of function evaluations to keep computing times under control. Other applications of this approach can be found in engineering disciplines where the simulation relies on the solution of a system of PDEs, for example in the context of performance optimization for complex physical devices such as engines, see e.g. [2, 18].

There is a very large stream of literature on black-box optimization in general, also called *derivative-free optimization* (sometimes generating confusion). Numerous methods have been proposed, and the choice of a particular method should depend on the number of function evaluations allowed, the dimension of the problem, and its structural properties. Heuristic approaches are very common thanks to their simplicity, for example scatter search, simulated annealing and evolutionary algorithms; see [13, 12] for an overview. However, these methods are not specifically tailored for the setting of this paper and often require a large number of function evaluations, as has been noted by [15, 29] among others. In general, methods that do not take advantage of the inherent smoothness of the objective function may take a long time to converge [6]. Unfortunately, because of the assumption of expensive function evaluation, estimating partial derivatives by finite differences is impractical and often has prohibitive computational cost. A commonly used approach in this context is that of building an approximation model of f, also called *response surface*

or *surrogate model*. Examples of this approach are the Radial Basis Function (RBF) method of [15] (see also [27]), the stochastic RBF method [29], and the kriging-based Efficient Global Optimization method (EGO) of [23]. The surrogate model constructed by these methods is a global model that uses all available information on f, as opposed to methods that only build a local model such as trust-region based methods [6,7]. Other methods for black-box optimization rely on direct search, i.e., they do not build a surrogate model of the objective function. An overview of direct search methods can be found in [24], and a comprehensive treatment is given in [7]. We refer the reader to [31] and the references therein for a very recent survey on black-box methods and an extensive computational evaluation.

In this paper we focus on problems that are nonconvex, relatively small-dimensional, and for which only a small number of function evaluations is allowed. For this type of problems, algorithms based on a surrogate model are typically considered among the most effective. In particular, empirical evidence [20] suggests that the RBF method is more effective on engineering problems, despite the appealing theoretical properties of other methodologies such as EGO. Besides this empirical evidence, there are three additional reasons that make a surrogate model method more appealing than direct search in our context. The first reason is that looking for alternative optima, or at least a set of good solutions, is easier if we can rely on a surrogate model. The second reason is that it is intuitively easier to "warm-start" such a method in a context where each function evaluation (i.e. simulation) produces some data that allows for fast recomputation of different but related objective functions, because the model of these new objective functions can be quickly built. The third reason is that the surrogate model can sometimes be used to allow the fast (potentially inaccurate) exploration of the objective function around the optimum: we study this possibility in Section 5.7. These properties are important for our motivating application from a practical perspective, which explains our choice.

In this paper, we review the RBF method and present some extensions aimed at improving its practical performance. Our most significant contributions to the class of RBF methods are a fast procedure for automatic model selection, and an approach to accelerate convergence in case we have access to an additional oracle that returns noisy function values (i.e. affected by error) but is less expensive to evaluate than the exact oracle for f. These contributions could be adapted to other surrogate model based methods and should therefore be considered of general interest, rather than specific for the RBF method. Our implementation of the method is an open-source library called RBFOpt, and we show that it is competitive with state-of-the-art commercial software on a set of test problems taken from the literature. In particular, the two main contributions of these paper significantly decrease the number of function evaluations for global convergence on our test set. Furthermore, we show that on our test set, the proposed methodology for automatic model selection yields a measure of model quality that is helpful in deciding whether or

$\phi(r)$		$d_{\min}$
r	(linear)	0
r^3	(cubic)	1
$r^2 \log r$	(thin plate spline)	1
$\sqrt{r^2+\gamma^2}$	(multiquadric)	0
$\frac{1}{\sqrt{r^2+\gamma^2}}$	(inverse multiquadric)	-1
$e^{-\gamma r^2}$	(Gaussian)	-1

Table 1 Common RBF functions.

not the surrogate model is accurate around the optimum, in order to perform sensitivity analysis without requiring additional oracle evaluations.

The rest of the paper is organized as follows. In Section 2 we review the RBF method. Sections 3 and 4 present some extensions to improve the efficacy of the method. Section 5 describes our implementation and reports the results of a computational evaluation on a set of test problems taken from the literature. Section 6 concludes the paper.

2 The Radial Basis Function algorithm for black-box optimization

The introduction of radial basis functions to black-box optimization dates back to [26], but the first fully-developed algorithm exploiting RBFs is due to [15]. We now review the main components of the RBF method as introduced by the latter paper. Its main idea is to use RBF interpolation to build a surrogate model, and define a measure of "bumpiness". Intuitively, given a target objective function value, the bumpiness at a point measures the likelihood that the target function value occurs there, based on the interpolation points. The implicit assumption is that the unknown function f does not oscillate too much, therefore we aim for a model that can explain the data and minimizes the bumpiness. [15] also proposes a strategy to select target function values. The method iteratively chooses a target value, finds the point in the search space that minimizes bumpiness for that target value, and evaluates f at such point.

For a formal description, we must define the surrogate model used in the RBF method. Let $\Omega := \{x \in [x^L, x^U]\} \subset \mathbb{R}^n$, $\Omega_I := \{x \in [x^L, x^U] : x_i \in \mathbb{Z}\ \forall i \in \{1, \ldots, q\}\} \subseteq \Omega$. Given k distinct points $x_1, \ldots, x_k \in \Omega$, the radial basis function interpolant s_k is defined as:

$$s_k(x) := \sum_{i=1}^{k} \lambda_i \phi(\|x - x_i\|) + p(x), \tag{2}$$

where $\phi : \mathbb{R}_+ \to \mathbb{R}$, $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$ and p is a polynomial of degree d. The minimum degree $d_{\min}$ to guarantee existence of the interpolant depends on the form of the functions ϕ. Table 1 gives the most commonly used radial basis functions $\phi(r)$ and the corresponding value of $d_{\min}$. The parameter $\gamma > 0$ can

be used to change the shape of these functions (but it is usually set to 1).

If $\phi(r)$ is cubic or thin plate spline, $d_{\min} = 1$ and we obtain an interpolant of the form:

$$s_k(x) := \sum_{i=1}^{k} \lambda_i \phi(\|x - x_i\|) + h^T \begin{pmatrix} x \\ 1 \end{pmatrix}, \tag{3}$$

where $h \in \mathbb{R}^{n+1}$. The values of λ_i, h can be determined by solving the following linear system:

$$\begin{pmatrix} \Phi & P \\ P^T & 0_{(n+1)\times(n+1)} \end{pmatrix} \begin{pmatrix} \lambda \\ h \end{pmatrix} = \begin{pmatrix} F \\ 0_{n+1} \end{pmatrix}, \tag{4}$$

with:

$$\Phi = (\phi(\|x_i - x_j\|))_{i,j=1,\dots,k}, \quad P = \begin{pmatrix} x_1^T & 1 \\ \vdots & \vdots \\ x_k^T & 1 \end{pmatrix}, \quad \lambda = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{pmatrix}, \quad F = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_k) \end{pmatrix}.$$

If $\operatorname{rank}(P) = n + 1$, the system (4) is nonsingular [15].

If $\phi(r)$ is linear or multiquadric, $d_{\min} = 0$ and P is the all-one column vector of dimension k, whereas in the inverse multiquadric and Gaussian case, $d_{\min} = -1$ and P is removed from system (4). The dimensions of the zero matrix and vector in (4) are adjusted accordingly.

Next, we define the measure of bumpiness σ. The motivation for the use of bumpiness comes from the theory of natural cubic spline interpolation in dimension one (RBFs can be seen as its extension to multivariate functions). It is well known that the natural cubic spline interpolant whose parameters are found by solving a system as (4) (where $n = 1$ and $\phi(r) = r^3$) is the function which minimizes $\int_{\mathbb{R}} [g''(x)]^2 \, \mathrm{d}x$ among all the functions $g : \mathbb{R} \to \mathbb{R}$ such that $\forall i \in \{1, \dots, k\}$ $g(x_i) = f(x_i)$. Hence, $\int_{\mathbb{R}} [g''(x)]^2 \mathrm{d}x$ is a good measure of bumpiness. It is shown in [15] that in the case of RBF interpolants in dimension n, this can be generalized to:

$$\begin{aligned} \sigma(s_k) &= (-1)^{d_{\min}+1} \sum_{i=1}^{k} \lambda_i s_k(x_i) = (-1)^{d_{\min}+1} \sum_{i=1}^{k} \sum_{j=1}^{k} \lambda_i \lambda_j \phi(\|x_i - x_j\|) = \\ &= (-1)^{d_{\min}+1} \lambda^T \Phi \lambda. \end{aligned} \tag{5}$$

Let us assume that after k function values $f(x_1), \dots, f(x_k)$ are evaluated, we want to find a point in Ω_I where it is likely that the unknown function attains a target value $f_k^* \in \mathbb{R}$ (strategies for selecting f_k^* will be discussed in the following). Let s_y be the RBF interpolant subject to the conditions:

$$s_y(x_i) = f(x_i) \qquad \forall i \in \{1, \dots, k\} \tag{6}$$

$$s_y(y) = f_k^*. \tag{7}$$

The assumption of the RBF method is that a likely location for the point y with function value f_k^* is the one that minimizes $\sigma(s_y)$. That is, we look for

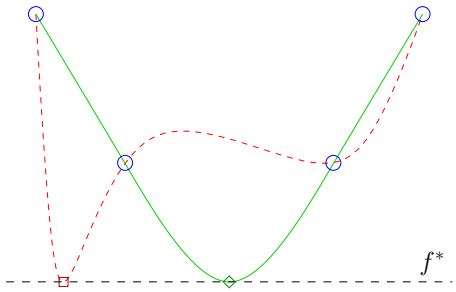

Fig. 1 An example with four interpolation points (circles) and a value of f_k^* represented by the horizontal dashed line. The RBF method assumes that it is more likely that a point with value f_k^* is located at the diamond rather than the square, because the resulting interpolant is less bumpy.

the interpolant that is "the least bumpy". A sketch of this idea is given in Figure 1.

Instead of computing the minimum of $\sigma(s_y)$ to find the least bumpy interpolant, we define an equivalent optimization problem that is easier to solve. Let ℓ_k be the RBF interpolant to the points $(x_i, 0)$, $\forall i \in \{1, \ldots, k\}$ and $(y, 1)$. A solution to (6)-(7) can be rewritten as:

$$s_y(x) = s_k(x) + [f_k^* - s_k(y)]\ell_k(x),\ x \in \mathbb{R}^n,$$

which clearly interpolates at the desired points by definition of ℓ_k. Let $\mu_k(y)$ be the coefficient corresponding to y of ℓ_k. $\mu_k(y)$ can be computed by extending the linear system (4), which becomes [15]:

$$\begin{pmatrix} \Phi & u(y) & P \\ u(y)^T & \phi(0) & \pi(y)^T \\ P^T & \pi(y) & 0 \end{pmatrix} \begin{pmatrix} \alpha(y) \\ \mu_k(y) \\ b(y) \end{pmatrix} = \begin{pmatrix} 0^k \\ 1 \\ 0^{n+1} \end{pmatrix}, \tag{8}$$

where $u(y) = (\phi(\|y - x_1\|), \ldots, \phi(\|y - x_k\|))^T$ and $\pi(y)$ is $\begin{pmatrix} y^T & 1 \end{pmatrix}$ when $d_{\min} = 1$, 1 when $d_{\min} = 0$ and it is not used when $d_{\min} = -1$. With algebraic manipulations (see [15,27]) we can obtain from the system (8) the following expression for $\mu_k(y)$:

$$\mu_k(y) = \frac{1}{\phi(0) - (u(y)\ \pi(y))^T \begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix}^{-1} (u(y)\ \pi(y))}. \tag{9}$$

A way of storing the factorization of Φ to speed up the computation of $\mu_k(y)$ is described in [3].

It can be shown [15] that computing the minimum of $\sigma(s_y)$ over $y \in \mathbb{R}^n$ is equivalent to minimizing the utility function:

$$g_k(y) = (-1)^{d_{\min}+1}\mu_k(y)[s_k(y) - f_k^*]^2, \qquad y \in \Omega \setminus \{x_1, \ldots, x_k\}.$$

Unfortunately g_k and μ_k are not defined at $x_1, \ldots, x_k$, and $\lim_{x \to x_i} \mu_k(x) = \infty$, $\forall i \in \{1, \ldots, k\}$. To avoid numerical troubles, [15] suggests maximizing the following function:

$$h_k(x) = \begin{cases} \frac{1}{g_k(x)} & \text{if } x \notin \{x_1, \ldots, x_k\} \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

which is differentiable everywhere on Ω.

We now have all the necessary ingredients to describe the RBF algorithm, first introduced in [15]. We remark that $q = 0$ in the framework of [15], i.e. there are no integer variables. An extension to $q > 0$ is given in [20], and our exposition below follows the latter paper. The original algorithm can be recovered by substituting Ω for Ω_I.

- **Initial step:** Choose linearly independent $x_1, \ldots, x_n \in \Omega_I$ using an initialization strategy. Set $k \leftarrow n$. Compute the RBF s_k that minimizes $\sigma(s_k)$ subject to the interpolation conditions:

$$s_k(x_i) = f(x_i) \qquad \forall i \in \{1, \ldots, k\}.$$

- **Iteration step:** Repeat the following steps.
  - Choose a target value $f_k^* \in [-\infty, \min_{x \in \Omega_I} s_k(x)]$ (the choice $f_k^* = \min_{x \in \Omega_I} s_k(x)$ is admissible only if $f_k^* \neq f(x_i)$ $\forall i \in \{1, \ldots, k\}$).
  - Compute

$$x_{k+1} = \arg\max_{x \in \Omega_I} h_k(x), \tag{11}$$

    where $h(x)$ is defined as in (10).
  - Evaluate f at x_{k+1} and compute the RBF interpolant s_{k+1} that minimizes $\sigma(s_{k+1})$ subject to $s_{k+1}(x_i) = f(x_i)$ $\forall i \in \{1, \ldots, k+1\}$.
  - If we exceed a prescribed number of function evaluations, stop. Otherwise, set $k \leftarrow k + 1$.

The pseudo-code of the RBF method is given in Algorithm 1.

We still need to specify a strategy for choosing sample points in the Initial step and the target value f_k^* at each Iteration step. These will be the subject of the next two sections. Afterwards, we discuss a number of modifications to the basic algorithms that have been proposed in the literature and found to be beneficial in practice.

2.1 Choice of the initial sample points

A natural choice for the initial sample points is to pick the 2^n corner points of the box Ω, but this is reasonable only for small values of n. A commonly used strategy [15,19,20] for selecting the initial sample points is to choose $n + 1$ corner points of the box Ω, and the central point of Ω, but this could prioritize the exploration in a part of the domain. [20] chooses x^L and $x^L + e_i^T(x_i^U - x_i^L)$

input : oracle for f, domain $[x^L, x^U]$, maximum # evaluations $n_{\max}$
output: best solution found within $n_{\max}$ evaluations

evaluate f at k_0 starting points $x_1 \dots x_{k_0}$ with $n+1 \leq k_0 \leq n_{\max}$;
$i \longleftarrow \arg\min\{f(x_i), \forall i \in \{1, \dots, k_0\}\}$;
$(x^*, f^*) \longleftarrow (x_i, f(x_i))$;
$k \longleftarrow k_0$;
while $k < n_{\max}$ **do**
  compute the interpolant $s_k(x)$ using the k points evaluated so far;
  choose a target value f_k^* and select the next evaluation point x_{k+1};
  evaluate $f(x_{k+1})$ through the oracle;
  if $f(x_{k+1}) < f^*$ **then**
    $x^* \longleftarrow x_{k+1}$;
    $f^* \longleftarrow f(x_{k+1})$;
  end
  $k \longleftarrow k+1$;
end
return (x^*, f^*)

Algorithm 1: Pseudo-code of the RBF method.

for $i = 1, \dots, n$ as initial corner points, where e_i is the i-th vector of the standard orthonormal basis. Note that these points may not be feasible for Ω_I. We can round the integer components of the points to achieve feasibility.

Another commonly used strategy is to use a Latin Hypercube experimental design, typically chosen among some randomly generated Latin Hypercube designs according to a maximum minimum distance or a minimum maximum correlation criterion. Again, points sampled this way may not be feasible for Ω_I, and we apply rounding. Note that some of the rounded points may coincide, in which case additional sample points have to be constructed because we require linear independence. This is our default strategy in the computational experiments.

[20] considers the case where some explicit constraints are given in the problem formulation and added to Ω_I, and suggests sampling more points than strictly necessary (i.e. $> n+1$), and picking the first $n+1$ feasible ones. In practice, feasibility for Ω_I should not be too difficult to obtain, otherwise solving the initial problem (1) is hopeless. In our setting, where Ω_I is a box with integrality on some variables, the simple rounding strategy appears to be sufficient for practical purposes.

2.2 Selection of the target value f_k^*

To tackle the problem of selecting the target value f_k^* at each Iteration step, we employ the technique proposed in [20], that generalizes [15], as described below. Let $y^* := \arg\min_{x \in \Omega_I} s_k(x)$, $f_{\min} := \min_{i=1,\dots,k} f(x_i)$, and $f_{\max} := \max_{i=1,\dots,k} f(x_i)$. In particular, we employ a cyclic strategy that picks target values $f_k^* \in [-\infty, s_k(y^*)]$ according to the following sequence of length $\kappa + 2$:

– Step -1 (*InfStep*): Choose $f_k^* = -\infty$. In this case the problem of finding x_{k+1} can be rewritten as:

$$x_{k+1} = \arg\max_{x \in \Omega_I} \frac{1}{(-1)^{d_{\min}+1}\mu_k(x)}.$$

This is an *exploration* phase: the algorithm tries to improve the surrogate model in unknown parts of the domain.

– Step $h \in \{0, \ldots, \kappa - 1\}$ (*Global search*): Choose

$$f_k^* = s_k(y^*) - (1 - h/\kappa)^2(f_{\max} - s_k(y^*)). \tag{12}$$

In this case, there is a balance between improving the model quality and finding the minimum. Notice that if $f_k^* = s_k(y^*)$, then

$$x_{k+1} = \arg\max_{x \in \Omega_I} \frac{1}{(-1)^{d_{\min}+1}s_k(x)}. \tag{13}$$

Hence, if $(-1)^{d_{\min}+1} = 1$ there is not need to solve the problem, as $x_{k+1} = y^*$.

– Step κ (*Local search*): If $s_k(y^*) < f_{\min} - 10^{-10}|f_{\min}|$ accept y^* as the new sample point x_{k+1} without solving (11). Otherwise choose $f_k^* = f_{\min} - 10^{-2}|f_{\min}|$. This is an *exploitation* phase: we try to find the best objective function value based on the current surrogate model.

The choice of the target values is important for convergence of the method. In order to show ϵ-convergence to a global optimum for any continuous function, it is necessary and sufficient [33] that the sequence of points (x_k) generated by the algorithm is dense in the projection of Ω_I over $\mathbb{R}^{n-q}$, i.e. the continuous variables, for every value of the integer variables in $\mathbb{Z}^q \cap \Omega_I$. [15] considers the case where $q = 0$ and shows that (x_k) is dense over Ω when ϕ is linear, cubic or thin plate spline, if f^* is "small enough" throughout the optimization algorithm.

Theorem 1 *[15] Let $q = 0$, $\phi(r) = r$, $\phi(r) = r^2 \log r$ or $\phi(r) = r^3$. Further, choose the integer m such that $0 \leq m \leq n$ in the linear case, $1 \leq m \leq n + 1$ in the thin plate spline case, and $1 \leq m \leq n + 2$ in the cubic case. Let (x_k), $k \in \mathbb{N}$ be the sequence generated by Algorithm 1, and s_k be the the RBF that interpolates $(x_i, f(x_i))$, $\forall i \in \{1, \ldots, k\}$. Assume that, for infinitely many $k \in \mathbb{N}$, the choice of f_k^* satisfies:*

$$\min_{y \in \Omega} s_k(y) - f_k^* > \tau \Delta_k^{\rho/2} \|s_k\|_\infty, \tag{14}$$

where $\tau > 0, 0 \leq \rho < d_{\min}$ are constants, and $\Delta_k := \min_{1 \leq i \leq k-1} \|x_k - x_i\|$. Then the sequence (x_k) is dense in Ω.

Corollary 1 *[15] Let f be continuous and the assumptions of Theorem 1 hold. Furthermore, assume that $f_k^* = -\infty$ for infinitely many $k \in \mathbb{N}$. Then Algorithm 1 converges to a global optimum of f as $k \to \infty$.*

Despite what Corollary 1 suggests, the computational evaluations of the RBF method in the literature typically skip *InfStep* ($f_k^* = -\infty$). This can be explained by the fact that *InfStep* completely disregards the objective function, hence it rarely helps speeding up convergence in practice. We remark that [15] provides a simplified version of equation (14) that does not contain $\|s_k\|_\infty$, giving a condition that is easy to check algorithmically. However, it seems unlikely that the simplified formula can be of practical use, because the global convergence guarantee requires an infinite number of function evaluations anyway.

2.3 Improvements over the basic algorithm

Several modifications of Algorithm 1 and the target value selection strategy have been proposed in the literature, with the aim of improving the practical performance of the algorithm. We now describe the modifications that are active in our default implementation of the algorithm, all of which except the last one are taken from existing literature. These modifications can be turned off or replaced by alternative routines that fulfill the same role; the alternatives are documented in the software. In this paper we follow the settings suggested by the literature. A computational evaluation for some of these settings is given in Section 5.

- [19] suggests transforming the domain of f into the unit hypercube. This strategy is implemented in the *rbfSolve* function of the MATLAB toolkit TOMLAB. In our tests, we found this transformation to be beneficial only when the bounds of the domain are significantly skewed. When all variables are defined over an interval of approximately the same size we did not observe any benefit from this transformation, and in fact sometimes performance deteriorated. Note that the transformation cannot be applied on integrality-constrained variables. After computational testing, our default strategy is to transform the domain into the unit hypercube on problems with no integer variables and such that the ratio of the lengths of the largest to smallest variable domain exceeds a given threshold, set to 5 by default.
- To prevent harmful oscillations of the RBF interpolant due to large differences in the function values, [15] suggests clipping the function values $f(x_i)$ at the median (in other words, replacing values larger than the median by the median). This approach is also adopted by [3,27]. We follow this approach with one small change: function values are clipped at the median only if the ratio of the largest to smallest absolute function value exceeds a given threshold, set to 10^3 by default.
- For the same reason of preventing large differences in function values, it has been proposed to rescale the codomain of f. [30] uses the *plog* (paired log) approach, which consists in replacing each function value using the

following transformation:

$$\mathrm{plog}(x) = \begin{cases} \log(1+x) & \text{if } x \geq 0, \\ -\log(1-x) & \text{if } x < 0. \end{cases}$$

[20] replaces the values $f(x_i) > \max(0, f_{\min}) + 10^5$ with $\max(0, f_{\min}) + 10^5 + \log_{10}(f(x_i) - \max(0, f_{\min}) + 10^5)$. Our implementation offers the following three choices for function scaling:

  - *off*: we employ the original, unscaled function values;
  - *log* scaling: if $f_{\min} \geq 1$ we replace each $f(x_i)$ with $\log(f(x_i))$, otherwise we replace it with $\log(f(x_i) + 1 + |f_{\min}|)$ (similar to [20]);
  - *affine* scaling: we replace each $f(x_i)$ with $\frac{f(x_i) - f_{\min}}{f_{\max} - f_{\min}}$.

- In the *Global search* step, [15,27] replace $f_{\max}$ in equation (12) with a dynamically chosen value $f(x_{\pi(\alpha(k))})$, defined as follows. Let k_0 be the number of initial sampling points, h the index of the current *Global search* iteration as in Section 2.2, π a permutation of $\{1, \ldots, k\}$ such that $f(x_{\pi(1)}) \leq f(x_{\pi(2)}) \leq \cdots \leq f(x_{\pi(k)})$, and

$$\alpha(k) = \begin{cases} k & \text{if } h = 0 \\ \alpha(k-1) - \left\lfloor \frac{k-k_0}{\kappa} \right\rfloor & \text{otherwise.} \end{cases}$$

  As a result, $f_{\max}$ is used to define the target value f_k^* only at the first step ($h = 0$) of each *Global search* cycle. In subsequent steps, we pick progressively lower values of $f(x_i)$, so as to stabilize search by avoiding too large differences between the minimum of the RBF interpolant, and the target value.
- If the initial sample points are chosen with a random strategy (for example, a Latin Hypercube design), whenever we detect that the algorithm is stalling, we apply a complete restart strategy [27, Sect. 5]. Restart strategies have been applied to numerous combinatorial optimization problems, such as satisfiability [14] and integer programming [1]. In the context of the RBF algorithm, the restart strategy as introduced in [27] works by restarting the algorithm from scratch (including the generation of new initial sample points) whenever the best known solution does not improve by at least a specified value (0.1% by default) after a given number of optimization cycles (5 by default). In our experience restarts tend to be more useful if the initial sample points are chosen according to a randomized strategy (otherwise the random number generator has impact only on the solvers for the auxiliary problems).
- A known issue of the RBF method, explicitly pointed out in [27, Sect. 4], is that large values of h in *Global search* do not necessarily imply that the algorithm is performing a "relatively local" search as intended. In fact, the next iterate can be arbitrarily far from the currently known best point, and this can severely hamper convergence on problems where the global minimum is in a steep valley. To alleviate this issue, [27, Sect. 4.3] proposes a "restricted global minimization of the bumpiness function". The basic idea

is to progressively restrict the search box around the best known solution during a *Global search* cycle. In particular, instead of solving (13) over Ω_I, we intersect Ω_I with the box $[\min_{y\in\Omega_I} s_k(y) - \beta_k(x^U x^L), \min_{y\in\Omega_I} s_k(y) + \beta_k(x^U x^L)]$, where $\beta_k = 0.5(1 - h/\kappa)$ if $(1 - h/\kappa) \leq 0.5$, and $\beta_k = 1$ otherwise (the numerical constants indicated are the values suggested by [27]). It is easy to verify that this restricts the global search to a box centered on the global minimizer of the RBF interpolant: the box coincides with Ω_I at the beginning of every *Global search* cycle, but gets smaller as h increases. This turns out to be very beneficial on problems with steep global minima.
– A simple strategy that we found to be effective (see Section 5) is to repeat the *Local search* step in case a *Local search* successfully improves the best known solution. In our experiments, it was not beneficial to perform *Local search* more than twice in a row, as this runs a high risk of focusing too much on a local minimum, forsaking global search.

3 Automatic model selection

One of the drawbacks of the RBF method is that there is no mechanism to assess model quality. There are many possible surrogate models depending on the choice of the basis functions among those of Table 1, and it is difficult to predict a priori which one of these models would have the best performance on a specific problem.

We propose an assessment of model quality using a cross validation scheme. This allows us to dynamically choose the surrogate model that appears to be the most accurate for the problem at hand. Cross validation is a commonly used model validation technique in statistics. Given a data set, cross validation consists in using part of the data set to fit a model, and testing its quality on the remaining part of the data set. The process is then iterated rotating the parts of the data set used for model fitting and for testing.

Let s_k be our surrogate model for f based on k evaluation points $x_1, \ldots, x_k$. We assume that the points are sorted by increasing function value: $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_k)$. We perform cross validation as follows. For $j \in \{1, \ldots, k\}$, we can fit a surrogate model $\tilde{s}_{k,j}$ to the points $(x_i, f(x_i))$ for $i = 1, \ldots, k, i \neq j$ and evaluate the performance of $\tilde{s}_{k,j}$ at $(x_j, f(x_j))$. In particular, we compute the value $q_{k,j} = |\tilde{s}_{k,j}(x_j) - f(x_j)|$ to assess the predictive power of the model. We then average $q_{k,j}$ over $j = 1, \ldots, k$ to compute a model quality score. This approach is known as leave-one-out cross validation in statistics.

We perform model selection at the beginning of every cycle of the search strategy to select f_k^*, see Section 2.2. Our aim is to select the RBF model with the best predictive power. Since the algorithm iterates between local search and global search, we choose two different models: one for local search, one for global search. We achieve this goal by computing the average value $\bar{q}_{10\%}$ of $q_{k,j}$ for $j = 1, \ldots, \lfloor 0.1k \rfloor$, and the average value $\bar{q}_{70\%}$ of $q_{k,j}$ for $j = 1, \ldots, \lfloor 0.7k \rfloor$, for a subset of the basis functions of Table 1. The rationale of our approach is that $\bar{q}_{10\%}$ is an estimate of how good a particular surrogate model is at

predicting function values for the points that have a low function value, which are arguably the most important for local search. On the other hand, for global search it seems reasonable to choose a model that has good predictive performance on a larger range of function values, hence we use $\bar{q}_{70\%}$. The points with the highest function values are the farthest from the minimum and our assumption is that they can be disregarded.

The RBF model with the lowest value of $\bar{q}_{10\%}$ is employed in the subsequent optimization cycle for the *Local search* step and the *Global search* step with $h = \kappa - 1$, while the RBF model with lowest value of $\bar{q}_{70\%}$ is employed for all the remaining steps. In our experiments, the RBF models that we consider are those with cubic, thin plate spline or multiquadric (with $\gamma = 1$) basis functions. We exclude the linear basis function because in our experience it sometimes leads to numerically unstable models, and its inclusion did not yield a noticeable performance increase in terms of model quality. It is possible to also include different scaling parameters as described in Section 2.3 in the evaluation, but we did not pursue this possibility.

A drawback of leave-one-out cross validation is that it is typically expensive to perform. However, in the setting of this paper the number of points k is usually low, hence computing $\bar{q}_{10\%}$ and $\bar{q}_{70\%}$ only takes fraction of a second. We can show that these two values can be computed by solving a sequence of LPs that can be efficiently warmstarted. In our experiments this turned out to be unnecessary and not worth the overhead of communicating with an LP solver, but we believe that this approach may be of interest for larger values of k, therefore we give an overview of the main idea.

To carry out the cross validation scheme, we must compute k RBF interpolants. Instead of repeatedly solving the linear system (4), we can set up an optimization problem as follows:

$$\begin{aligned} & \min\ (-1)^{d_{\min}+1}\lambda^T \Phi\lambda \\ & \text{s.t.:} \quad \Phi\lambda + Ph + \xi = F \\ & \qquad\qquad P^T\lambda = 0_{n+1} \\ \forall i \in \{1,\dots,k\} \setminus \{j\} \qquad & 0 \ \le\ \xi_i \le 0 \\ & -\infty \ \le\ \xi_j \le +\infty \\ & \qquad\qquad \lambda_j = 0. \end{aligned} \tag{15}$$

In (15) we minimize the bumpiness of the interpolant, subject to the interpolation conditions. Observe that in this problem, the j-th interpolation point is ignored: the corresponding constraint is relaxed because of the free slack variable, and λ_j is set to zero so the RBF centered on x_j has no contribution. (15) is a QP, but we now show that it can be reformulated into an LP. Using the equality constraints, we can rewrite the objective function:

$$\lambda^T \Phi\lambda = \lambda^T (F - Ph - \xi) = \lambda^T F - \lambda^T Ph - \lambda^T \xi.$$

Furthermore, $P^T\lambda = 0_{n+1}$, and we can write:

$$\lambda^T \xi = \sum_{i=1,\, i\neq j}^{k} \lambda_i \xi_i + \lambda_j \xi_j = 0.$$

This holds because $\lambda_j = 0$ and $\xi_i = 0\ \forall i \neq j$. Hence the objective function of (15) can be rewritten as $\lambda^T F$, and the problem can be reformulated as the following LP:

$$\begin{aligned} & \min\ (-1)^{d_{\min}+1} \lambda^T F \\ & \text{s.t.:} \quad \Phi\lambda + Ph + \xi = F \\ & \qquad\qquad P^T\lambda = 0_{n+1} \\ \forall i \in \{1,\dots,k\} \setminus \{j\} \qquad & 0 \leq \xi_i \leq 0 \\ & -\infty \leq \xi_j \leq +\infty \\ & \qquad\qquad \lambda_j = 0. \end{aligned} \tag{16}$$

Notice that changing the index j in (16) involves modifications of the variable bounds. Therefore, we can solve a sequence of LP problems of the form (16) with the dual simplex method.

4 The RBF method with a noisy oracle

In practical applications it is common to have a trade-off between computing time and accuracy of the black-box function $f(x)$: the simulation software used to compute $f(x)$ can often be parameterized to achieve different levels of precision. In particular, accuracy of simulations is typically not linear in computing time, therefore one can get reasonable estimations of the true value of $f(x)$ in a fraction of the time.

To speed-up the optimization process, we would like to exploit low accuracy but faster simulations. We assume that in addition to the oracle $f(x)$ we have access to $\tilde{f}(x)$ such that $f(x) = \tilde{f}(x)(1 + \varepsilon_r) + \varepsilon_a$, where $\varepsilon_r, \varepsilon_a$ are random variables with bounded support and unknown distribution. This corresponds to having a relative error term and an absolute error term on top of the "true" function value $f(x)$. We assume that ε_r takes values in $[-\epsilon_r, \epsilon_r]$, ε_a takes values in $[-\epsilon_a, \epsilon_a]$, and both ϵ_r, ϵ_a are known. In practice, the values of ϵ_r, ϵ_a can be determined on the basis of domain knowledge for the specific simulation software, or an estimation based on practice. The approach that we propose works in any situation where ϵ_r, ϵ_a overestimate the true error terms, therefore even a rough estimate suffices. We assume that evaluating $\tilde{f}$ is less expensive than evaluating f, i.e. $\tilde{f}$ can be computed significantly faster.

Our approach consists in: a first stage where we aim to solve (1) using $\tilde{f}(x)$ and employing the RBF method until some termination condition is met, typically based on number of function evaluation or an estimation of model quality; a second stage, where (1) is reoptimized performing additional function evaluations using $f(x)$. Recall that the RBF algorithm, as described in Section 2, does not allow new function evaluations at previously evaluated points $x_1, \dots, x_k$. Let x_χ denote the last point at which $\tilde{f}$ was sampled, and $L = \{1, \dots, \chi\}$. Because function values known so far $\tilde{f}(x_1), \dots, \tilde{f}(x_\chi)$ are affected by error under our assumptions, we must modify the algorithm so that subsequent function values at points $x_k, k > \chi$ can replace previous ones, if necessary. Furthermore, it is reasonable to allow the interpolant to deviate from

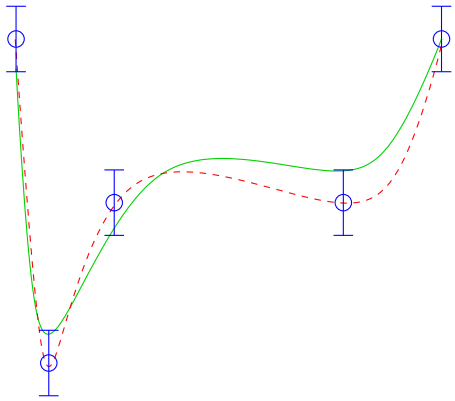

Fig. 2 Function evaluations affected by errors: the values returned by $\tilde{f}$ are the circles. The dashed line interpolates exactly at those points, the solid line is a less bumpy interpolant, and is still within the allowed error tolerances. Problem (17) would prefer the latter interpolant.

the values $\tilde{f}(x_1), \ldots, \tilde{f}(x_\chi)$ by an amount within the allowed error estimates ϵ_r, ϵ_a. In particular, instead of solving (4) to determine the interpolant, we introduce a vector of slack variables $\xi \in \mathbb{R}^k$ and solve the problem:

$$\begin{array}{rrl} \min & (-1)^{d_{\min}+1}\lambda^T \Phi \lambda & \\ \text{s.t.:} & \Phi\lambda + Ph + \xi = F & \\ & P^T\lambda = 0_{n+1} & \\ \forall i \in L & -\epsilon_r|\tilde{f}(x_i)| - \epsilon_a \leq \xi_i \leq \epsilon_r|\tilde{f}(x_i)| + \epsilon_a & \\ \forall i \in \{1,\ldots,k\} \setminus L & \xi_i = 0. & \end{array} \tag{17}$$

Here, F is assumed to contain $\tilde{f}(x_i)$ instead of $f(x_i)$ for all $i \in L$. Problem (17) minimizes the bumpiness of the RBF interpolant, subject to the interpolation conditions. The inequalities involving ξ allow the interpolant to take any value within the error tolerances ϵ_r, ϵ_a of the noisy function values $\tilde{f}(x_i), i \in L$. A sketch of this idea is given in Figure 2. If we set $\xi = 0$ for all i, thereby eliminating ξ from the problem, deriving the KKT optimality conditions recovers the original system (4). Note that (17) admits at least one solution if (4) admits a solution, and (17) is a convex quadratic problem because of the conditional positive semidefiniteness of Φ. In practice, to avoid numerical difficulties in the solution of (17) we use a local solver starting from the solution of (4).

A drawback of this method is that it requires an estimation of ϵ. A related approach was adopted by [21], whereby all function values are allowed to deviate from the given $f(x_1), \ldots, f(x_k)$, but these deviations are penalized in the objective function according to a pre-specified penalty parameter. The difference between our approach and the one of [21] is that we require to specify the range within which function values are allowed to vary, whereas [21] requires to specify the value of the penalty parameter in the objective function and computes the error terms accordingly. We believe that estimating a penalty parameter may prove harder in practice than providing an error range, hence our approach may be more natural for practitioners.

Our approach to allow new function evaluations to take place at previously evaluated points is the following. We compute x_{k+1} solving (11) where target values are chosen according to the cyclic strategy described in Section 2. We have $x_{k+1} \in \Omega_I \setminus \{x_1, \ldots, x_k\}$ by construction. For $w \in L$, let $s_{k,w}$ be the RBF interpolant subject to the conditions:

$$\begin{aligned} s_{k,w}(x_i) &= \tilde{f}(x_i) && \forall i \in L \setminus \{w\}, \\ s_{k,w}(x_i) &= f(x_i) && \forall i \in \{1, \ldots, k, \} \setminus L, \\ s_{k,w}(x_w) &= f_k^*, \end{aligned} \tag{18}$$

and let s_k^* be the interpolant subject to the conditions:

$$\begin{aligned} s_k^*(x_i) &= \tilde{f}(x_i) && \forall i \in L, \\ s_k^*(x_i) &= f(x_i) && \forall i \in \{1, \ldots, k, \} \setminus L, \\ s_k^*(x_{k+1}) &= f_k^*, \end{aligned} \tag{19}$$

both of which can be computed solving (17). When we are in the *Local search* phase of the target value selection strategy, we compare $\sigma(s_{k,w})$ with the value of $\sigma(s_k^*)$ for all $w \in L$ such that $f_k^* \in [\tilde{f}(x_w) - \epsilon_r |\tilde{f}(x_w)| - \epsilon_a, \tilde{f}(x_w) + \epsilon_r |\tilde{f}(x_w)| + \epsilon_a]$. In other words, we compare the bumpiness of the RBF interpolant at the suggested new point x_{k+1}, with the bumpiness of the RBF interpolant if the function value at x_w were set to the target f_k^*. We do this only for points x_w such that f could take the value f_k^* at x_w, according to the specified error bounds. This way, we can verify whether we obtain a smoother interpolant by placing the new point at a previously unexplored location, or at one of the previously existing points. If this is the case and $\sigma(s_{k,w}) < \sigma(s_k^*)$ for some $w \in L$, we evaluate the function $f(x_w)$ (i.e. the exact oracle f rather than the noisy oracle $\tilde{f}$), replace the corresponding value, and set $L \leftarrow L \setminus \{w\}$. Note that this step will be performed at most $|L| = \chi$ times and does not affect global convergence in the limit.

A further modification of the algorithm, that we found to be beneficial in practice, is to evaluate $f(x)$ at points where $\tilde{f}(x)$ has a potentially optimal or satisfactory function value. In particular, if a target objective function value is known, after every evaluation of $\tilde{f}(x)$ we check whether the returned value could be optimal up to the optimality tolerance and the error terms $\varepsilon_r, \varepsilon_a$. If that is the case, we immediately evaluate $f(x)$ at the same point and use the corresponding function value. In our experiments, this significantly accelerated convergence. From a practical perspective, while we cannot expect that the optimal objective function value be known in advance, domain knowledge can often provide a target value and an optimality tolerance such that solutions within the specified tolerance are considered satisfactory, hence our approach can be applied.

5 Computational experiments

In this section we discuss computational experiments performed with RBFOpt, our implementation of the RBF method. All experiments were carried out on a server equipped with four Intel Xeon E5-4620 CPUs (2.20 GHz, 8 cores, Hyper Threading and Turbo Boost disabled) and 128GB RAM (32GB for each processor), running Linux.

5.1 Implementation

We implemented the RBF method in Python and in MATLAB/Octave. The tests reported in this paper use our Python implementation, therefore we refer to the Python version, that relies on Pyomo [17,16] and the Coopr library. We use PyDOE to generate experimental designs, and NumPy for linear algebra. The Python implementation is Python3-compliant, but some of the required packages – most notably Coopr – rely on Python2.7, hence we only support Python2.7 until all required packages switch to Python3.

Our implementation can be downloaded from the website of the authors. It is open-source and is free for non-commercial academic use: more specifically, academic users are free to use, modify and redistribute the source code under the same licensing terms. The exact licensing terms are available on the website.

To solve the nonlinear (mixed-integer in the presence of integer variables) optimization problems generated during the various steps of the algorithm, a nonlinear solver is necessary. In our tests, we use IPOPT [34] for continuous problems, and BONMIN [4] with the NLP-based Branch-and-Bound algorithm for mixed-integer problems. To optimize nonconvex functions such as (10), we rely on a simple multi-start strategy for IPOPT if there are no integer variables, and on BONMIN's Branch-and-Bound algorithm in the presence of integer variables.

Besides our own version, we are aware of only one available implementation of the RBF method: rbfSolve in the commercial TOMLAB toolkit for MATLAB.

5.2 Test instances

We test our implementation on 26 unconstrained problems taken from the literature, listed in Table 2. We provide more information on the instances below. These problems were originally proposed as a testbed for global optimization solvers, and are now considered fairly easy in terms of global optimization. However, they can prove very challenging for black-box solvers that do not exploit analytical information on the problems.

– Dixon-Szegö [11] problems: we included the most common instances in the test set. These instances are the de-facto standard test set used in all com-

Table 2 Details of the instances used for the tests.

Instance	Dimension	Domain	Type	Source
branin	2	$[-5, 10] \times [0, 15]$	NLP	Dixon-Szegö [11]
camel	2	$[-3, 3] \times [-2, 2]$	NLP	Dixon-Szegö [11]
ex4_1_1	1	$[-2, 11]$	NLP	GLOBALLIB
ex4_1_2	1	$[1, 2]$	NLP	GLOBALLIB
ex8_1_1	2	$[-1, 2] \times [-1, 1]$	NLP	GLOBALLIB
ex8_1_4	2	$[-2, 4] \times [-5, 2]$	NLP	GLOBALLIB
gear	4	$[12, 60]^4$	MINLP	MINLPLib [5]
goldsteinprice	2	$[-2, 2]^2$	NLP	Dixon-Szegö [11]
hartman3	3	$[0, 1]^3$	NLP	Dixon-Szegö [11]
hartman6	6	$[0, 1]^6$	NLP	Dixon-Szegö [11]
least	3	$[0, 600] \times [-200, 200] \times [-5, 5]$	NLP	GLOBALLIB
nvs04	2	$[0, 50]^2$	MINLP	MINLPLib [5]
nvs06	2	$[1, 50]^2$	MINLP	MINLPLib [5]
nvs09	10	$[3, 9]^{10}$	MINLP	MINLPLib [5]
nvs16	2	$[0, 50]^2$	MINLP	MINLPLib [5]
perm0_8	8	$[-1, 1]^8$	NLP	Neumaier [25]
perm_6	6	$[-6, 6]^6$	NLP	Neumaier [25]
rbrock	2	$[-10, 5] \times [-10, 10]$	NLP	GLOBALLIB
schoen_10_1	10	$[0, 1]^{10}$	NLP	Schoen [32]
schoen_10_2	10	$[0, 1]^{10}$	NLP	Schoen [32]
schoen_6_1	6	$[0, 1]^6$	NLP	Schoen [32]
schoen_6_2	6	$[0, 1]^6$	NLP	Schoen [32]
shekel10	4	$[0, 10]^4$	NLP	Dixon-Szegö [11]
shekel5	4	$[0, 10]^4$	NLP	Dixon-Szegö [11]
shekel7	4	$[0, 10]^4$	NLP	Dixon-Szegö [11]

putational evaluations of the RBF method and in many other derivative-free approaches, see e.g. [15,27,19].

- MINLPLib [5] problems: we included all unconstrained instances in the library. For some problems, none of the tested algorithms was able to find the optimal solution if applied on the problem with the original variable bounds. Therefore, in some cases we restricted the bounds to decrease the difficulty.
- GLOBALLIB problems: we selected a subset of the unconstrained instances in the library. Some problems were excluded because too easy or too similar to other problems in our collection.
- Schoen [32] problems: we randomly generated two problems of dimension 6 and two of dimension 10. All problems have 50 stationary point, three of which are global minima with value -1000, and the remaining ones attain a value picked uniformly at random in the interval $[0, 1000]$. Having steep global minima allows us to test the performance of our implementation in a situation that is considered difficult to handle for the RBF method, see [27]. Another advantage of using problems of this class is that we can choose the dimension of the space. In particular, we test problems with 10 decision variables, which is larger than most of the instances encountered in RBF literature. To avoid overrepresentation of a class of instances in our test set, we generate only four random problems of this class.

– Neumaier [25] problems: we included one problem of class "perm", and one of class "perm0", generated with parameters $n = 6, \beta = 60$ and $n = 8, \beta = 100$ respectively. These problems were conceived to be challenging for global optimization solvers, and are in our experience very difficult to solve with a black-box approach. The global minimum of these instances is originally 0, but achieving an optimality tolerance of 1% or 0.01 is essentially hopeless for these problems. Hence, we translated the functions up by 1000.

An extensive computational evaluation of black-box solvers is discussed in [31], which uses a much larger test set than ours. However, the setting of that paper is different because the variable bounds are relatively large, the problem dimension is typically higher, and a larger budget of function evaluations is allowed (up to 2500, while we limit ourselves to 150). The type of problems on which the RBF method is expected to perform better is different, and for these reasons, [31] does not provide computational results for any implementation of the RBF method despite discussing it.

5.3 Comparison of algorithmic settings

The following list summarizes the different settings that we considered, see Sections 2.3 and 3 for details:

– scaling [affine, log, off]: the type of scaling used;
– R: restart the algorithm after 6 cycles without improvement of the best solution found;
– B: restricted global minimization of the bumpiness function;
– L: if the local search step improves the best solution, it is repeated a second time;
– auto: automatic model selection using cross validation to choose the basis function

The "default" configuration employs the cubic basis function, a random Latin Hypercube design (generated with the maximum minimum distance criterion) for the selection of the first sampling points, no InfStep, and 5 global search steps (i.e., $\kappa = 5$). This is in accordance with [15, 27]. The number of function evaluations is capped at 150, the time limit for the NLP solver is set to 60 seconds, and for the MINLP solver to 120 seconds. We parameterize BONMIN to repeat NLP solutions up to 20 times at the root (effectively, this acts as a multi-start approach on nonconvex continuous problems), and 10 times at nodes in case of infeasibility. The time limit for each run is set to 4 hours. Typically, hitting this time limit is indicative of numerical problems in the solution process, e.g. the system (4) becomes badly conditioned. If this happens, we consider the corresponding run as a failure.

We evaluate the performance obtained with our implementation on the test instances of Table 2. Detailed results are given in the Appendix; here we give a summary reporting: the geometric mean of the number of function evaluations

to find a solution within 1% of the global optimum (over 20 runs with different random seeds – a value of 150 evaluations was used for failed runs), the geometric standard deviation in parentheses, the total number or successful runs. Each row represents a different configuration of the algorithm. The best values are in boldface. The geometric means are computed as follows: first, for each instance we compute the arithmetic average of the number of function evaluations. Then, we compute the geometric mean of these arithmetic averages across the instances. The reason for choosing this approach is that within the same instance, we perform several random trials to get an estimate of the expected number of function evaluations through sampling, hence the arithmetic average is the natural estimator. After obtaining these numbers, we aggregate them with a geometric mean so that each instance is given equal weight, rather than putting more emphasis on problems that require more function evaluations (such as problems where the algorithm does not converge within the 150 function evaluations).

To compare different versions of the algorithm, we perform a Friedman test using the average number of function evaluations on each instance as blocks (rows), and the versions of the algorithm as groups (columns). The null hypothesis of the test is that there is no difference among the groups. This allows us to assess if one of the algorithms is consistently better than the others on the majority of the instances. For details on and assumptions of the Friedman test, we refer to [8]. Note that the Friedman test does not take into account the magnitude of the differences among the values, but results with the Quade test (a non-parametric statistical test that takes into account differences in magnitude) are essentially in agreement, hence we only report results for the Friedman test. All comparisons are performed at the 95% significance level. If an algorithm on the row is better (i.e. fewer function evaluations according to the Friedman test) than an algorithm on the column, we indicate it with a "*****". This is detected using post-hoc analysis when the p-value < 0.05. The p-value is reported in the caption, along with a reference to the table(s) in the Appendix with the detailed results.

We would like to answer the following research questions:

1. Which algorithmic configuration is the best, and in particular, are the improvements of Section 2.3 beneficial in practice?
2. Is our approach to handle noisy function evaluations effective?
3. Is automatic model selection using cross validation beneficial in practice?
4. Is our implementation competitive with the state-of-the-art?
5. Can the surrogate models produced by the algorithm be useful to perform sensitivity analysis around the optimum?

The first question is investigated in the rest of this section. The second and the third questions are investigated in Sections 5.4-5.5. The fourth question is discussed in Section 5.6. The fifth question is discussed in Section 5.7.

We report the performance of with different settings of the algorithms and different scaling procedures in Tables 3-5. It appears that "off" scaling is always not worse and sometimes better than other scaling procedures. Similar

Table 3 Results obtained with the default configuration and different function value scaling procedures. Data taken from Table 16. The Friedman test does not reject the null hypothesis (p-value 0.111).

Setting	# evaluations	# solved
affine	75.04 (2.75)	216
log	67.95 (2.95)	215
off	**66.99** (2.93)	**231**

Table 4 Results obtained with the RB configuration and different function scaling procedures. Data taken from Table 17. According to a Friedman test, "off" scaling performs better than "affine" (p-value 0.025).

Setting	# evaluations	# solved	affine	log	off
affine	66.07 (2.81)	296			
log	64.81 (2.97)	234			
off	**61.69** (2.79)	**318**	*		

Table 5 Results obtained with the RBL configuration and different function value scaling procedures. Data taken from Table 18. According to a Friedman test, "off" and "affine" scaling perform better than "log" (p-value 0.015).

Setting	# evaluations	# solved	affine	log	off
affine	64.57 (2.84)	304		*	
log	64.69 (2.97)	239			
off	**60.76** (2.82)	**329**		*	

Table 6 Results obtained with "off" scaling and different algorithm settings (see Tables 16-20, off scaling columns). According to a Friedman test, "BL" and "RBL" perform better than default, "R", "L", "RL" (p-value 0.000).

Setting	# evaluations	# solved	def.	R	B	L	RB	RL	BL	RBL
def.	66.99 (2.93)	231								
R	67.79 (2.95)	226								
B	60.81 (2.78)	**332**	*	*				*		
L	66.03 (2.93)	242		*						
RB	61.69 (2.79)	318	*	*						
RL	66.58 (2.94)	240								
BL	**60.38** (2.81)	330	*	*		*		*		
RBL	60.76 (2.81)	329	*	*		*		*		

conclusions can be reached with different algorithmic settings. Hence, we turn function value scaling off in the following. In Table 6 we compare default, "R", "B", "L", "RB", "RL", "BL", "RBL" with no scaling. We can see that combined together, the impact of the algorithmic improvement becomes significant: the reduction in the number of function evaluations with "RBL" is more than 10% as compared to the default configuration, and the number of instances solved increases by 42%. The most impactful settings appear to be "B" and "L", in this order. There is essentially no difference between "B", "BL" and "RBL" in terms of results. However, "RBL" is considerably faster in terms of computing time: more than twice as fast, in our experiments. This is because when the algorithm stalls, the solution of the auxiliary problems

Table 7 Results obtained with the "RBL" configuration without noise, with noise 10%, and with noise 20% (see Table 21). According to a Friedman test, noise 10% performs better than the other algorithms (p-value 0.006).

Setting	# evaluations	# solved	RBL	RBL n10%	RBL n20%
RBL	60.76 (2.81)	**329**			
RBL n10%	**42.67** (3.77)	300	*		*
RBL n20%	45.50 (3.61)	293			

can become very slow. Restarts are helpful in preventing stalling. Since the difference in terms of performance is negligible, we prefer "RBL" to "BL". In the rest of this paper we use the "RBL" configuration. Table 6 answers our first research question.

5.4 Experiments with a noisy oracle

In the context of this computational evaluations, we need a way to simulate the access to a noisy but faster oracle for the function f. To this end, our approach is to simulate the noisy oracle by applying to f a relative noise generated uniformly at random between $\pm 10\%$ or $\pm 20\%$, as well as an absolute noise generated uniformly at random between ± 0.01 (to avoid exact oracle evaluations around zero). We assume that each noisy oracle evaluation has a computational cost of one third of an exact oracle evaluations, i.e. the total number of function evaluations as compared to the algorithm in the previous section is computed as (# exact evaluations) + (# noisy evaluations)/3. This choice is arbitrary but the numbers are realistic in our experience. The stopping criterion for the algorithm is 75 exact function evaluation and 225 noisy evaluations, which is equivalent to 150 exact function evaluations as in the previous section. We set ϵ_r to 10% or 20% and ϵ_a to 0.01. The results of the comparison between the "RBL" configuration with and without noise are presented in Table 7 (for detailed statistics see Table 21).

We can see that the approach we propose yields a significant reduction (more than 25%) in the average number of (equivalent) function evaluations required to converge to the global optimum, even with a 20% relative noise. With a relative noise of 10%, the reduction is by more than 30%. The price to pay, as can be observed in the detailed tables of results, is that the number of solved instances decreases, and as a consequence the geometric standard deviations increase. In particular, the speed-up is large on most of the instances, but performance deteriorates noticeably on the "shekel" and "schoen_10" instances. A possible explanation is that these functions have steep global minima with a low function value: the noisy oracle may give a very poor indication on the location of the corresponding valleys if the relative error is large, and if the algorithm never gets close to the global minimum, exact function evaluations are not used. Hence, the algorithm may be slow or fail to converge. Still, because of the large improvement on the majority of the test set, the

Table 8 Results obtained with and without automatic model selection for "RBL". The Friedman test does not reject the null hypothesis (p-value 0.205).

Setting	# evaluations	# solved
RBL cubic	60.76 (2.82)	329
RBL thin plate spline	60.03 (2.67)	339
RBL multiquadric	67.02 (2.74)	252
RBL auto	**55.68** (2.84)	**345**

Friedman test detects a statistically significant difference at the 95% level in favor of "RBL n10%".

It can be argued that we are testing our approach under the most favorable conditions, namely when the error estimates ϵ_r, ϵ_a are exactly equal to the true maximum relative and absolute noise applied to the function values. This is a valid concern, because we would like our method to work even with rough overestimates of the true noise, given that in practice it may be difficult to obtain accurate estimates. To assess the robustness of the proposed approach in a more challenging context, we repeat the experiments with the "RBL" configuration of our algorithm, setting ϵ_r to 20% and ϵ_a to 0.01, and applying a noise on the oracle for the objective function that is uniformly chosen at random between $\pm 10\%$. In other words, the relative error estimate provided to the algorithm is double the amount of the true relative noise. In this case, "RBL" solves 301 instances, and the geometric mean of the number of function evaluations is 43.09 (geometric standard deviation 3.68). Comparing to Table 7, we can see that there is hardly any difference with the performance of "RBL n10%". A similar observation can be made setting ϵ_r to 30% and applying a true relative noise between $\pm 20\%$: in this case "RBL" solves 292 instances, and the geometric mean of the number of function evaluations is 44.17 (geometric standard deviation 3.73), which is essentially the same performance as "RBL n20%". We conclude that on this test set, the performance of our approach to handle noisy function evaluations seems to have the desirable property of depending on the true noise, rather than the estimated noise ϵ_r.

5.5 Automatic model selection using cross validation

We now proceed to test the automatic model selection method presented in Section 3. We label this configuration "auto", as opposed to the default configuration that uses a pre-determined basis function. We test the "auto" configuration against all three more commonly used types of basis functions: cubic, thin plate spline, and multiquadric. In Tables 8-10 we compare the results obtained with "RBL" with or without noise, and with our without automatic model selection using cross validation. For the tests with noise, we only report results with the cubic basis function, as tests with the other basis functions did not yield additional insight.

Looking at the results, we see that "RBL auto" requires fewer function evaluations (the reduction is $\approx 9\%$) and solves more instances than "RBL

Table 9 Results obtained with and without automatic model selection for "RBL" with noise level 10% (see Table 23). The Friedman test does not reject the null hypothesis (p-value 0.503).

Setting	# evaluations	# solved
RBL n10% cubic	42.68 (3.77)	300
RBL n10% auto	**40.87** (3.59)	**320**

Table 10 Results obtained with and without automatic model selection for "RBL" with noise level 20%. The Friedman test does not reject the null hypothesis (p-value 0.832)

Setting	# evaluations	# solved
RBL n20% cubic	45.50 (3.60)	293
RBL n20% auto	**45.38** (3.57)	**303**

cubic" or "RBL thin plate spline", although the difference is not detected by a Friedman test. This is our best performing algorithm configuration so far, solving more instances than any other tested configuration and showing that the automatic model selection is useful in our experiments. In particular, automatic model selection improves over any one of the three tested basis function, suggesting that it is able to find the best performing model. It is interesting to compare our "auto" configuration with the best single basis function for each instance, i.e. the results that could be obtained in the hypothetical situation of being able to guess the best performing basis function before solving the instance. This "RBL best-basis-function" would require on average 54.33 function evaluations (geometric standard deviation 2.81), solving 364 instances, and is therefore only marginally better than "RBL auto" on our test set. The results suggest that our model selection scheme is able to correctly guess the best surrogate model in most situations.

The same results carry over when exploiting a noisy oracle with relative error at most 10%: automatic model selection is able to reduce the number of function evaluations by $\approx 7\%$, and solves more instances. However, with a relative noise of 20%, the benefit from using automatic model selection thins out considerably and is hardly noticeable. This can be explained with the fact that automatic model selection relies on the function evaluations to assess model performance: in a context where the function evaluations are affected by a significant relative error, assessing model quality becomes difficult, and therefore our proposed procedure brings little advantage. Still, even with a large noise our "auto" configuration is no worse than the default one and finds the global optimum on a few more instances. This answers our third research question.

5.6 Comparison with the literature

In this section we investigate how our implementation compares with results from the literature. The testbed for this section consists of the seven Dixon-

Table 11 Best results obtained with "RBL auto" with Latin Hypercube sampling.

Instance	best
branin	21
goldsteinprice	29
hartman3	16
hartman6	50
shekel10	37
shekel5	63
shekel7	67

Szegö functions, because they are the only functions for which results are consistently reported. We use the version of our algorithm that seems to be the most effective, namely "RBL auto". For most of the papers we cite below, we only report the best available results.

A major issue is that the settings of the computational evaluations are not always reported in full details. Thus, in some cases we could not retrieve exact information about the algorithms. More importantly, many papers report a single result for each instance, i.e. a single number of function evaluations. For implementations of the RBF method, it seems unlikely that any algorithm can be fully deterministic even if the initial sample points are chosen deterministically: the auxiliary problems that have be solved are nonconvex problems that are not solved to global optimality, employing e.g. multistart heuristics. Unfortunately, in some cases we do not know how to interpret the results reported in the papers (i.e. if it is an average number of evaluations over repeated runs, or the best result achieved, or a one-shot test). We report results verbatim from these papers anyway, and we give in Table 11 the best results (over 20 runs) obtained with our "RBL auto" configuration of the RBF algorithm, using a Latin Hypercube for the initial sampling, instead of the averages reported in previous sections.

A summary of the algorithms reported, their settings, and corresponding references is given in Table 12. It is obvious that there are many different settings employed for the different algorithms, and many differences are not captured in Table 12. For example, in [21] the limit on function evaluations is set 150 (if the algorithm failed, it is not counted in the computation of the average), whereas in [28] it is 500. ARBFMIP employs the cubic RBF for the hartman6 and goldsteinprice instances, and thin plate spline for the remaining instances; it also considers 7 corner points and a central point only as initial samples for the hartman6 instance. To add to the confusion, the same algorithm can be called in different ways in different papers: RBF in [15], Gutmann in [21], Gutmann-RBF in [27], and RBFGLOB in [19] are the same algorithm. Similarly, CORS-RBF in [19] is CORS-RBF (sp1) in [28], and CORS-RBF in [27] is CORS-RBF (sp2) in [28].

We report results in different tables depending on the strategy to choose the initial points. Table 13 reports the average number of evaluations for the algorithms presented in Table 12 that employ an initial sampling based on

Table 12 Summary of available information on the algorithms from the literature discussed in our paper: strategy of the initial sampling ((S)LH stands for (Symmetric) Latin Hypercube) and number of points, type of basis function, Table (in this paper) where the results are reported, and references for a description of the algorithm and for the results reported in this paper.

Algorithm	Initial sampling	RBF	Table	References
qualsolve-C	Corners (2^n)	thin plate spline	13	[21]
qualsolve-LH	LH ($n+1$)	thin plate spline	14	[21]
Gutmann-C	Corners (2^n)	thin plate spline	13	[15,21,19,28]
rbfSolve-C	Corners (2^n)	thin plate spline	13	[21]
rbfSolve-LH	LH ($n+1$)	thin plate spline	14	[21]
EGO-C	Corners (2^n)	thin plate spline	13	[21]
EGO-LH	LH ($n+1$)	thin plate spline	14	[21]
CORS-SP1	Corners (2^n)	thin plate spline	13	[21,28]
CORS-SP2	Corners (2^n)	thin plate spline	13	[28]
ARBFMIP	Corners (2^n+1) or ($7+1$)	thin plate spline/cubic	13	[19]
AQUARUS-CGRBF	SLH $2(n+1)$	cubic	14	[30]
AQUARUS-LMSRBF	SLH $2(n+1)$	cubic	14	[30]
CG-RBF-Restart	SLH $(n+1)(n+2)/2$	thin plate spline	14	[27]
CORS-RBF-Restart	SLH $(n+1)(n+2)/2$	thin plate spline	14	[27]
rbfSolve	Corners (2^n)	cubic	15	[3,19,28,27]
DIRECT			15	[15,19,22,28]
EGO			15	[15,19,23,28]
MCS			15	[19]
DE			15	[15,19]
GLOBAL-QN (old/new)			15	[10]
GLOBAL-UNI (old/new)			15	[10]

Table 13 Comparison with the literature, for algorithms such that the initial sample points are chosen as the corner points.

Instance	RBL auto -C	qualsolve -C	Gutmann -C	rbfSolve -C	EGO -C	CORS -SP1	ARBFMIP	CORS -SP2
branin	30.3	32	44	59	21	34	22	40
goldsteinprice	82.4	60	63	84	125	49	21	64
hartman3	26	46	43	18	17	25	31	61
hartman6	102.60	99	112	109	92	108	43	104
shekel10	124.30 (70%)	71	51	(0%)	(0%)	51	25	64
shekel5	119.80 (75%)	70	76	(0%)	(0%)	41	34	52
shekel7	124.15 (70%)	85	76	(0%)	(0%)	46	31	64

corner points, as well as our "RBL auto" configuration initialized with the 2^n corners as starting points. Table 14 does the same for the algorithms employing a Latin Hypercube initial sampling strategy. Finally, Table 15 reports results for the remaining algorithms: either the initial sampling strategy is not required, or it is not specified in the paper.

These tables show that our implementation of the RBF algorithm seems to be competitive with existing methods from the literature, on the Dixon-Szegö test set. It is difficult to draw statistically meaningful conclusions on such a small set of instances, however from the average number of function evaluations to find a global optimum it seems that our algorithm outperforms DIRECT, EGO, DE, GLOBAL, AQUARS, and performs similarly to the best algorithms described in the literature. In particular, our best results seem competitive with the commercial implementation rbfSolve.

Table 14 Comparison with the literature, for algorithms such that the initial sample points are chosen according to a Latin Hypercube design.

Instance	RBL auto	qualsolve -LH	rbfSolve -LH	EGO -LH	AQUARS -CGRBF	AQUARS -LMSRBF	CG-RBF -Restart	CORS-RBF -Restart
branin	30.5	26.9	62.7	23	39.43	31.73	46.60	43.90
goldsteinprice	52.5	30.4	63	(0%)	65.77	35.33	61.60	59.27
hartman3	44.5	38.8	28	31	38.13	46.23	63.17	54.03
hartman6	92.82 (85%)	50.7 (95%)	122	(0%)	129.80	178.70	214.47 (90%)	199.67 (93.3%)
shekel10	83.46 (65%)	78 (60%)	(0%)	(0%)	121.10	179.63	169.33	121.30
shekel5	80.71 (35%)	61 (30%)	(0%)	(0%)	164.67	212.77	259.77 (93.3%)	216.97 (93.3%)
shekel7	96 (25%)	66 (60%)	(0%)	(0%)	152.70	178.03	156.23	150.77

Table 15 Comparison with the literature, for algorithms such that the initial sample is not necessary or is not specified.

Instance	rbfSolve	DIRECT	EGO	MCS	DE	GLOBAL -QN (old)	GLOBAL -QN (new)	GLOBAL -UNI (old)	GLOBAL -UNI (new)
branin	26	63	28	30	1190	330	77	464	172
goldsteinprice	27	101	32	40	1018	436	277	386	446
hartman3	22	83	35	79	476	216	196	697	1449
hartman6	87	213	121	74	7220	1446	703	2610	2614
shekel10	76	97	(0%)	103	6251	2396	2378	2689	3429
shekel5	96	103	(0%)	83	6400	990	1090	1083	1450
shekel7	72	97	(0%)	106	6194	1767	1718	1974	2527

5.7 Using the surrogate model for sensitivity analysis

An advantage of surrogate model based methods for black-box optimization is that they yield a model of the objective function as a byproduct of the optimization process. We want to investigate whether this model can be used to analyze the behaviour of the objective function around the optimum, without resorting to additional oracle evaluations. Clearly, we cannot expect the surrogate model to be fully accurate. We already have a procedure to assess the quality of the surrogate model: we now want to apply it to detect if the surrogate model can be trusted around the optimum. In this section we provide an empirical assessment of this idea.

We proceed as follows. At the end of the optimization process (i.e. one of the following events occur: a solution within 1% of the global optimum is found, we hit the function evaluation limit, or we hit the time limit) we fit an RBF interpolant $\hat{f}$ to all known points, eliminating possible duplicate points in case of restarts. For this interpolant, we choose the basis function that gives the best value of $\bar{q}_{10\%}$. We obtained similar results skipping the automatic basis function selection and using the default (cubic) basis function, hence we only discuss the first case.

Let x^* be the point that attains the best known value when the algorithm terminates, and denote by e_i the i-th vector of the standard orthonormal basis. We compare the value of $f(x^* \pm \Delta_i e_i)$ for $i = 1, \ldots, n$ and small $\Delta_i > 0$ with the value returned by the interpolant: we call the quantity $|f(x^* \pm \Delta_i e_i) - \hat{f}(x^* \pm \Delta_i e_i)|/|f(x^* \pm \Delta_i e_i)|$ the *model error* at the given point. In the formula, all $\pm$ symbols take the same value $+$ or $-$. We test the values $\Delta_i = \delta(x_i^U - x_i^L)$

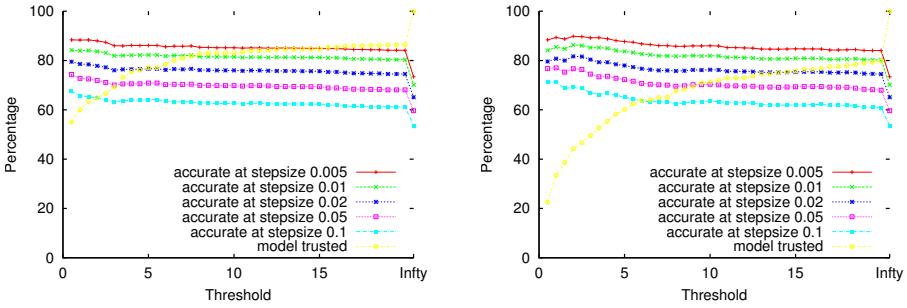

Fig. 3 Percentage of surrogate models that are trusted and corresponding percentage of model errors that are below 10%, for threshold policies of the form $\bar{q}_{10\%} \leq x$ (left figure) and $\bar{q}_{20\%} \leq x$ (right figure), where x is the value on the x-axis. The left axis goes from 0 to 20 with 0.5 increments, but the last point is out-of-scale and indicates $x = \infty$ (label: "Infty").

for $\delta = 0.005, 0.01, 0.02, 0.05, 0.1$. Our main question is whether the values $\bar{q}_{k\%}$ provide useful information about the accuracy of the model for some values of k. We plotted graphs of $\bar{q}_{k\%}$ for $k = 10, 20, \ldots, 100$ against the model errors. This is a large amount of data; we report a summary of our findings.

For practical purposes, we decided that model errors of more than 10% are not acceptable, and that we are looking for a simple threshold policy for $\bar{q}_{k\%}$ to determine whether or not the surrogate model should be trusted around the optimum. For the tested values of k%, we tried different thresholds t and plotted the aggregated model errors. We plot these graphs for $k = 10, 20$ in Figure 3, where we give the fraction of model errors (among all points within the domain located $\pm\Delta_i e_i$ away from x^*, $i = 1, \ldots, n$) that are below 10%, and the fractions of models that are trusted based on the given threshold. Ideally, we want these values to be as high as possible; for $k \geq 30$, all the curves are shifted noticeably towards the bottom, therefore we do not report the corresponding results. From the graphs, it seems that $\bar{q}_{20\%} \leq 10\%$ performs relatively well in practice: up to $\delta = 0.02$, about 75% of the time the model errors stay below 10%.

A natural question is to determine if there is benefit in using the threshold policy for $\bar{q}_{20\%}$ as compared to simply trusting the model every time. To answer this question, in Figure 4 we compare the model errors for the models that are trusted by our threshold policy, to the errors for all the models. We can see that the reduction is significant. Using our policy, $\approx 60\%$ of the errors are within 1% for $\delta = 0.01$, and more importantly, in almost 90% of the cases the errors are smaller than 50%, whereas without the threshold policy we observe a significant fraction of the errors above 50%.

To summarize, one should not expect that the surrogate model can always predict the unknown objective function with high accuracy. However, in our experiments evaluating the quality of the model via cross-validation is helpful in assessing model accuracy: using a simple threshold policy on a measure of model quality, we are able to identify a large number of the inaccurate

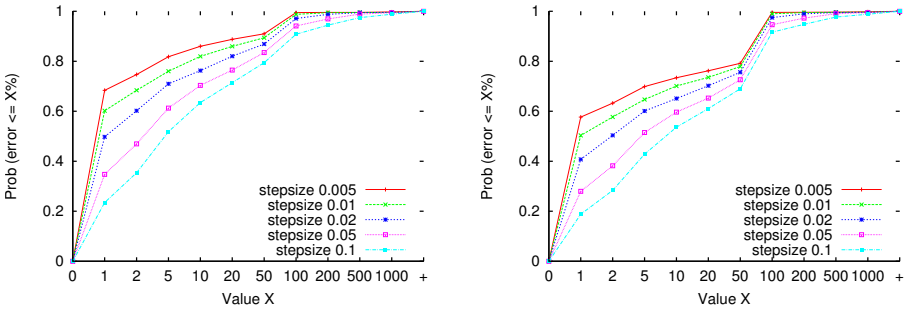

Fig. 4 Empirical cumulative distribution function of the model errors for the threshold policy $\bar{q}_{20\%} \leq 10\%$ (left) and for no policy (right).

surrogate models. For small changes around the optimum, in most cases the model errors are less than 10%, and large errors are rare. In some practical applications, there may be value in using this approach to perform sensitivity analysis as opposed to performing more expensive oracle evaluations.

We investigate one more possible research direction related to model quality. Using the notation of Section 3, it is straightforward to notice that $\lim_{k\to\infty} q_{k,j} = 0$ for all $j = 1, \ldots, k$ on continuous problems, because $\tilde{s}_{k,j}$ agrees with f on a dense subset of Ω. Hence, $\bar{q}_{100\%}$ goes to zero in the limit. We are interested in determining if $\bar{q}_{100\%}$ is strongly correlated with a more traditional measure of model quality, namely $\int_\Omega |s_k(x) - f(x)|\,\mathrm{d}x$, to understand if we can draw conclusions on the global quality of the surrogate model using an inexpensive computation. Conversely, evaluating the integral is very time-consuming, but it can be done for $n \leq 3$. Thus, we apply our algorithm on a set of 11 two- and three-dimensional problem instances, and record $\bar{q}_{100\%}$ and $\int_\Omega |s_k(x) - f(x)|\,\mathrm{d}x$ after every iteration. The Pearson's correlation coefficient between the two samples is 0.599 on average over this set, but unfortunately it is very low (and even negative) on some of the problems. While there is on average a positive correlation between $\bar{q}_{100\%}$ and $\int_\Omega |s_k(x) - f(x)|\,\mathrm{d}x$ across the iterations of the algorithm (which is not surprising since both sequences go to zero as k increases), due to the unpredictability of this measure we decide not to explore this direction further.

6 Conclusions

In this paper we provided an overview of the RBF method for black-box optimization, which is considered one of the best surrogate model based methods for derivative-free optimization. We proposed some modifications of the algorithm with the aim of improving practical performance. Our two main contributions are a methodology to perform automatic model selection using a cross-validation scheme, and an approach to exploit noisy but faster function evaluations. Computational experiments show that these contributions are beneficial in practice, yielding a noticeable reduction in the number of function

evaluations to achieve convergence to within 1% of the global optimum on a collection of test problems taken from the literature.

Our implementation of the algorithm is open-source and available in a library called RBFOpt, which is free for noncommercial academic use. A comparison with other implementations of the RBF algorithm, as well as other methods from the literature, shows that RBFOpt is competitive with the best performing methods available, including commercial software.

Acknowledgements The authors are grateful for the financial support by the SUTD-MIT International Design Center under grant IDG21300102.

References

1. Achterberg, T.: SCIP: Solving constraint integer programs. Mathematical Programming Computation **1**(1), 1–41 (2009)
2. Baudoui, V.: Optimisation robuste multiobjectifs par modèles de substitution. Ph.D. thesis, University of Toulouse Paul Sabatier (2012)
3. Björkman, M., Holmström, K.: Global optimization of costly nonconvex functions using radial basis functions. Optimization and Engineering **1**(4), 373–397 (2000)
4. Bonami, P., Biegler, L., Conn, A., Cornuéjols, G., Grossmann, I., Laird, C., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wächter, A.: An algorithmic framework for convex Mixed Integer Nonlinear Programs. Discrete Optimization **5**, 186–204 (2008)
5. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib — a collection of test models for Mixed-Integer Nonlinear Programming. INFORMS Journal on Computing **15**(1) (2003). URL `http://www.gamsworld.org/minlp/minlplib.htm`
6. Conn, A.R., Scheinberg, K., Toint, P.L.: Recent progress in unconstrained nonlinear optimization without derivatives. Mathematical Programming **79**(1-3), 397–414 (1997). DOI 10.1007/BF02614326
7. Conn, A.R., Scheinberg, K., Vicente, L.N.: Introduction to Derivative-Free Optimization. MPS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics (2009)
8. Conover, W.J.: Practical Nonparametric Statistics, 3rd edition. Wiley (1999)
9. Costa, A., Nannicini, G., Schroepfer, T., Wortmann, T.: Black-box optimization of lighting simulation in architectural design. In: CSD&M Asia 2014. Springer (2014). Accepted for publication
10. Csendes, T., Pál, L., Sendn, J.O.H., Banga, J.R.: The global optimization method revisited. Optimization Letters **2**(4), 445–454 (2008)
11. Dixon, L., Szego, G.: The global optimization problem: an introduction. In: L. Dixon, G. Szego (eds.) Towards Global Optimization, pp. 1–15. North Holland, Amsterdam (1975)
12. Gendreau, M., Potvin, J.Y. (eds.): Handbook of Metaheuristics, 2nd edition. Kluwer, Dordrecht (2010)
13. Glover, F., Kochenberger, G. (eds.): Handbook of Metaheuristics. Kluwer, Dordrecht (2003)
14. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. Journal of Automated Reasoning **24**(1-2), 67–100 (2000). DOI 10.1023/A:1006314320276
15. Gutmann, H.M.: A radial basis function method for global optimization. Journal of Global Optimization **19**, 201–227 (2001). 10.1023/A:1011255519438
16. Hart, W.E., Laird, C., Watson, J.P., Woodruff, D.L.: Pyomo – Optimization Modeling in Python, *Springer Optimization and Its Applications*, vol. 67. Springer (2012)
17. Hart, W.E., Watson, J.P., Woodruff, D.L.: Pyomo: modeling and solving mathematical programs in python. Mathematical Programming Computation **3**(3), 219–260 (2011). DOI 10.1007/s12532-011-0026-8

18. Hemker, T.: Derivative free surrogate optimization for mixed-integer nonlinear black-box problems in engineering. Master's thesis, Technischen Universität Darmstadt (2008)
19. Holmström, K.: An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. Journal of Global Optimization **41**(3), 447–464 (2008)
20. Holmström, K., Quttineh, N.H., Edvall, M.M.: An adaptive radial basis algorithm (arbf) for expensive black-box mixed-integer constrained global optimization. Optimization and Engineering **9**(4), 311–339 (2008)
21. Jakobsson, S., Patriksson, M., Rudholm, J., Wojciechowski, A.: A method for simulation based optimization using radial basis functions. Optimization and Engineering **11**(4), 501–532 (2010)
22. Jones, D., Perttunen, C., Stuckman, B.: Lipschitzian optimization without the lipschitz constant. Journal of Optimization Theory and Applications **79**(1), 157–181 (1993)
23. Jones, D., Schonlau, M., Welch, W.: Efficient global optimization of expensive black-box functions. Journal of Global optimization **13**(4), 455–492 (1998)
24. Kolda, T.G., Lewis, R.M., Torczon, V.J.: Optimization by direct search: new perspectives on some classical and modern methods. SIAM Review **45**(3), 385–482 (2003)
25. Neumaier, A.: Neumaier's collection of test problems for global optimization. URL `http://www.mat.univie.ac.at/~neum/glopt/my_problems.html`. Retrieved in May 2014
26. Powell, M.: Recent research at cambridge on radial basis functions. In: New Developments in Approximation Theory, *International Series of Numerical Mathematics*, vol. 132, pp. 215–232. Birkhauser Verlag, Basel (1999)
27. Regis, R., Shoemaker, C.: Improved strategies for radial basis function methods for global optimization. Journal of Global Optimization **37**, 113–135 (2007). 10.1007/s10898-006-9040-1
28. Regis, R.G., Shoemaker, C.A.: Constrained global optimization of expensive black box functions using radial basis functions. Journal of Global Optimization **31**(1), 153–171 (2005)
29. Regis, R.G., Shoemaker, C.A.: A stochastic radial basis function method for the global optimization of expensive functions. INFORMS Journal on Computing **19**(4), 497–509 (2007). DOI 10.1287/ijoc.1060.0182
30. Regis, R.G., Shoemaker, C.A.: A quasi-multistart framework for global optimization of expensive functions using response surface models. Journal of Global Optimization **56**(4), 1719–1753 (2013)
31. Rios, L.M., Sahinidis, N.V.: Derivative-free optimization: a review of algorithms and comparison of software implementations. Journal of Global Optimization **56**(3), 1247–1293 (2013)
32. Schoen, F.: A wide class of test functions for global optimization. Journal of Global Optimization **3**(2), 133–137 (1993)
33. Törn, A., Žilinskas: Global optimization. Springer (1987)
34. Wächter, A., Biegler, L.T.: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Mathematical Programming **106**(1), 25–57 (2006)

Tables of results

The details of the numerical results obtained with the RBF algorithm are presented in Tables 16-25. For each instance we perform 20 runs of the algorithm, changing the random seed. The algorithm fails if it cannot find a solution having a relative error less than or equal to 1% from the global optimum within 150 function evaluations. The relative error is computed as $|f^* - F^*|/|F^*|$, where f^* is the best solution found and F^* is the global optimum of the problem. In case of $F^* = 0$, the error is computed as $|f^* - F^*|$. The statistics presented on the tables are the number of successful trials out of 20 ("#sol."), the average number of function evaluations, the standard deviation from the mean, and the average relative error after 150 evaluations for those instances where the algorithm does not converge.

Table 16 Results obtained with the default configuration and different function value scaling procedures.

	affine			log			off		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	12	98.65	1.95	20	65.95	0.00	20	37.40	0.00
camel	20	73.85	0.00	20	22.60	0.00	20	40.35	0.00
ex4_1_1	20	19.45	0.00	20	15.00	0.00	20	19.85	0.00
ex4_1_2	19	20.85	2.04	20	9.50	0.00	20	9.60	0.00
ex8_1_1	20	7.65	0.00	20	7.65	0.00	20	7.40	0.00
ex8_1_4	20	27.45	0.00	20	30.75	0.00	20	28.45	0.00
gear	20	7.55	0.00	20	7.70	0.00	20	7.30	0.00
goldsteinprice	5	126.65	6.50	17	66.30	4.48	10	122.70	25.71
hartman3	20	44.45	0.00	20	58.50	0.00	20	36.90	0.00
hartman6	9	127.55	6.27	5	134.05	8.68	10	122.40	6.33
least	0	150.00	142.02	0	150.00	105.73	0	150.00	194.71
nvs04	7	118.00	194.44	0	150.00	194.44	7	115.05	3254.27
nvs06	0	150.00	32.82	0	150.00	34.78	0	150.00	28.55
nvs09	20	14.10	0.00	20	17.25	0.00	20	14.20	0.00
nvs16	4	128.45	1240.00	9	102.90	1192.73	8	108.15	1786.67
perm0_8	0	150.00	685.47	2	145.80	17.93	0	150.00	107.10
perm_6	0	150.00	40989.39	0	150.00	83745.44	0	150.00	101652.21
rbrock	0	150.00	9.45	1	145.05	82.83	1	147.45	16.40
schoen_10_1	0	150.00	56.07	0	150.00	79.04	0	150.00	44.43
schoen_10_2	0	150.00	29.18	0	150.00	61.59	0	150.00	12.93
schoen_6_1	5	147.25	14.79	0	150.00	36.31	3	143.10	14.68
schoen_6_2	5	140.95	22.26	0	150.00	27.60	4	140.55	20.70
shekel10	4	141.10	51.64	1	147.25	47.74	2	146.00	53.94
shekel5	3	145.35	47.72	0	150.00	49.95	1	149.05	45.98
shekel7	3	145.65	48.19	0	150.00	49.27	5	140.45	49.84

Table 17 Results obtained with the "RB" configuration and different function value scaling procedures.

	affine			log			off		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	12	95.25	2.05	20	37.95	0.00	20	31.80	0.00
camel	20	60.40	0.00	20	22.70	0.00	20	40.45	0.00
ex4_1_1	20	18.60	0.00	20	14.60	0.00	20	20.20	0.00
ex4_1_2	20	8.65	0.00	20	8.15	0.00	20	9.65	0.00
ex8_1_1	20	7.35	0.00	20	8.00	0.00	20	7.30	0.00
ex8_1_4	20	28.85	0.00	20	34.75	0.00	20	27.60	0.00
gear	20	7.50	0.00	20	7.70	0.00	20	7.30	0.00
goldsteinprice	6	141.30	3.38	20	39.85	0.00	19	77.40	1.82
hartman3	20	51.15	0.00	19	55.75	2.69	19	50.85	2.48
hartman6	14	105.35	4.57	14	113.80	6.03	14	102.95	3.24
least	0	150.00	301.34	0	150.00	75.21	0	150.00	237.96
nvs04	16	73.10	194.44	0	150.00	194.44	13	96.10	194.44
nvs06	2	141.65	14.16	3	136.70	8.53	2	142.10	15.41
nvs09	20	14.10	0.00	20	19.30	0.00	20	14.25	0.00
nvs16	10	101.60	1397.33	10	114.25	1920.00	14	104.55	551.11
perm0_8	0	150.00	341.94	3	142.80	9.56	0	150.00	222.53
perm_6	0	150.00	57361.46 28	0	150.00	41399.39	0	150.00	24394.28
rbrock	1	147.10	10.33	1	148.20	49.11	5	134.90	11.63
schoen_10_1	8	145.45	19.70	0	150.00	73.94	9	144.45	15.16
schoen_10_2	14	131.25	1.54	0	150.00	64.55	12	143.90	1.80
schoen_6_1	19	92.90	1.09	0	150.00	29.91	20	91.00	0.00
schoen_6_2	17	91.95	33.63	0	150.00	35.52	17	89.45	73.85
shekel10	6	132.45	21.09	2	148.10	35.21	4	137.20	21.04
shekel5	5	132.75	32.06	2	145.30	38.04	2	143.90	33.13
shekel7	6	133.90	25.50	0	150.00	27.06	8	123.25	29.12

Table 18 Results obtained with the "RBL" configuration and different function value scaling procedures.

	affine			log			off		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	14	97.40	1.84	20	38.30	0.00	20	32.80	0.00
camel	18	67.00	1.16	20	22.00	0.00	20	38.80	0.00
ex4_1_1	20	14.85	0.00	20	14.60	0.00	20	16.15	0.00
ex4_1_2	20	8.05	0.00	20	8.15	0.00	20	9.15	0.00
ex8_1_1	20	7.35	0.00	20	7.80	0.00	20	7.30	0.00
ex8_1_4	20	28.00	0.00	20	37.40	0.00	20	29.80	0.00
gear	20	7.50	0.00	20	7.70	0.00	20	7.30	0.00
goldsteinprice	5	127.75	2.93	20	39.75	0.00	18	79.55	4.35
hartman3	20	46.75	0.00	18	61.30	2.92	20	48.45	0.00
hartman6	18	98.55	6.41	15	108.50	4.70	17	98.40	5.48
least	0	150.00	266.94	0	150.00	116.48	0	150.00	241.37
nvs04	15	78.75	194.44	0	150.00	194.44	12	101.90	194.44
nvs06	4	135.65	15.49	6	127.65	9.17	3	142.60	16.15
nvs09	20	14.10	0.00	20	18.60	0.00	20	14.25	0.00
nvs16	13	92.75	1691.43	10	114.35	1920.00	13	110.40	883.81
perm0_8	0	150.00	360.79	4	146.25	30.87	1	148.80	221.15
perm_6	0	150.00	104435.23	0	150.00	34643.75	0	150.00	38174.83
rbrock	1	146.35	17.84	1	147.35	62.05	5	142.45	13.90
schoen_10_1	9	144.90	20.76	0	150.00	69.76	15	135.95	25.98
schoen_10_2	15	124.65	2.01	0	150.00	61.99	14	128.60	1.86
schoen_6_1	18	92.50	1.34	0	150.00	27.31	20	83.45	0.00
schoen_6_2	16	92.25	53.57	0	150.00	26.54	16	88.40	97.25
shekel10	6	130.95	34.27	3	143.40	35.85	3	142.20	27.67
shekel5	6	134.10	34.28	1	149.35	33.74	7	128.55	41.90
shekel7	6	134.85	39.70	1	146.85	30.37	5	133.80	21.43

Table 19 Results obtained with the "R", "B" and "L" configurations and "off" scaling procedure.

	R			B			L		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	20	38.70	0.00	20	31.80	0.00	20	35.25	0.00
camel	20	42.50	0.00	20	38.50	0.00	20	39.50	0.00
ex4_1_1	20	19.85	0.00	20	20.20	0.00	20	18.15	0.00
ex4_1_2	20	9.60	0.00	20	9.65	0.00	20	9.60	0.00
ex8_1_1	20	7.40	0.00	20	7.30	0.00	20	7.40	0.00
ex8_1_4	20	28.45	0.00	20	27.60	0.00	20	27.40	0.00
gear	20	7.30	0.00	20	7.30	0.00	20	7.30	0.00
goldsteinprice	11	123.80	31.48	20	71.55	0.00	12	115.20	11.49
hartman3	20	35.90	0.00	20	45.85	0.00	20	40.20	0.00
hartman6	9	129.95	5.68	11	106.70	4.28	10	118.00	6.60
least	0	150.00	264.34	0	150.00	200.53	0	150.00	204.31
nvs04	4	131.10	194.44	8	105.90	194.44	5	125.55	194.44
nvs06	0	150.00	28.69	11	117.70	9.32	0	150.00	27.83
nvs09	20	14.20	0.00	20	14.25	0.00	20	14.20	0.00
nvs16	9	106.55	775.76	9	108.70	1221.82	8	107.40	1484.44
perm0_8	0	150.00	170.01	0	150.00	183.08	1	147.20	176.23
perm_6	0	150.00	116469.28	0	150.00	20224.70	0	150.00	73415.96
rbrock	1	147.45	35.22	7	132.00	9.16	3	145.05	19.10
schoen_10_1	0	150.00	43.80	9	144.45	15.16	0	150.00	52.80
schoen_10_2	0	150.00	14.77	14	143.05	1.73	0	150.00	18.10
schoen_6_1	3	143.10	15.69	20	91.00	0.00	6	136.05	16.94
schoen_6_2	4	142.00	18.69	17	89.45	110.28	6	130.30	22.65
shekel10	2	147.85	47.83	10	129.65	36.96	4	144.35	59.24
shekel5	0	150.00	39.48	5	142.25	40.78	5	138.50	60.54
shekel7	3	144.15	43.61	11	121.55	31.48	2	144.25	45.92

Table 20 Results obtained with the "RL" and "BL" configurations and "off" scaling procedure.

	RL			BL		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	20	35.25	0.00	20	32.80	0.00
camel	20	39.50	0.00	20	36.90	0.00
ex4_1_1	20	18.15	0.00	20	16.15	0.00
ex4_1_2	20	9.60	0.00	20	9.15	0.00
ex8_1_1	20	7.40	0.00	20	7.30	0.00
ex8_1_4	20	27.40	0.00	20	29.80	0.00
gear	20	7.30	0.00	20	7.30	0.00
goldsteinprice	9	125.05	6.39	18	77.70	3.74
hartman3	20	41.35	0.00	20	46.05	0.00
hartman6	11	116.90	3.99	11	104.15	4.30
least	0	150.00	271.77	0	150.00	213.07
nvs04	5	138.50	194.44	7	112.90	194.44
nvs06	0	150.00	24.66	6	133.95	8.58
nvs09	20	14.20	0.00	20	14.25	0.00
nvs16	11	104.60	675.56	8	113.55	1013.33
perm0_8	1	147.20	201.63	1	148.80	147.19
perm_6	0	150.00	163072.95	0	150.00	24567.01
rbrock	2	148.85	35.28	7	135.20	10.72
schoen_10_1	0	150.00	52.96	15	135.95	25.98
schoen_10_2	0	150.00	19.02	16	127.15	2.25
schoen_6_1	6	136.05	17.90	20	83.45	0.00
schoen_6_2	6	130.30	17.39	16	88.40	110.26
shekel10	2	145.70	49.02	8	136.15	47.75
shekel5	5	138.50	56.27	8	127.75	46.82
shekel7	2	144.25	41.60	9	127.95	29.08

Table 21 Results obtained with the "RBL" configuration using a noisy oracle with 10% or 20% relative error.

	RBL n10%			RBL n20%		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	20	15.27	0.00	20	17.05	0.00
camel	20	16.67	0.00	20	16.83	0.00
ex4_1_1	20	8.62	0.00	20	9.70	0.00
ex4_1_2	20	4.78	0.00	20	5.38	0.00
ex8_1_1	20	4.63	0.00	20	6.17	0.00
ex8_1_4	20	9.85	0.00	20	10.37	0.00
gear	20	3.92	0.00	20	3.87	0.00
goldsteinprice	20	36.73	0.00	20	36.87	0.00
hartman3	20	28.25	0.00	20	29.32	0.00
hartman6	17	67.15	4.96	16	78.55	3.77
least	0	150.00	213.58	0	150.00	244.60
nvs04	13	70.72	205.62	14	68.48	214.09
nvs06	1	146.82	15.95	4	128.67	15.73
nvs09	20	5.67	0.00	20	7.35	0.00
nvs16	18	42.58	1114.05	16	54.25	1050.43
perm0_8	0	150.00	193.64	0	150.00	138.00
perm_6	0	150.00	46107.10	0	150.00	35623.78
rbrock	4	131.08	15.59	4	126.13	19.37
schoen_10_1	3	142.47	33.62	1	147.15	91.73
schoen_10_2	7	128.25	6.04	2	145.07	19.94
schoen_6_1	15	90.62	8.74	15	95.58	21.51
schoen_6_2	14	84.08	51.15	15	96.22	8.54
shekel10	6	131.88	31.68	3	142.85	38.53
shekel5	1	147.13	43.51	2	145.18	44.30
shekel7	1	145.50	36.54	1	147.38	43.56

Table 22 Results obtained with automatic model selection for the "RBL" configuration of the algorithm.

	RBL			RBL auto		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	20	32.80	0.00	20	30.50	0.00
camel	20	38.80	0.00	20	34.40	0.00
ex4_1_1	20	16.15	0.00	20	14.15	0.00
ex4_1_2	20	9.15	0.00	20	8.60	0.00
ex8_1_1	20	7.30	0.00	20	7.30	0.00
ex8_1_4	20	29.80	0.00	20	25.40	0.00
gear	20	7.30	0.00	20	7.30	0.00
goldsteinprice	18	79.55	4.35	20	52.50	0.00
hartman3	20	48.45	0.00	20	44.50	0.00
hartman6	17	98.40	5.48	17	101.40	5.08
least	0	150.00	241.37	0	150.00	204.73
nvs04	12	101.90	194.44	19	64.40	194.44
nvs06	3	142.60	16.15	0	150.00	13.29
nvs09	20	14.25	0.00	20	14.25	0.00
nvs16	13	110.40	883.81	20	48.75	0.00
perm0_8	1	148.80	221.15	0	150.00	147.20
perm_6	0	150.00	38174.83	0	150.00	44134.67
rbrock	5	142.45	13.90	5	135.70	10.81
schoen_10_1	15	135.95	25.98	11	139.05	28.80
schoen_10_2	14	128.60	1.86	14	132.50	1.55
schoen_6_1	20	83.45	0.00	18	101.00	1.78
schoen_6_2	16	88.40	97.25	16	102.30	32.67
shekel10	3	142.20	27.67	13	106.75	60.11
shekel5	7	128.55	41.90	7	125.75	51.73
shekel7	5	133.80	21.43	5	136.50	47.04

Table 23 Results obtained with automatic model selection for the "RBL" configuration of the algorithm and a noisy oracle with 10% relative error.

	RBL n10%			RBL n10% auto		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	20	15.27	0.00	20	18.20	0.00
camel	20	16.67	0.00	20	24.20	0.00
ex4_1_1	20	8.62	0.00	20	8.72	0.00
ex4_1_2	20	4.78	0.00	20	4.68	0.00
ex8_1_1	20	4.63	0.00	20	4.63	0.00
ex8_1_4	20	9.85	0.00	20	11.25	0.00
gear	20	3.92	0.00	20	3.92	0.00
goldsteinprice	20	36.73	0.00	14	66.07	17.49
hartman3	20	28.25	0.00	17	41.80	4.63
hartman6	17	67.15	4.96	17	62.32	5.56
least	0	150.00	213.58	0	150.00	177.53
nvs04	13	70.72	205.62	18	40.33	212.61
nvs06	1	146.82	15.95	5	121.13	15.39
nvs09	20	5.67	0.00	20	5.67	0.00
nvs16	18	42.58	1114.05	20	19.92	0.00
perm0_8	0	150.00	193.64	0	150.00	216.22
perm_6	0	150.00	46107.10	0	150.00	21914.07
rbrock	4	131.08	15.59	3	135.82	13.67
schoen_10_1	3	142.47	33.62	3	142.48	38.86
schoen_10_2	7	128.25	6.04	14	100.47	4.44
schoen_6_1	15	90.62	8.74	19	59.82	2.48
schoen_6_2	14	84.08	51.15	17	60.83	44.77
shekel10	6	131.88	31.68	5	129.25	38.48
shekel5	1	147.13	43.51	5	130.38	42.01
shekel7	1	145.50	36.54	3	137.23	39.74

Table 24 Results obtained with automatic model selection for the "RBL" configuration of the algorithm and a noisy oracle with 20% relative error.

	RBL n20%			RBL n20% auto		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	20	17.05	0.00	20	17.78	0.00
camel	20	16.83	0.00	20	24.53	0.00
ex4_1_1	20	9.70	0.00	20	9.17	0.00
ex4_1_2	20	5.38	0.00	20	5.28	0.00
ex8_1_1	20	6.17	0.00	20	6.30	0.00
ex8_1_4	20	10.37	0.00	20	11.82	0.00
gear	20	3.87	0.00	20	3.87	0.00
goldsteinprice	20	36.87	0.00	14	81.55	8.04
hartman3	20	29.32	0.00	20	33.72	0.00
hartman6	16	78.55	3.77	17	80.85	2.82
least	0	150.00	244.60	0	150.00	187.66
nvs04	14	68.48	214.09	17	41.35	227.86
nvs06	4	128.67	15.73	2	141.88	14.61
nvs09	20	7.35	0.00	20	7.08	0.00
nvs16	16	54.25	1050.43	20	24.33	0.00
perm0_8	0	150.00	138.00	0	150.00	233.28
perm_6	0	150.00	35623.78	0	150.00	46667.82
rbrock	4	126.13	19.37	3	136.97	11.56
schoen_10_1	1	147.15	91.73	1	147.75	92.31
schoen_10_2	2	145.07	19.94	4	140.13	27.72
schoen_6_1	15	95.58	21.51	17	90.15	9.95
schoen_6_2	15	96.22	8.54	15	87.05	38.02
shekel10	3	142.85	38.53	4	139.50	40.64
shekel5	2	145.18	44.30	4	139.05	50.50
shekel7	1	147.38	43.56	5	132.70	54.83

Table 25 Results obtained with the "RBL" configuration and "off" scaling procedures and a noisy oracle with 10% relative error, but relative error estimate ϵ_r of the algorithm set at 20%, and with 20% relative error but relative error estimate ϵ_r set at 30%.

	RBL n10% ($\epsilon_r = 20\%$)			RBL n20% ($\epsilon_r = 30\%$)		
Instance	#sol.	avg. eval.	error	#sol.	avg. eval.	error
branin	20	17.70	0.00	20	17.40	0.00
camel	20	16.22	0.00	20	20.32	0.00
ex4_1_1	20	9.37	0.00	20	9.98	0.00
ex4_1_2	20	5.10	0.00	20	5.18	0.00
ex8_1_1	20	5.02	0.00	20	5.78	0.00
ex8_1_4	20	10.57	0.00	20	10.32	0.00
gear	20	4.05	0.00	20	3.75	0.00
goldsteinprice	18	40.78	1.59	18	41.83	14.63
hartman3	20	28.42	0.00	20	24.02	0.00
hartman6	19	63.47	6.49	15	78.87	3.81
least	0	150.00	210.17	1	145.67	231.35
nvs04	18	50.45	239.58	13	89.67	243.78
nvs06	1	146.55	16.24	5	126.83	6.36
nvs09	20	6.40	0.00	20	6.05	0.00
nvs16	20	30.52	0.00	20	19.60	0.00
perm0_8	1	145.82	175.63	1	148.12	115.15
perm_6	0	150.00	21109.16	0	150.00	68840.79
rbrock	3	138.22	14.23	4	130.48	28.16
schoen_10_1	2	145.35	52.51	1	147.85	81.46
schoen_10_2	5	136.15	21.75	3	141.90	27.15
schoen_6_1	14	96.63	17.81	6	131.88	27.13
schoen_6_2	14	89.97	38.24	14	97.67	24.28
shekel10	1	146.55	53.14	6	133.03	35.57
shekel5	1	147.50	52.46	2	145.50	41.92
shekel7	4	137.03	41.23	3	139.68	56.29