

# SNBOFIT – Stable Noisy Optimization by Branch and Fit

WALTRAUD HUYER and ARNOLD NEUMAIER

Universität Wien

---

The software package SNBOFIT for bound constrained (and soft constrained) noisy optimization of an expensive objective function is described. It combines global and local search by branching and local fits. The program is made robust and flexible for practical use by allowing for hidden constraints, batch function evaluations, change of search regions, etc.

Categories and Subject Descriptors: G.1.6 [**Optimization**]: Global optimization; Constrained optimization

General Terms: Algorithms

Additional Key Words and Phrases: branch and bound, derivative-free, surrogate model, parallel evaluation, hidden constraints, expensive function values, noisy function values, soft constraints

---

## 1. INTRODUCTION

SNBOFIT (*stable noisy optimization by branch and fit*) is a MATLAB package designed for selecting continuous parameter settings for simulations or experiments, performed with the goal of optimizing some user-specified criterion. Specifically, we consider the optimization problem

$$\begin{aligned} \min f(x) \\ \text{s.t. } x \in [u, v], \end{aligned}$$

where we use interval notation for *boxes*

$$[u, v] := \{x \in \mathbb{R}^n \mid u_i \leq x_i \leq v_i, i = 1, \dots, n\},$$

with  $u, v \in \mathbb{R}^n$  and  $u_i < v_i$  for  $i = 1, \dots, n$ , i.e.,  $[u, v]$  is bounded with nonempty interior. A box  $[u', v']$  with  $[u', v'] \subseteq [u, v]$  is called a *subbox* of  $[u, v]$ . Moreover, we assume that  $f$  is a function  $f : D \rightarrow \mathbb{R}$ , where  $D$  is a subset of  $\mathbb{R}^n$  containing  $[u, v]$ . We will call the process of obtaining an approximate function value  $f(x)$  an *evaluation* at the point  $x$  (typically by simulation or measurement).

While there are many software packages that can handle such problems, they usually cannot cope well with one or more of the following difficulties arising in practice:

---

Authors' address: Fakultät für Mathematik, Universität Wien, Nordbergstraße 15, A-1090 Wien, Austria.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0098-3500/20YY/1200-0001 \$5.00

- (I1) the function values are expensive (for example, obtained by performing complex experiments or simulations);
- (I2) instead of a function value requested at a point  $x$ , only a function value at some nearby point  $\tilde{x}$  is returned;
- (I3) the function values are noisy (inaccurate due to experimental errors or to low precision calculations);
- (I4) the objective function may have several local minimizers;
- (I5) no gradients are available;
- (I6) the problem may contain hidden constraints, i.e., a requested function value may turn out not to be obtainable;
- (I7) the problem may involve additional soft constraints, i.e., constraints which need not be satisfied accurately;
- (I8) the user wants to evaluate at several points simultaneously, e.g., by making several simultaneous experiments or parallel simulations;
- (I9) function values may be obtained so infrequently or in a heterogeneous environment that, between obtaining function values, the computer hosting the software is used otherwise or even switched off;
- (I10) the objective function or the search region may change during optimization, e.g., because users inspect the data obtained so far and this suggests to them a more realistic or more promising goal.

Many different algorithms (see, e.g., the survey [Powell 1998]) have been proposed for unconstrained or bound constrained optimization when first derivatives are not available. Conn et al. [1997] distinguish two classes of derivative-free optimization algorithms. *Sampling methods* or direct search methods proceed by generating a sequence of points; pure sampling methods tend to require rather many function values in practice. *Modeling methods* try to approximate the function over a region by some model function and the much cheaper surrogate problem of minimizing the model function is solved. Not all algorithms are equally suited for the application to noisy objective functions; for example, it would not be meaningful to use algorithms that interpolate the function at points that are too close together.

A method called DACE (*design and analysis of computer experiments*) [Sacks et al. 1989; Welch et al. 1992] deals with finding a surrogate function for a function generated by computer experiments, which consist of a number of runs of a computer code with various inputs. These codes are typically expensive to run and the output is deterministic, i.e., rerunning the code with the same input gives identical results, but the output is distorted by high-frequency, low-amplitude oscillations. The lack of random error makes computer experiments different from physical experiments. The problem of fitting a response surface model to the observed data consists of the design problem, which is the problem of the choice of points where data should be collected, and the analysis problem of how the data should be used to obtain a good fit. The output of the computer code is modeled as a realization of a stochastic process; the method of analysis for such models is known as kriging in the mathematical geostatistics literature. For the choice of the sample points, “space-filling” designs such as orthogonal array-based Latin hy-

percubes [Tang 1993] are important; see [McKay et al. 1979] for a comparison of random sampling, stratified sampling and Latin hypercube sampling.

The SPACE algorithm (*stochastic process analysis of computer experiments*) by Schonlau [Schonlau 1997; 2001; Schonlau et al. 1998] (see also [Jones et al. 1998] for the similar algorithm EGO) uses the DACE approach resp. Bayesian global optimization in order to find the global optimum of a computer model. The function to be optimized is modeled as a Gaussian stochastic process, where all previous function evaluations are used to fit the statistical model. The algorithm contains a parameter that controls the balance between local and global search components of the optimization. The minimization is done in stages, i.e., rather than only one point, the algorithm samples a specified number of points at a time. Moreover, a method for dealing with nonlinear inequality constraints from additional response variables is proposed.

The authors of [Booker et al. 1999; Trosset and Torczon 1997] consider traditional iterative methods and DACE as opposite ends of the spectrum of derivative-free optimization methods. They combine pattern search methods and DACE for the bound constrained optimization of an expensive objective function. Pattern search algorithms are iterative algorithms that produce a sequence of points from an initial point, where the search for the next point is restricted to a grid containing the current iterate and the grid is modified as optimization progresses. The kriging approach is used to construct a sequence of surrogate models for the objective functions, which are used to guide a grid search for a minimizer. Moreover, Booker et al. [1999] also consider the case that the routines evaluating the objective function may fail to return  $f(x)$  even for feasible  $x$ , i.e., the case of hidden constraints.

Jones [2001] presents a taxonomy of existing approaches for using response surfaces for global optimization. Seven methods are compared and illustrated with numerical examples that show their advantages and disadvantages.

Elster and Neumaier [1995] develop an algorithm for the minimization of a low-dimensional, noisy function with bound constraints, where no knowledge about the statistical properties of the noise is assumed, i.e., it may be deterministic or stochastic (but must be bounded). The algorithm is based on the use of quadratic models minimized over adaptively defined trust regions together with the restriction of the evaluation points to a sequence of nested grids.

Anderson and Ferris [2001] consider the unconstrained optimization of a function subject to random noise, where it is assumed that averaging repeated observations at the same point leads to a better estimate of the objective function value. They develop a simplicial direct search method including a stochastic element and prove convergence under certain assumptions on the noise; however, their proof of convergence does not work in the absence of noise.

Carter et al. [2001] deal with algorithms for bound constrained noisy problems in gas transmission pipeline optimization. They consider the DIRECT algorithm of Jones et al. [1993], which proceeds by repeated subdivision of the feasible region, implicit filtering, a sampling method designed for problems that are low-amplitude, high-frequency perturbations of smooth problems, and a new hybrid of implicit filtering and DIRECT, which attempts to combine the best features of the two other algorithms. In addition to the bound constraints, the objective function may fail

to return a value for some feasible points. The traditional approach assigns a large value to the objective function when it cannot be evaluated, which results in slow convergence if the solution lies on a constraint boundary. In [Carter et al. 2001] a modification is proposed; the function value is derived from nearby feasible points rather than assigning an arbitrary value. In [Choi and Kelley 2000], implicit filtering is coupled with the BFGS quasi-Newton update.

The UOBYQA (*unconstrained optimization by quadratical approximation*) algorithm by Powell [2002] for unconstrained optimization takes account of the curvature of the objective function by forming quadratic models by Lagrange interpolation. A typical iteration of the algorithm generates a new point either by minimizing the quadratic model subject to a trust region bound, or by a procedure that should improve the accuracy of the model. The evaluations are made in a way that reduces the influence of noise and therefore the algorithm is suitable for noisy objective functions. Vanden Berghen and Bersini [2005] present a parallel, constrained extension of UOBYQA and call it CONDOR (*constrained, non-linear, direct, parallel optimization using the trust region method*).

None of the algorithms mentioned are robustly dealing with the difficulties mentioned, at least not in the implementations available to us. The goal of the present paper is to describe a new algorithm that addresses all the above points. In Section 2, the basic setup of SNOBFIT is presented; for details we refer to Section 6. In Sections 3 to 5, we describe some important ingredients of the algorithm. In Section 7, the convergence of the algorithm is established, and in Section 8 a new penalty function is proposed to handle soft constraints with some quality guarantees. Finally, in Section 9 numerical results are presented.

## 2. BASIC SETUP OF SNOBFIT

The SNOBFIT package described in this paper solves the optimization problem

$$\begin{aligned} \min f(x) \\ \text{s.t. } x \in [u, v], \end{aligned} \tag{1}$$

given a method that (in between calls to SNOBFIT) evaluates  $f$  at or near a given number of points, or returns NaN to signal that a requested function value does not exist. Additional soft constraints (not represented in (1)) are handled in SNOBFIT by a novel penalty approach. In the following we call a *job* the attempt to solve a single problem of the form (1) with SNOBFIT. The function (but not necessarily the search region  $[u, v]$ ) and some of the tuning parameters of SNOBFIT stay the same in a job. A job consists of several *calls* to SNOBFIT. We use the notion *initial call* for the first call of a job and *continuation call* for any later call of SNOBFIT on the same job.

Based on the function values already available, the algorithm builds internally around each point local models of the function to minimize, and returns in each step a number of points whose evaluation is likely to improve these models or is expected to give better function values. Since the total number of function evaluations is kept low, no guarantees can be given that a global minimum is located; convergence is proved in Section 7 only if the number of function evaluations tends to infinity. Nevertheless, the numerical results in Section 9 show that in low dimensions the algorithm often converges in very few function evaluations compared with previous

methods. In SNOBFIT, the best function value is retained even if it was overly optimistic (due to noise); if this is undesirable, a statistical postprocessing may be advisable.

The algorithm is safeguarded in many ways to deal with the issues (I1)–(I10) mentioned in the previous section, either in the basic SNOBFIT algorithm itself, or in one of the drivers.

- SNOBFIT does not need gradients (issue (I5)); instead, these are estimated from surrogate functions.
- The surrogate functions are not interpolated but fitted to a stochastic (linear or quadratic) model to take noisy function values (issue (I3)) into account; see Section 3.
- At the end of each call the intermediate results are stored in a working file (settling issue (I9)).
- The optimization proceeds through a number of calls to SNOBFIT, which process the points and function values evaluated between the calls, and produce a user-specified number of new recommended evaluation points, thus allowing parallel function evaluation (issue (I8)); a driver program `snobdriver.m`, which is part of the package, shows how to do this.
- If no points and function values are provided in an initial call, SNOBFIT generates a randomized space-filling design to enhance the chance that the surrogate model is globally valid; see the end of Section 4.
- For fast local convergence (needed because of expensive function values, issue (I1)), SNOBFIT handles local search from the best point with a full quadratic model. Since this involves  $O(n^6)$  operations for the linear algebra, this limits the number of variables that can be handled without excessive overhead. (Update methods based on already estimated gradients would produce cheaper quadratic models. If gradients were exact, this would give finite termination after at most  $n$  steps on noise-free quadratics. However, this property and with it the super-linear convergence is lost in the present context since the estimated gradients are inaccurate. Our use of quadratic models ensures the finite termination on noise-free quadratic functions.)
- SNOBFIT combines local and global search and allows the user to control which of both should be emphasized, by classifying the recommended evaluation points into five classes with different scope; see Section 4.
- To account for possible multiple local minima (issue (I4)), SNOBFIT maintains a global view of the optimization landscape by recursive partitioning of the search box; see Section 5.
- For the global search, SNOBFIT searches simultaneously in several promising sub-regions, determined by finding so-called local points, and predicts their quality by cheaper models based on fitting the function values at so-called safeguarded nearest neighbors; see Section 3.
- To account for issue (I10) (resetting goals), the user may, after each call, change the search region and some of the parameters, and at the end of Section 6 we give a recipe how to proceed if a change of the objective function or the accuracy

- is desired. Using points outside the box bounds  $[u, v]$  as input will result in an extension of the search region (see Section 6, Step 1).
- SNOBFIT allows for hidden constraints and assigns to such points a fictitious function value based on the function values of nearby points with finite function value (issue (I6)); see Section 3.
  - Soft constraints (issue (I7)) are handled by a new penalty approach which guarantees some sort of optimality; see the second driver program `snobsoftdriver.m` and Section 8.
  - The user may evaluate the function at other points than the ones suggested by SNOBFIT (issue (I2)); however, if done regularly in an unreasonable way, this may destroy the convergence properties.
  - It is permitted to use a point for which the function has already been evaluated again as input with a different function value; in that case an averaged function value is computed (see Section 6, Step 2) – this might be useful in case of very noisy functions (issue (I3)).

The fact that so many different issues must be addressed necessitates a large number of details and decisions that must be taken. While each detail can be motivated, there are often a number of possible alternatives to achieve some effect, and since these effects influence each other in a complicated way, the best decisions can only be determined by experiment. We weeded out a number of alternatives tried and found less efficient, and report only on the choices that survived in the distributed package.

The MATLAB code of the version of SNOBFIT described in this paper is available on the web at <http://www.mat.univie.ac.at/~neum/software/snobfit/>. To be able to use SNOBFIT, the user should know the overall structure of the package, detailed in the remainder of this section, and then be able to use (and modify, if necessary) the drivers `snobdriver.m` for bound constraints and `snobsoftdriver.m` for general soft constraints.

In these drivers, an initial call with an empty list of points and function values is made. The same number of points is generated by each call to SNOBFIT, and the function is evaluated at the points suggested by SNOBFIT. The function values are obtained by calls to a MATLAB function; the user can adapt it to reading the function values from a file, for example. In contrast to the drivers for the test problems described in Sections 9.1 and 9.3, where stopping criteria using the known function value  $f_{\text{glob}}$  at the global optimizer are used, these drivers stop if the best function value has not been improved in a certain number of calls to SNOBFIT.

In the following we denote with  $X$  the set of points for which the objective function has already been evaluated at some stage of SNOBFIT. One call to SNOBFIT roughly proceeds as follows; a detailed description of the input and output parameters will be given later in this section. In each call to SNOBFIT, a (possibly empty) set of points and corresponding function values are put into the program, together with some other parameters. Then the algorithm generates (if possible) the requested number  $n_{\text{req}}$  of new evaluation points; these points and their function values should preferably be used as input for the following call to SNOBFIT. The suggested evaluation points belong to five classes indicating how such a point has

been generated, in particular, whether it has been generated by a local or a global aspect of the algorithm. The points of classes 1 to 3 represent the more local aspect of the algorithm and are generated with the aid of local linear or quadratic models at positions where good function values are expected. The points of classes 4 and 5 represent the more global aspect of the optimization, and are generated in large unexplored regions. These points are reported together with a model function value computed from the appropriate local model.

In addition to the search region  $[u, v]$ , which is the box that will be partitioned by the algorithm, we consider a box  $[u', v'] \subset \mathbb{R}^n$  in which the points generated by SNOBFIT should be contained. The vectors  $u'$  and  $v'$  are input variables for each call to SNOBFIT; their most usual application occurs when the user wants to explore a subbox of  $[u, v]$  more thoroughly (which almost amounts to a restriction of the search region), but it is also possible to choose  $u'$  and  $v'$  such that  $[u', v'] \not\subset [u, v]$  – in that case  $[u, v]$  is extended such that it contains  $[u', v']$ . We use the notion *box tree* for a certain kind of partitioning structure of a box  $[u, v]$ . The box tree corresponds to a partition of  $[u, v]$  into subboxes each containing a point. Each node of the box tree consists of a (nonempty) subbox  $[\underline{x}, \bar{x}]$  of  $[u, v]$  and a point  $x$  in  $[\underline{x}, \bar{x}]$ . Apart from the leaves, each node of a box tree with corresponding box  $[\underline{x}, \bar{x}]$  and point  $x$  has two children containing two subboxes of  $[\underline{x}, \bar{x}]$  obtained by splitting  $[\underline{x}, \bar{x}]$  along a splitting plane perpendicular to one of the axes. The point  $x$  belongs to one of the child nodes; the other child node is assigned a new point  $x'$ .

In each call to SNOBFIT, a (possibly empty) list  $x^j, j = 1, \dots, J$ , of points (which do not have to be in  $[u, v]$ , but input of points  $x^j \notin [u, v]$  will result in an extension of the search region), their corresponding function values  $f_j$ , the uncertainties of the function values  $\Delta f_j$ , a natural number  $n_{\text{req}}$  (the desired number of points to be generated by SNOBFIT), two  $n$ -vectors  $u'$  and  $v'$ ,  $u' \leq v'$ , and a real number  $p \in [0, 1]$  are fed into the program. For measured function values, the values  $\Delta f_j$  should reflect the expected accuracy; for computed function values, they should reflect the estimated size of rounding errors and/or discretization errors. Setting  $\Delta f_j$  to a nonpositive value indicates that the user does not know what to set  $\Delta f_j$  to, and the program resets  $\Delta f_j$  to  $\sqrt{\varepsilon}$ , where  $\varepsilon$  is the machine precision.

If the user has not been able to obtain a function value for  $x^j$ ,  $f_j$  should be set to NaN (“not a number”), whereas  $x^j$  should be deleted if a function evaluation has not even been tried. Internally, the NaNs are replaced (when meaningful) by appropriate function values; see Section 3.

When a job is started (i.e., in the case of an initial call to SNOBFIT), a “resolution vector”  $\Delta x > 0$  is needed as additional input. It is assumed that the  $i$ th coordinate is measured in units  $\Delta x_i$ , and this resolution vector is only set in an initial call and stays the same during the whole job. The effect of the resolution vector is that the algorithm only suggests evaluation points whose  $i$ th coordinate is an integral multiple of  $\Delta x_i$ .

The program then returns  $n_{\text{req}}$  (occasionally fewer; see the end of Section 4) suggested evaluation points in the box  $[u', v']$ , their class (as defined in Section 4), the model function values at these points, the current best point and the current best function value. The idea of the algorithm is that these points and their function values are used as input for the next call to SNOBFIT, but the user may feed instead

other points or even an old point with a newly evaluated function value into the program. For example, some suggested evaluations may not have been successful, or the experiment does not allow to locate the position precisely; the position obtained differs from the intended position but can be measured with a precision higher than the error made. The reported model function values are supposed to help the user to decide whether it is worthwhile to evaluate the function at a suggested point.

In the continuation calls to SNOBFIT,  $\Delta x$ , a list of “old” points, their function values and a box tree are loaded from a working file created at the end of the previous call to SNOBFIT.

### 3. SAFEGUARDED NEAREST NEIGHBORS, HIDDEN CONSTRAINTS, AND LOCAL MODELS

When sufficiently many points with at least two different function values  $\neq \text{NaN}$  are available, local surrogate models are created by linear least squares fits. (In the case that all function values are the same or NaNs, fitting a nontrivial local model is impossible, and nothing is done.) In the presence of noise, fitting reliable linear models near some point requires the use of a few, say  $\Delta n$  more data points than parameters. The choice  $\Delta n = 5$  used in the implementation is a compromise between choosing  $\Delta n$  too small (resulting in an only slightly overdetermined fit) and too large (requiring much computational effort).

If the total number  $N := |X|$  of points is large enough for a fit,  $N \geq n + \Delta n + 1$ , the  $n + \Delta n$  *safeguarded nearest neighbors* of each point  $x = (x_1, \dots, x_n)^T \in X$  are determined as follows. Starting with an initially empty list of safeguarded nearest neighbors, we pick, for  $i = 1, \dots, n$ , a point in  $X$  closest to  $x$  among the points  $y = (y_1, \dots, y_n)^T$  not yet in the list satisfying  $|y_i - x_i| \geq \Delta x_i$ , if such a point exists. Except in the degenerate case that some coordinate has not been varied at all, this safeguard ensures that, for each coordinate  $i$ , there is at least one point in the list of safeguarded nearest neighbors whose  $i$ th coordinate differs from that of  $x$  by at least  $\Delta x_i$ ; in particular, the safeguarded nearest neighbors will not all be on one line. This list of up to  $n$  safeguarded nearest neighbors is filled up step by step by the point in  $X \setminus \{x\}$  closest to  $x$  not yet in the list, until  $n + \Delta n$  points are in the list. This is implemented efficiently, using that for each new point, only the distances to the previous points are needed, and these can be found in  $O(nN^2)$  operations over the whole history. For the range of values of  $n$  and  $N$  for which the algorithm is intended, this is negligible work. For old points for which a point closer to the most distant point in the safeguarded neighborhood is added to  $X$ , the update of the neighborhood costs only a few additional operations: compare with the stored largest distance, and recalculate the neighborhood if necessary.

All our model fits require valid function values at the data points used. Therefore, for each point with function value NaN (i.e., a point where the function value could not be determined due to a hidden constraint), we use in the fits a temporarily created fictitious function value, defined as follows. Let  $f_{\min}$  and  $f_{\max}$  be the minimal and maximal function value among the safeguarded nearest neighbors of  $x$ , excepting the neighbors where no function value could be obtained, and let  $f_{\min}$  and  $f_{\max}$  be the minimal and maximal function value among all points in  $X$  with finite function values in the case that all safeguarded nearest neighbors of  $x$  had



function value NaN. Then we set  $f = f_{\min} + 10^{-3}(f_{\max} - f_{\min})$  and  $\Delta f = \Delta f_{\max}$ . The fictitious function value is recomputed if the nearest neighbors of a point with function value NaN have changed.

A local model around each point  $x$  is fitted with the aid of the nearest neighbors as follows. Let  $x^k$ ,  $k = 1, \dots, n + \Delta n$ , be the nearest neighbors of  $x$ ,  $f := f(x)$ ,  $f_k := f(x^k)$ , let  $\Delta f$  and  $\Delta f_k$  be the uncertainties in the function values  $f$  and  $f_k$ , respectively, and  $D := \text{diag}(\Delta f / \Delta x_i^2)$ ; note that  $\Delta f_k > 0$  (at least after resetting); see Section 2. Then we consider the equations

$$f_k - f = g^T(x^k - x) + \varepsilon_k((x^k - x)^T D(x^k - x) + \Delta f_k), \quad k = 1, \dots, n + \Delta n, \quad (2)$$

where  $\varepsilon_k$ ,  $k = 1, \dots, n + \Delta n$  are model errors and  $g$  is the gradient to be estimated. The errors  $\varepsilon_k$  account for second (and higher) order deviation from linearity, and for measurement errors; the form of the weight factor guarantees reasonable scaling properties. As we shall see later, one can derive from (2) a sensible separable quadratic surrogate optimization problem (4); a more accurate model would be too expensive to be constructed and minimized around every point.

The model errors are minimized by writing (2) as  $Ag - b = \varepsilon$ , where  $\varepsilon := (\varepsilon_1, \dots, \varepsilon_{n+\Delta n})^T$ ,  $b \in \mathbb{R}^{n+\Delta n}$ , and  $A \in \mathbb{R}^{(n+\Delta n) \times n}$ . In particular,  $b_k := (f - f_k)/Q_k$ ,  $A_{ki} := (x_i - x_i^k)/Q_k$ ,  $Q_k := (x^k - x)^T D(x^k - x) + \Delta f_k$ ,  $k = 1, \dots, n + \Delta n$ ,  $i = 1, \dots, n$ . We make a reduced singular value decomposition  $A = U\Sigma V^T$  and replace the  $i$ th singular value  $\Sigma_{ii}$  by  $\max(\Sigma_{ii}, 10^{-4}\Sigma_{11})$ , to enforce a condition number  $\leq 10^4$ . Then the estimated gradient is determined by  $g := V\Sigma^{-1}U^T b$  and the estimated standard deviation of  $\varepsilon$  by

$$\sigma := \sqrt{\|Ag - b\|_2^2 / \Delta n}.$$

The factor after  $\varepsilon_k$  in (2) is chosen such that for points with a large  $\Delta f_k$  (i.e., inaccurate function values) and a large  $(x^k - x)^T D(x^k - x)$  (i.e., far away from  $x$ ), a larger error in the fit is permitted. Since  $n + \Delta n$  equations are used to determine  $n$  parameters, the system of equations is overdetermined by  $\Delta n$  equations, typically yielding a well-conditioned problem.

A local quadratic fit is computed around the best point  $x^{\text{best}}$ , i.e., the point with the lowest objective function value. Let  $K := \min(n(n+3), N-1)$ , and assume that  $x^k$ ,  $k = 1, \dots, K$ , are the points in  $X$  closest to but distinct from  $x^{\text{best}}$ . Let  $f_k := f(x^k)$  be the corresponding function values and put  $s^k := x^k - x^{\text{best}}$ . Define model errors  $\hat{\varepsilon}_k$  by the equations

$$f_k - f_{\text{best}} = g^T s^k + \frac{1}{2}(s^k)^T G s^k + \hat{\varepsilon}_k((s^k)^T H s^k)^{3/2}, \quad k = 1, \dots, K; \quad (3)$$

here the vector  $g \in \mathbb{R}^n$  and the symmetric matrix  $G \in \mathbb{R}^{n \times n}$  are to be estimated and  $H := (\sum s^l (s^l)^T)^{-1}$  is a scaling matrix which ensures the affine invariance of the fitting procedure (cf. [Deuffhard and Heindl 1979]).

The fact that we expect the best point to be possibly close to the global minimizer warrants the fit of a quadratic model. The exponent  $\frac{3}{2}$  in the error term reflects the expected cubic approximation order of the quadratic model. We refrained from adding an independent term for noise in function value since we found no useful affine invariant recipe that keeps the estimation problem tractable. The  $M := n + \binom{n+1}{2} = \frac{n(n+3)}{2}$  parameters in (3) consisting of the components of  $g$

and the upper triangle of  $G$  are determined by minimizing  $\sum \hat{\varepsilon}_k^2$ . This is done by writing (3) as  $Aq - b = \hat{\varepsilon}$ , where the vector  $q \in \mathbb{R}^M$  contains the parameters to be determined,  $\hat{\varepsilon} := (\hat{\varepsilon}_1, \dots, \hat{\varepsilon}_K)^T$ ,  $A \in \mathbb{R}^{K \times M}$  and  $b \in \mathbb{R}^K$ . Then  $q$  is computed with the aid of the backslash operator in MATLAB as  $q = A \backslash b$ ; in the underdetermined case  $K < M$  the backslash operator computes a regularized solution. We compute  $H$  by making a reduced  $QR$  factorization

$$(s^1, \dots, s^K)^T = QR$$

with an orthogonal matrix  $Q \in \mathbb{R}^{K \times n}$  and a square upper triangular matrix  $R \in \mathbb{R}^{n \times n}$  and obtain  $(s^k)^T H s^k = \|R^{-T} s^k\|^2$ . In the case that  $N \geq n(n+3) + 1$ ,  $\frac{n(n+3)}{2}$  parameters are fitted from  $n(n+3)$  equations, and again the contribution of the points closer to  $x^{\text{best}}$  is larger.

#### 4. RECOMMENDED EVALUATION POINTS

SNOBFIT generates points of different character belonging to five classes. The points selected for evaluation are chosen from these classes in proportions that can be influenced by the user. This gives the user some control over the desired balance between local and global search. The classes are now described in turn.

Class 1 contains at most one point, determined from the local quadratic model around the best point  $x^{\text{best}}$ . If the function value at this point is already known, class 1 is empty.

To be more specific, let  $\Delta x \in \mathbb{R}^n$ ,  $\Delta x > 0$ , be a resolution vector as defined in Section 2, and let  $[u', v'] \subseteq [u, v]$  be the box where the points are to be generated. The point  $w$  of class 1 is obtained by approximately minimizing the local quadratic model around  $x^{\text{best}}$

$$q(x) := f_{\text{best}} + g^T(x - x^{\text{best}}) + \frac{1}{2}(x - x^{\text{best}})^T G(x - x^{\text{best}})$$

(where  $g$  and  $G$  are determined as described in the previous section) over the box  $[x^{\text{best}} - d, x^{\text{best}} + d] \cap [u', v']$ . Here  $d := \max(\max_k |x^k - x^{\text{best}}|, \Delta x)$  (with component-wise max and absolute value), and  $x^k$ ,  $k = 1, \dots, K$ , are the points from which the model has been generated. The components of  $w$  are subsequently rounded to integral multiples of  $\Delta x_i$  into the box  $[u', v']$ .

Since our quadratic model is only locally valid, it suffices to minimize it locally. The minimization is performed with the bound constrained quadratic programming package MINQ from <http://www.mat.univie.ac.at/~neum/software/minq>, designed to find a stationary point, usually a local minimizer, though not necessarily close to the best point. If  $w$  is a point already in  $X$ , a point is generated from a uniform distribution on  $[x^{\text{best}} - d, x^{\text{best}} + d] \cap [u', v']$  and rounded as above, and if this again results in a point already in  $X$ , at most nine more attempts are made to find a point on the grid that is not yet in  $X$ . Since it is possible that this procedure is not successful in finding a point not yet in  $X$ , a call to SNOBFIT may occasionally not return a point of class 1.

A point of class 2 is a guess  $y$  for a putative approximate local minimizer, generated by the recipe described in the next paragraph from a so-called local point. Let  $x \in X$ ,  $f := f(x)$ , and let  $f_1$  and  $f_2$  be the smallest and largest among the function

values of the  $n + \Delta n$  safeguarded nearest neighbors of  $x$ , respectively. The point  $x$  is called *local* if  $f < f_1 - 0.2(f_2 - f_1)$ , i.e., if its function value is “significantly smaller” than that of its nearest neighbors. We require this stronger condition in order to avoid too many local points produced by an inaccurate evaluation of the function values. (This significantly improved performance in noisy test cases.) A call to SNOBFIT will not produce any points of class 2 if there are no local points in the current set  $X$ .

More specifically, for each point  $x \in X$ , we define a trust region size vector  $d := \max(\frac{1}{2} \max_k |x^k - x|, \Delta x)$ , where  $x^k$  ( $k = 1, \dots, n + \Delta n$ ) are the safeguarded nearest neighbors of  $x$ . We solve the problem

$$\begin{aligned} \min g^T p + \sigma p^T D p \\ \text{s.t. } p \in [-d, d] \cap [u' - x, v' - x], \end{aligned} \quad (4)$$

where  $g$ ,  $\sigma$  and  $D$  are defined as in the previous section. This problem is motivated by the model (2), replacing the unknown (and  $x$ -dependent)  $\varepsilon_k$  by the estimated standard deviation. This gives a conservative estimate of the error and usually ensures that the surrogate minimizer is not too far from the region where higher order terms missing in the linear model are not relevant. (4) is a bound constrained separable quadratic program and can therefore be solved explicitly for the minimizer  $\hat{p}$  in  $O(n)$  operations. The point  $y$  obtained by rounding the coordinates of  $x + \hat{p}$  to integral multiples of the coordinates of  $\Delta x$  into the interior of  $[u', v']$  is a potential new evaluation point. If  $y$  is a point already in  $X$ , a point is generated from a uniform distribution on  $[x - d, x + d] \cap [u', v']$  and rounded as above. If this again results in a point already in  $X$ , up to four more attempts are made to find a point  $y$  on the grid that is not yet in  $X$  before the search is stopped.

In order to give the user some measure for assessing the quality of  $y$  (see the end of Section 2), we also compute an estimate

$$f_y := f + g^T(y - x) + \sigma((y - x)^T D(y - x) + \Delta f). \quad (5)$$

for the function value at  $y$ .

All points obtained in this way when  $x$  runs through the local points are taken to be in class 2; the remaining points obtained in the same way from nonlocal points are taken to be in class 3. Points of class 2 and 3 are alternative good points and selected with a view to their expected function value, i.e., they represent another aspect of the local search part of the algorithm. However, since they may occur near points throughout the available sample, they also contribute significantly to the global search strategy.

The points of class 4 are points in so far unexplored regions; i.e., they are generated in large subboxes of the current partition. They represent the most global aspect of the algorithm. For a box  $[\underline{x}, \bar{x}]$  with corresponding point  $x$ , the point  $z$  of class 4 is defined by

$$z_i := \begin{cases} \frac{1}{2}(\underline{x}_i + x_i) & \text{if } x_i - \underline{x}_i > \bar{x}_i - x_i, \\ \frac{1}{2}(x_i + \bar{x}_i) & \text{otherwise,} \end{cases}$$

and  $z_i$  is rounded into  $[\underline{x}_i, \bar{x}_i]$  to the nearest integral multiple of  $\Delta x_i$ . The input parameter  $p$  denotes the desired fraction of points of class 4 among the points of class 2 to 4; see Section 6, Step 8.

Points of class 5 are only produced if the algorithm does not manage to reach the desired number of points by generating points of classes 1 to 4, for example, when there are not enough points available yet to build local quadratic models, which happens in particular in an initial call with an empty set of input points and function values. These points are chosen from a set of random points such that their distances from the points already in the list are maximal. The points of class 5 extend  $X$  to a space-filling design.

In order to generate  $n_5$  points of class 5,  $100n_5$  random uniformly distributed points in  $[u', v']$  are sampled and their coordinates are rounded to integral multiples of  $\Delta x_i$ , which yields a set  $Y$  of points. Initialize  $X'$  as the union of  $X$  and the points in the list of recommended evaluation points of the current call to SNOBFIT, and let  $Y := Y \setminus X'$ . Then the set  $X_5$  of points of class 5 is generated by the following algorithm, which generates the points in  $Y_5$  such that they differ as much as possible in some sense from the points in  $X'$  and from each other:

```

 $X_5 = \emptyset$ 
if  $X' = \emptyset$  &  $Y \neq \emptyset$ 
  choose  $y \in Y$ 
   $X' = \{y\}; Y = Y \setminus \{y\}; X_5 = X_5 \cup \{y\};$ 
end if
while  $|X_5| < n_5$  &  $Y \neq \emptyset$ 
   $y = \operatorname{argmax}_{y \in Y} \min_{x \in X'} \|x - y\|_2;$ 
   $X' = X' \cup \{y\}; Y = Y \setminus \{y\}; X_5 = X_5 \cup \{y\};$ 
end while

```

The algorithm may occasionally yield less than  $n_5$  points, namely in the case that the set  $Y$  as defined before applying the algorithm contains less than  $n_5$  points or is even empty. In particular, this happens in the case that all points of the grid generated by  $\Delta x$  are already in  $X'$ .

To prevent getting essentially copies of the same point in one call, points of classes 2, 3 and 4 are only accepted if they differ from all previously generated suggested evaluation points in the same call by at least  $0.1(v'_i - u'_i)$  in at least one coordinate  $i$ . (This condition is not enforced for the points of class 5 since they are not too close to the other points by construction.) The algorithm may fail to generate the requested number  $n_{\text{req}}$  of points in the case that the grid generated by  $\Delta x$  has already been (nearly) exhausted on the search box  $[u', v']$ . In that case a warning is given, and the user should change  $[u', v']$  and/or refine  $\Delta x$ ; for the latter see the end of Section 6.

## 5. THE BRANCHING ALGORITHM

SNOBFIT proceeds by partitioning the search region  $[u, v]$  into subboxes, each containing exactly one point where the function has been evaluated. After the input of new points and function values by the user at the beginning of each call to SNOBFIT, some boxes will contain more than one point and therefore a splitting algorithm is required.

We want to split a subbox  $[\underline{x}, \bar{x}]$  of  $[u, v]$  containing the pairwise distinct points  $x^k$ ,  $k = 1, \dots, K$ ,  $K \geq 2$ , such that each subbox contains exactly one point. If  $K = 2$ , we choose  $i$  with  $|x_i^1 - x_i^2|/(v_i - u_i)$  maximal and split along the  $i$ th coordinate at

$y_i = \lambda x_i^1 + (1 - \lambda)x_i^2$ ; here  $\lambda$  is the golden section number  $\rho := \frac{1}{2}(\sqrt{5} - 1) \approx 0.62$  if  $f(x^1) \leq f(x^2)$ , and  $\lambda = 1 - \rho$  otherwise. The subbox with the lower function value gets the larger part of the original box so that it is eligible for being selected for the generation of a point of class 4 more quickly.

In the case  $K > 2$ , the coordinate in which the points have maximal variance relative to  $[u, v]$  is selected for a perpendicular split, and the split is done in the largest gap and closer to the point with the larger function value, so that the point with the smaller function value will again get the larger part of the box. This results in the following algorithm:

```

while there is a subbox containing more than one point
    choose the subbox containing the highest number of points
    choose  $i$  such that the variance of  $x_i/(v_i - u_i)$  is maximal, where the
    variance is taken over all points  $x$  in the subbox
    sort the points such that  $x_i^1 \leq x_i^2 \leq \dots$ 
    split in the coordinate  $i$  at  $y_i = \lambda x_i^j + (1 - \lambda)x_i^{j+1}$ , where
     $j = \operatorname{argmax}(x_i^{j+1} - x_i^j)$ , with
     $\lambda = \rho$  if  $f(x^j) \leq f(x^{j+1})$ , and  $\lambda = 1 - \rho$  otherwise
end while
    
```

To each subbox  $[\underline{x}, \bar{x}]$  we assign its *smallness*

$$S := - \sum_{i=1}^n \operatorname{round} \left( \log_2 \frac{\bar{x}_i - \underline{x}_i}{v_i - u_i} \right), \quad (6)$$

where  $\operatorname{round}$  is the function rounding to nearest integers and  $\log_2$  denotes the logarithm to the base 2. This quantity roughly measures how many bisections are necessary to obtain this box from  $[u, v]$ . We have  $S = 0$  for the exploration box  $[u, v]$ , and  $S$  is large for small boxes. For a more thorough global search, boxes with low smallness should be preferably explored when selecting new points for evaluation.

## 6. THE SNOBFIT ALGORITHM

Using the ingredients discussed in the previous sections, one call to SNOBFIT proceeds in the following 11 steps.

STEP 1. The search box  $[u, v]$  is updated if the user changed it as described in Section 2. The vectors  $u$  and  $v$  are defined such that  $[u, v]$  is the smallest box containing  $[u', v']$ , all new input points and, in the case of a continuation call to SNOBFIT, also the box  $[u^{\text{old}}, v^{\text{old}}]$  from the previous iteration.  $[u, v]$  is considered to be the box to be explored and a box tree of  $[u, v]$  is generated; however, all suggested new evaluation points are in  $[u', v']$ . Moreover, we initialize  $J_4$ , which will contain the indices of boxes marked for the generation of a point of class 4, as the empty set.

STEP 2. Duplicates in the set of points consisting of the “new” points and in a continuation call also the “old” points from the previous iterations are thrown away, and the corresponding function value  $f$  and uncertainty  $\Delta f$  are updated as follows. If a point has been put into the input list  $m$  times with function values  $f_1, \dots, f_m$  and corresponding uncertainties  $\Delta f_1, \dots, \Delta f_m$  (if some of the  $f_i$  are

NaNs and some aren't, the NaNs are thrown away and  $m$  is reset), the quantities  $f$  and  $\Delta f$  are defined by

$$f = \frac{1}{m} \sum_{i=1}^m f_i, \quad \Delta f = \sqrt{\frac{1}{m} \sum_{i=1}^m ((f_i - f)^2 + \Delta f_i^2)}.$$

STEP 3. All current boxes containing more than one point are split according to the algorithm described in Section 5. The smallness (also defined in Section 5) is computed for these boxes. If  $[u, v]$  is larger than  $[u^{\text{old}}, v^{\text{old}}]$  in a continuation call, the box bounds and the smallness are updated for the boxes for which this is necessary.

STEP 4. If we have  $|X| < n + \Delta n + 1$  for the current set  $X$  of points, or if all function values  $\neq$  NaN (if any) are identical, go to Step 11. Otherwise, for each new point  $x$  a vector pointing to  $n + \Delta n$  safeguarded nearest neighbors is computed. The neighbor lists of some old points are updated in the same way if necessary.

STEP 5. For any new input point  $x$  with function value NaN (which means that the function value could not be determined at that point) and for all old points marked infeasible whose nearest neighbors have changed, we define fictitious values  $f$  and  $\Delta f$  as specified in Section 3.

STEP 6. Local fits (as described in Section 3) around all new points and all old points with changed nearest neighbors are computed and the potential points  $y$  of class 2 and 3 and their estimated function values  $f_y$  are determined as in Section 4.

STEP 7. The current best point  $x^{\text{best}}$  and the current best function value  $f_{\text{best}}$  in  $[u', v']$  are determined; if the objective function has not been evaluated in  $[u', v']$  yet,  $n_1 = 0$  and go to Step 8. A local quadratic fit around  $x^{\text{best}}$  is computed as described in Section 3 and the point  $w$  of class 1 is generated as described in Section 4. If such a point  $w$  was generated, let  $w$  be contained in the subbox  $[\underline{x}^j, \bar{x}^j]$  (in the case that  $w$  belongs to more than one box, a box with minimal smallness is selected). If  $\min_i ((\bar{x}_i^j - \underline{x}_i^j)/(v_i - u_i)) > 0.05 \max_i ((\bar{x}_i^j - \underline{x}_i^j)/(v_i - u_i))$ , the point  $w$  is put into the list of suggested evaluation points, and otherwise (if the smallest side length of the box  $[\underline{x}^j, \bar{x}^j]$  relative to  $[u, v]$  is too small compared to the largest one) we set  $J_4 = \{j\}$ , i.e., a point of class 4 is to be generated in this box later, which seems to be more appropriate for such a long, narrow box. This gives  $n_1$  evaluation points ( $n_1 = 0$  or 1).

STEP 8. To generate the remaining  $m := n_{\text{req}} - n_1$  evaluation points, let  $n' := \lfloor pm \rfloor$  and  $n'' := \lceil pm \rceil$ . Then a random number generator sets  $m_1 = n'$  with probability  $mp - n'$  and  $m_1 = n''$  otherwise. Then  $m - m_1$  points of classes 2 and 3 together and  $m_1$  points of class 4 are to be generated.

STEP 9. If  $X$  contains any local points, first at most  $m - m_1$  points  $y$  generated from the local points are chosen in the order of ascending model function values  $f_y$  to yield points of class 2. If the desired number of  $m - m_1$  points has not been reached yet, afterwards points  $y$  pertaining to nonlocal  $x \in X$  are taken (again in the order of ascending  $f_y$ ).

For each potential point of class 2 or 3 generated in this way, a subbox  $[\underline{x}^j, \bar{x}^j]$  of the box tree with minimal smallness is determined with  $y \in [\underline{x}^j, \bar{x}^j]$ . If  $\min_i ((\bar{x}_i^j - \underline{x}_i^j)/(v_i - u_i)) \leq 0.05 \max_i ((\bar{x}_i^j - \underline{x}_i^j)/(v_i - u_i))$ , the point  $y$  is not put into the list

of suggested evaluation points but instead we set  $J_4 = J_4 \cup \{j\}$ , i.e., the box is marked for the generation of a point of class 4.

STEP 10. Let  $S_{\min}$  and  $S_{\max}$  denote the minimal and maximal smallness, respectively, among the boxes in the current box tree, and let  $M := \lfloor \frac{1}{3}(S_{\max} - S_{\min}) \rfloor$ . For each smallness  $S = S_{\min} + m$ ,  $m = 0, \dots, M$ , the boxes are sorted according to ascending function values  $f(x)$ . First a point of class 4 is generated from the box with  $S = S_{\min}$  with smallest  $f(x)$ . If  $J_4 \neq \emptyset$ , then the points of class 4 belonging to the subboxes with indices in  $J_4$  are generated. Subsequently, a point of class 4 is generated in the box with smallest  $f(x)$  at each nonempty smallness level  $S_{\min} + m$ ,  $m = 1, \dots, M$ , and then the smallness levels from  $S_{\min}$  to  $S_{\min} + M$  are gone through again etc. until we either have  $n_{\text{req}}$  recommended evaluation points or there are no eligible boxes for generating points of class 4 any more. We assign to the points of class 4 the model function values obtained from the local models pertaining to the points in their boxes.

A point of classes 2 to 4 is only accepted if it is not yet in  $X$  and not yet in the list of recommended evaluation points and if it differs from all points already in the current list of recommended evaluation points by at least  $\Delta y_i$  in at least one coordinate  $i$ .

STEP 11. If the number of suggested evaluation points is still less than  $n_{\text{req}}$ , the set of evaluation points is filled up with points of class 5 as described in Section 4. If local models are already available, the model function values for the points of class 5 are determined as the ones for the points of class 4 (see Step 10); otherwise, they are set to NaN.

**Stopping criterion.** Since SNOBFIT is called explicitly before each new evaluation, the search continues as long as the calling agent (an experimenter or a program) finds it reasonable to continue. A natural stopping test would be to quit exploration (or move exploration to a different “box of interest”) if for a number of consecutive calls to SNOBFIT no new point of class 1 is generated. Indeed, this means that SNOBFIT thinks that, according to the current model, the best point is already known; but since the model may be inaccurate, it is sensible to have this confirmed repeatedly before actually stopping.

**Changing the objective function.** Suppose that the objective function  $f$  is of the form  $f(x) = \varphi(y(x))$ , where the vector  $y(x) \in \mathbb{R}^k$  is obtained by time-consuming experiments or simulations but the function  $\varphi$  can be computed cheaply. In this case we may change  $\varphi$  during the solution process without wasting the effort spent in obtaining earlier function evaluations. Indeed, suppose that the objective function  $f$  has already been evaluated at  $x_1, \dots, x_M$  and that, at some moment, the user decides that the objective function  $\tilde{f}(x) = \psi(y(x))$  is more appropriate, where  $\psi$  can be computed cheaply, too. Then the already determined vectors  $y(x_1), \dots, y(x_M)$  can be used to compute  $\tilde{f}(x_1), \dots, \tilde{f}(x_M)$ , and we can start a new SNOBFIT job with  $x_1, \dots, x_M$  and  $\tilde{f}(x_1), \dots, \tilde{f}(x_M)$ , i.e., we use the old grid of points but make a new partition of the space before the SNOBFIT algorithm is continued. For examples, see Section 9.3.

**Changing the resolution vector  $\Delta x$ .** The program does not allow for changes

of the resolution vector  $\Delta x$  during a job but  $\Delta x$  is only set in an initial call. However, if the user wants to change the grid (in most cases the new grid will be finer), a new job can be started with the old points and function values as input, and a new partition of the search space will be generated. It does not matter if the old points are not on the new grid; the points generated by SNOBFIT are on the grid but the input points do not have to be on the grid.

## 7. CONVERGENCE OF THE ALGORITHM

For a convergence analysis, we assume that the exploration box  $[u, v]$  is scaled to  $[0, 1]^n$  and that we do not do any rounding, which means that all points of class 4 are accepted since they differ from the old point in the box. Moreover, we assume that  $[u', v'] = [u, v] = [0, 1]^n$  during the whole job, that at least one point of class 4 is generated from a box of minimal smallness in each call to SNOBFIT, and that the function is evaluated at the points suggested by SNOBFIT.

Then SNOBFIT is guaranteed to converge to a global minimizer if the objective function is continuous – or at least continuous in the neighborhood of a global optimizer. This follows from the fact that, under the assumptions made above, the set of points produced by SNOBFIT forms a dense subset of the search space. That is, given any point  $x \in [u, v]$  and any  $\delta > 0$ , SNOBFIT will eventually generate a point within a distance  $\delta$  from  $x$ .

When a box with smallness  $S$  is split into two parts, the following two cases are possible if  $S_1$  and  $S_2$  denote the smallnesses of the two subboxes. When the box is split into two equal halves, we have  $S_1 = S_2 = S + 1$ , and otherwise we have  $S_1 \geq S$  and  $S_2 \geq S + 1$  (after renumbering the two subboxes if necessary); i.e., at most one of the two subboxes has the same smallness as the parent box and the other one has a larger smallness. This implies that after a box has been split, the number of boxes with minimal smallness occurring in the current partition either stays the same or decreases by one.

The definition of the point of class 4 prevents splits in too narrow variables. Indeed, consider a box  $[\underline{x}, \bar{x}]$  containing the point  $x$  and let

$$\bar{x}_i - \underline{x}_i \geq \bar{x}_j - \underline{x}_j \quad \text{for } j = 1, \dots, n \quad (7)$$

(i.e., the  $i$ th dimension of the box is the largest one). In order to simplify notation, we assume that

$$\bar{x}_j - x_j \geq x_j - \underline{x}_j \quad (8)$$

and thus we have  $z_j = \frac{1}{2}(\bar{x}_j + x_j)$  for  $j = 1, \dots, n$  (the other cases can be handled similarly) for the point  $z$  of class 4. According to the branching algorithm defined in Section 5, the box will be split along the coordinate  $k$  with  $z_k - x_k = \frac{1}{2}(\bar{x}_k - x_k)$  maximal, i.e.,  $\bar{x}_k - x_k \geq \bar{x}_j - x_j$  for  $j = 1, \dots, n$ . Then (7), (8) and the definition of  $k$  imply

$$\bar{x}_k - \underline{x}_k \geq \bar{x}_k - x_k \geq \bar{x}_i - x_i \geq \frac{1}{2}(\bar{x}_i - \underline{x}_i),$$

which means that only splits along coordinates  $k$  with  $\bar{x}_k - \underline{x}_k \geq \frac{1}{2} \max(\bar{x}_j - \underline{x}_j)$  are possible. After a class 4 split along coordinate  $k$ , the  $k$ th side length of the larger part is at most a fraction  $\frac{1}{4}(1 + \sqrt{5}) \approx 0.8090$  of the original one.



These properties give the following convergence theorem for SNOBFIT without rounding and where at least one point of class 4 is generated in a box of minimal smallness  $S_{\min}$  in each call to SNOBFIT.

**THEOREM 7.1.** *Suppose that the global minimization problem (1) has a solution  $x^* \in [u, v]$ , and that  $f : [u, v] \rightarrow \mathbb{R}$  is continuous in a neighborhood of  $x^*$ , and let  $\varepsilon > 0$ . Then the algorithm will eventually find a point  $x$  with  $f(x) < f(x^*) + \varepsilon$ , i.e., the algorithm converges.*

**PROOF.** Since  $f$  is continuous in a neighborhood of  $x^*$ , there exists a  $\delta > 0$  such that  $f(x) < f(x^*) + \varepsilon$  for all  $x \in B := [x^* - \delta, x^* + \delta] \cap [u, v]$ . We prove the theorem by contradiction and assume that the algorithm will never generate a point in  $B$ . Let  $[\underline{x}^1, \bar{x}^1] \supseteq [\underline{x}^2, \bar{x}^2] \supseteq \dots$  be the sequence of boxes containing  $x^*$  after each call to SNOBFIT and  $x^k \in [\underline{x}^k, \bar{x}^k]$  be the corresponding points. Then  $\max_i |x_i^k - x_i^*| \geq \delta$  for  $k = 1, 2, \dots$  and therefore  $\max_i (\bar{x}_i^k - \underline{x}_i^k) \geq \delta$  for  $k = 1, 2, \dots$ . Since we assume that in each call to SNOBFIT at least one box with minimal smallness  $S_{\min}$  is split and the number of boxes with smallness  $S_{\min}$  does not increase but will eventually decrease after sufficiently many splits, after sufficiently many calls to SNOBFIT, there are no boxes with smallness  $S_{\min}$  left any more. If  $S_k$  denotes the smallness of the box  $[\underline{x}^k, \bar{x}^k]$ , we must therefore have  $\lim_{k \rightarrow \infty} S_k = \infty$ . Assume that  $\max_i (\bar{x}_i^{k_0} - \underline{x}_i^{k_0}) \geq 20 \min_i (\underline{x}_i^{k_0} - \underline{x}_i^{k_0})$  for some  $k_0$ . Then the next split that the box  $[\underline{x}^{k_0}, \bar{x}^{k_0}]$  will undergo is a class 4 split along a coordinate  $i_0$  with  $\bar{x}_{i_0}^{k_0} - \underline{x}_{i_0}^{k_0} \geq \frac{1}{2} \max_i (\bar{x}_i^{k_0} - \underline{x}_i^{k_0})$ . If this procedure is repeated, we will eventually obtain a contradiction to  $\max_i (\bar{x}_i^k - \underline{x}_i^k) \geq \delta$  for  $k = 1, 2, \dots$ . Therefore  $\delta \leq \max_i (\bar{x}_i^k - \underline{x}_i^k) < 20 \min_i (\bar{x}_i^k - \underline{x}_i^k)$  for  $k \geq k_1$  for some  $k_1$ , i.e.,  $\bar{x}_i^k - \underline{x}_i^k > \frac{\delta}{20}$  for  $i = 1, \dots, n$ ,  $k \geq k_1$ . This implies  $S_k \leq -n \log_2 \frac{\delta}{20}$  for  $k \geq k_1$ , which is a contradiction.  $\square$

Note that this theorem only guarantees that a point close to a global minimizer will eventually be found but it does not say how fast this convergence is. This accounts for the occasional slow jobs reported in Section 9 since we only make a finite number of calls to SNOBFIT. Also note that the actual implementation differs from the idealized version discussed in this section; in particular, choosing the resolution too coarse may prevent the algorithm from finding a good approximation to the global minimum value.

## 8. HANDLING SOFT CONSTRAINTS

In this section we consider the constrained optimization problem

$$\begin{aligned} \min f(x) \\ \text{s.t. } x \in [u, v], \quad F(x) \in \mathbf{F}, \end{aligned} \tag{9}$$

where, in addition to the assumptions made at the beginning of the paper,  $F : [u, v] \rightarrow \mathbb{R}^m$  is a vector of  $m$  continuous constraint functions  $F_1(x), \dots, F_m(x)$ , and  $\mathbf{F} := [\underline{F}, \bar{F}]$  is a box in  $\mathbb{R}^m$  defining the constraints on  $F(x)$ .

Traditionally (see [Fiacco and McCormick 1990]), constraints that cannot be handled explicitly are accounted for in the objective function, using simple  $l_1$  or  $l_2$  penalty terms for constraint violations, or logarithmic barrier terms penalizing

the approach to the boundary. There are also so-called exact penalty functions whose optimization gives the exact solution (see, e.g., [Nocedal and Wright 1999]); however, this only holds if the penalty parameter is large enough, and what is large enough cannot be assessed without having global information. The use of more general transformations may give rise to more precisely quantifiable approximation results. In particular, if it is known in advance that all constraints apart from the simple constraints are soft constraints only (so that some violation is tolerated), one may pick a transformation that incorporates prescribed tolerances into the reformulated simply constrained problem.

The implementation of soft constraints in SNOBFIT is based upon the following transformation which has nice theoretical properties, improving upon [Dallwig et al. 1997]:

**THEOREM 8.1.** (Soft optimality theorem). *For suitable  $\Delta > 0, \underline{\sigma}_i, \bar{\sigma}_i > 0, f_0 \in \mathbb{R}$ , let*

$$q(x) = \frac{f(x) - f_0}{\Delta + |f(x) - f_0|},$$

$$\delta_i(x) = \begin{cases} (F_i(x) - \underline{F}_i)/\underline{\sigma}_i & \text{if } F_i(x) \leq \underline{F}_i, \\ (F_i(x) - \bar{F}_i)/\bar{\sigma}_i & \text{if } F_i(x) \geq \bar{F}_i, \\ 0 & \text{otherwise,} \end{cases}$$

$$r(x) = 2(\sum \delta_i^2(x))/(1 + \sum \delta_i^2(x)).$$

*Then the merit function*

$$f_{\text{merit}}(x) = q(x) + r(x)$$

*has its range bounded by  $(-1, 3)$ , and the global minimizer  $\hat{x}$  of  $f_{\text{merit}}$  in  $[u, v]$  either satisfies*

$$F_i(\hat{x}) \in [\underline{F}_i - \underline{\sigma}_i, \bar{F}_i + \bar{\sigma}_i] \quad \text{for all } i, \quad (10)$$

$$f(\hat{x}) \leq \min\{f(x) \mid F(x) \in \mathbf{F}, x \in [u, v]\}, \quad (11)$$

*or one of the following two conditions holds:*

$$\{x \in [u, v] \mid F(x) \in \mathbf{F}\} = \emptyset, \quad (12)$$

$$f_0 < \min\{f(x) \mid F(x) \in \mathbf{F}, x \in [u, v]\}. \quad (13)$$

**PROOF.** Clearly,  $q(x) \in (-1, 1)$  and  $r(x) \in [0, 2)$ , so that  $f_{\text{merit}}(x) \in (-1, 3)$ . If there is a feasible point  $x$  with  $f(x) \leq f_0$  then  $q(x) \leq 0$ ,  $r(x) = 0$  at this point. Since  $f_{\text{merit}}$  is monotone increasing in  $q + r$ , we conclude from  $f_{\text{merit}}(\hat{x}) \leq f_{\text{merit}}(x)$  that

$$q(\hat{x}) \leq q(\hat{x}) + r(\hat{x}) \leq q(x) + r(x) = q(x),$$

$$-1 + r(\hat{x}) \leq q(\hat{x}) + r(\hat{x}) \leq q(x) + r(x) \leq 0,$$

hence  $f(\hat{x}) \leq f(x)$ , giving (11), and  $r(\hat{x}) \leq 1$ ,

$$\delta_i^2(\hat{x}) \leq \sum \delta_i^2(\hat{x}) \leq 1,$$

giving (10).  $\square$

Of course there are other choices for  $q(x)$ ,  $r(x)$ ,  $f_{\text{merit}}(x)$  with similar properties. The choices given are implemented in SNOBFIT and lead to a continuously differentiable merit function with Lipschitz-continuous gradient if  $f$  and  $F$  have these properties. (The denominator of  $q(x)$  is nonsmooth, but only when the numerator vanishes, so that this only affects the Hessian.)

Since the merit function is bounded by  $(-1, 3)$  even if  $f$  and/or some  $F_i$  are unbounded, the formulation is able to handle so-called *hidden constraints*. There, the conditions for infeasibility are not known explicitly but are discovered only when attempting to evaluate one of the functions involved. In such a case, if the function cannot be evaluated, SNOBFIT simply sets the merit function value to 3.

(12) and (13) are degenerate cases that do not occur if a feasible point is already known and we choose  $f_0$  as the function value of the best feasible point known (at the time of posing the problem). A suitable value for  $\Delta$  is the median of the  $|f(x) - f_0|$  for an initial set of trial points (in the context of global optimization often determined by a space-filling design [McKay et al. 1979; Owen 1992; 1994; Sacks et al. 1989; Tang 1993]).

The number  $\sigma_i$  measures the degree to which the constraint  $F_i(x) \in \mathbf{F}_i$  may be softened; suitable values are in many practical applications available from the meaning of the constraints. If the user asks for high accuracy, i.e., if some  $\sigma_i$  is taken to be very small, the merit function will be ill-conditioned, and the local quadratic models used by SNOBFIT may be quite inaccurate. In this case, SNOBFIT becomes very slow and cannot locate the global minimum with high accuracy. One should therefore request only moderately low accuracy from SNOBFIT, and postprocess the approximate solution with a local algorithm for noisy *constrained* optimization. Possibilities include CONDOR [Vanden Berghen and Bersini 2005], which, however, requires that the constraints are cheap to evaluate, or DFO [Conn et al. 1997], which has constrained optimization capabilities in its Version 2 [Scheinberg 2003], based on sequential surrogate optimization with general constraints.

## 9. NUMERICAL RESULTS

In this section we present some numerical results. In Subsection 9.1 a set of ten well-known test functions is considered in the presence of additive noise as well as for the unperturbed case. In Section 9.2, two examples of the six-hump camel function with additional hidden constraints are investigated. Finally, in Subsection 9.3 the theory of Section 8 is applied to some problems from the Hock–Schittkowski collection.

Since SNOBFIT has no stopping rules, the stopping tests used in the numerical experiments were chosen (for easy comparison) by reaching a condition depending on the (known) optimal function values.

### 9.1 Testing noisy function evaluations

In this subsection we report results of SNOBFIT on the 9 test functions used in [Jones et al. 1993], where an extensive comparison of algorithms is presented; this test set was also used in [Huyer and Neumaier 1999]. Moreover, we also consider the well-known two-dimensional Rosenbrock function. Let  $n$  be the dimension of the problem, and the default box bounds  $[u, v]$  from the literature were used. We set  $\Delta x = 10^{-5}(v - u)$  and  $p = 0.5$ . The algorithm was started with  $n + 6$  points

chosen at random from  $[u, v]$  and their  $i$ th coordinates were rounded to integral multiples of  $\Delta x_i$ . In each call to SNOBFIT,  $n + 6$  points in  $[u, v]$  were generated.

In order to simulate the presence of noise (issue (I3) from the Introduction), we considered

$$\tilde{f}(x) := f(x) + \sigma N,$$

where  $N$  is a normally distributed random variable with mean 0 and variance 1, and  $\Delta f$  was set to  $\max(3\sigma, \sqrt{\varepsilon})$ , where  $\varepsilon := 2.22 \cdot 10^{-16}$  is the machine precision. The case  $\sigma = 0$  corresponds to the unperturbed problems.

Let  $f^*$  be the known optimal function value, which is 0 for Rosenbrock and  $\neq 0$  for the other test functions. The algorithm was stopped if  $(f_{\text{best}} - f^*)/|f^*| < 10^{-2}$  for  $f^* \neq 0$ ; in the case  $f^* = 0$  it was stopped if  $f_{\text{best}} \leq 10^{-5}$ .

Since a random element is contained in the initial points as well as the function evaluations, 10 jobs were computed with each function and each value of  $\sigma$ , and the median of function calls needed to find a global minimizer was computed. In Table I, the dimensions  $n$ , the standard box bounds  $[u, v]$ , the numbers  $N_{\text{loc}}$  and  $N_{\text{glob}}$  of local and global minimizers in  $[u, v]$ , respectively, and the median of function values for  $\sigma = 0, 0.01$ , and  $0.1$  are given. In the case  $N_{\text{loc}} > N_{\text{glob}}$ , we have to deal with issue (I4) from the Introduction.

	$n$	$[u, v]$	$N_{\text{loc}}$	$N_{\text{glob}}$	$\sigma = 0$	$\sigma = 0.01$	$\sigma = 0.1$
Branin	2	$[-5, 10] \times [0, 15]$	3	3	56	52	48
Six-hump camel	2	$[-3, 3] \times [-2, 2]$	6	2	68	56	48
Goldstein–Price	2	$[-2, 2]^2$	4	1	132	144	184
Shubert	2	$[-10, 10]^2$	760	18	220	248	200
Hartman 3	3	$[0, 1]^3$	4	1	54	59	54
Hartman 6	6	$[0, 1]^6$	4	1	110	666	348
Shekel 5	4	$[0, 10]^4$	5	1	490	515	470
Shekel 7	4	$[0, 10]^4$	7	1	445	455	485
Shekel 10	4	$[0, 10]^4$	10	1	475	445	480
Rosenbrock	2	$[-5.12, 5.12]^2$	1	1	432	576	272

Table I. Dimensions, box bounds, numbers of local and global minimizers and results with SNOBFIT for the unperturbed and perturbed problems

Apart from the three outliers mentioned in a moment, it took the algorithm less than 3000 function evaluations to fulfil the stopping criterion. One job with Shekel 5 and  $\sigma = 0$  required 9910 function values since the algorithm was trapped for a while in the low-lying local minimizer  $(8, 8, 8, 8)$ , one job with Hartman 6 and  $\sigma = 0.01$  required 5004 function values, and one job with Rosenbrock and  $\sigma = 0.1$  even required 16688 function values to satisfy the stopping criterion. However, due to the nature of the Rosenbrock function (ill-conditioned Hessian at small function values), the points found for the perturbed problems are often far away from the minimizer  $(1, 1)$  of the unperturbed problem since the stopping criterion is fulfilled by finding a negative function value caused by perturbations. The points found ranged from  $(0.81142, 0.65915)$  to  $(1.1611, 1.35136)$  for  $\sigma = 0.01$  and from  $(0.64234, 0.4173)$  to  $(1.38506, 1.94155)$  for  $\sigma = 0.1$ . As expected, we obtained less accurate points for larger  $\sigma$ , and for all these points we have  $x_2 \approx x_1^2$ .

In the case of the unperturbed problem ( $\sigma = 0$ ), the results are competitive with results of noise-free global optimization algorithms (see [Jones et al. 1993; Huyer and Neumaier 1999; Hirsch et al. 2007]).

## 9.2 Hidden constraints

We consider the six-hump camel function on the standard box  $[-3, 3] \times [-2, 2]$ . Its global minimum is  $f^* = -1.0316284535$ , attained at the two points  $x^* = (\pm 0.08984201, \mp 0.71265640)$ , and the function also has two pairs of nonglobal local minimizers. We consider two different hidden constraints; each one cuts off the global minimizers:

- (a)  $4x_1 + x_2 \geq 2$
- (b)  $4x_1 + x_2 \geq 4$

For the constraint (a), we obtain a global minimizer  $x^* \approx (0.316954, 0.732185)$  at the boundary of the feasible set with function value  $f^* \approx -0.381737$ . We computed 10 jobs similarly as in the previous subsection, and the median of function evaluations needed to find the new  $f^*$  with a relative error of at most 1% was 708.

For the constraint (b), the global minimizer is  $x^* \approx (1.703607, -0.796084)$ , which is a local minimizer of the original problem, with function value  $f^* \approx -0.215464$ . In this case, the median of function evaluations needed to solve the problem with the required accuracy taken from 10 jobs was 92.

As expected, the problem where the minimizer is on the boundary of the feasible set is harder, i.e., requires a larger number of function evaluations, since the detailed position of the hidden boundary is needed here, which is difficult to find.

## 9.3 Soft constraints and change of the objective function

In this subsection we apply the theory of Section 8 to some problems from the well-known collection [Hock and Schittkowski 1981]. Since SNOBFIT only works for bound constrained problems and we do not want to introduce artificial bounds, we selected from the collection a few problems with finite bounds for all variables. Let  $n$  be the dimension of the problem. In addition to the standard starting point from [Hock and Schittkowski 1981],  $n + 5$  random points were sampled from a uniform distribution on  $[u, v]$ . If there is any feasible point among these  $n + 6$  points, we choose  $f_0$  as the objective function value of the best feasible point in this set; otherwise we set  $f_0 := 2f_{\max} - f_{\min}$ , where  $f_{\max}$  and  $f_{\min}$  denote the largest and smallest of the objective function values, respectively. Then we set  $\Delta$  to the median of  $|f(x) - f_0|$ . When the first point  $x$  with merit function value  $f_{\text{merit}}(x) < 0$  has been found, we update  $f_0$  and  $\Delta$  by the same recipe if a feasible point has already been found. This results in a change of the objective function (issue (I10)), which is handled as described in Section 6.

We use again  $p = 0.5$ ,  $\Delta x = 10^{-5}(v - u)$ ,  $\Delta f = \sqrt{\varepsilon}$  and generate  $n + 6$  points in each call to SNOBFIT. As accuracy requirements we chose  $\underline{\sigma}_i = \bar{\sigma}_i = \sigma b_i$ ,  $i = 1, \dots, m$ , where the  $b_i$  are the components of the vectors  $b$  given in Table II.  $b_i$  is the absolute value of the additive constant contained in the  $i$ th constraint, if nonzero; otherwise we chose  $b_i = 1$  for inequality constraints and (to increase the soft feasible region)  $b_i = 10$  for equality constraints.

	$n$	$m_i$	$m_e$	$x^*$	$b$
HS18	2	2	0	$(\sqrt{250}, \sqrt{2.5})$	(25, 25)
HS19	2	2	0	(14.095, 0.84296079)	(100, 82.81)
HS23	2	5	0	(1, 1)	(1, 1, 9, 1, 1)
HS31	3	1	0	$(\frac{1}{\sqrt{3}}, \sqrt{3}, 0)$	1
HS36	3	1	0	(20, 11, 15)	72
HS37	3	2	0	(24, 12, 12)	(72, 1)
HS41	4	0	1	$(\frac{2}{3}, \frac{1}{3}, \frac{1}{3}, 2)$	10
HS65	3	1	0	(3.650461821, 3.65046169, 4.6204170507)	48

Table II. Problem number, dimension, number of inequality constraints, number of equality constraints, and global minimizer of some Hock–Schittkowski problems

	$\sigma = 0.05$		$\sigma = 0.01$		$\sigma = 0.005$	
	$nf$	$x$	$nf$	$x$	$nf$	$x$
HS18	78	(13.9, 1.74)	265	(16.2, 1.53)	4778	(16.0, 1.55)
HS19	841	(13.9, 0.313)	1561	(14.1, 0.793)	353	(13.7, 0.111)
HS23	729	(0.968, 0.968)	1745	(0.996, 1.00)	1497	(0.996, 0.996)
HS31	30	(0.463, 1.16, 0.0413)	30	(0.459, 1.16, 0.0471)	30	(0.458, 1.16, 0.0474)
HS36	76	(20, 11, 15.6)	370	(20, 11, 15.2)	586	(19.8, 11, 15.3)
HS37	329	(27.5, 13.3, 10.4)	487	(20.6, 13.0, 13.0)	397	(23.5, 12.9, 11.5)
HS41	52	(0.423, 0.350, 0.504, 1.70)	264	(0.796, 0.223, 0.424, 2)	1364	(0.599, 0.425, 0.296, 2.00)
HS65	1468	(3.62, 3.55, 4.91)	1585	(3.58, 3.69, 4.68)	874	(3.71, 3.59, 4.64)

Table III. Results with the soft optimality theorem for some Hock–Schittkowski problems ( $x$  rounded to three significant digits for display)

Since the optimal function values on the feasible sets are known for the test functions, we stop when a point satisfying (10) and (11) has been found (which need not be the current best point of the merit function). For  $\sigma = 0.05, 0.01,$  and  $0.005$ , one job is computed (with the same set of input points for the initial call). In Table II, the numbers of the Hock–Schittkowski problems, their dimensions, their numbers  $m_i$  of inequality constraints, their numbers  $m_e$  of equality constraints, their global minimizers  $x^*$ , and the vectors  $b$  are listed. In Table III, the results for the three values of  $\sigma$  are given, i.e., the points  $x$  obtained and the corresponding numbers  $nf$  of function evaluations.

## 10. CONCLUDING REMARKS

The SNOBFIT package, a MATLAB implementation of the algorithm described available at <http://www.mat.univie.ac.at/~neum/software/snobfit/v2/>, successfully solves many low-dimensional global optimization problems using no gradients and only a fairly small number of function values, which may be noisy and possibly undefined. Parallel evaluation is possible and easy. Moreover, safeguards are taken to make the algorithm robust in the face of various issues.

The numerical examples show that SNOBFIT can handle successfully noisy functions and hidden constraints and that the soft optimality approach for soft constraints is feasible. Several undocumented variants of a number of details in SNOBFIT were tried out. The present version turned out to be the best one; in particular, it improves a version put on the web in 2004. Deleting any of the features degrades

performance, usually substantially on some problems. Thus all features are necessary for the success of the algorithm. Numerical experience with further tests suggests that SNOBFIT should be used primarily with problems of dimension  $\leq 10$ .

In contrast to many algorithms that only provide local minimizers, need an excessive number of sample points, or fail in the presence of noise, this makes the algorithm very suitable for tuning parameters in computer simulations for low accuracy computation (such as in the solution of three-dimensional partial differential equations) or the calibration of expensive experiments where function values can be measured only.

An earlier version of SNOBFIT was used successfully for a real life constrained optimization application involving the calibration of nanoscale etching equipment with expensive measured function values, in which most of the problems mentioned in the introduction were present.

An application to sensor placement optimization is given in [Guratzsch 2007].

#### ACKNOWLEDGMENT

The major part of the SNOBFIT package was developed within a project sponsored by IMS Ionen Mikrofabrikations Systeme GmbH, Wien, whose support we gratefully acknowledge. We'd also like to thank Erich Dolejsi for his help with debugging an earlier version of the program.

#### REFERENCES

- ANDERSON, E. J. AND FERRIS, M. C. 2001. A direct search algorithm for optimization of expensive functions by surrogates. *SIAM J. Optim.* 11, 837–857.
- BOOKER, A. J., DENNIS, JR., J. E., FRANK, P. D., SERAFINI, D. B., TORCZON, V., AND TROSSET, M. W. 1999. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optim.* 17, 1–13.
- CARTER, R. G., GABLONSKY, J. M., PATRICK, A., KELLEY, C. T., AND ESLINGER, O. J. 2001. Algorithms for noisy problems in gas transmission pipeline optimization. *Optim. Eng.* 2, 139–157.
- CHOI, T. D. AND KELLEY, C. T. 2000. Superlinear convergence and implicit filtering. *SIAM J. Optim.* 10, 1149–1162.
- CONN, A. R., SCHEINBERG, K., AND TOINT, P. L. 1997. Recent progress in unconstrained nonlinear optimization without derivatives. *Math. Programming* 79B, 397–414.
- DALLWIG, S., NEUMAIER, A., AND SCHICHL, H. 1997. GLOPT – a program for constrained global optimization. In *Developments in Global Optimization*, I. M. Bomze, T. Csendes, R. Horst, and P. M. Pardalos, Eds. Nonconvex Optimization and its Applications 18. Kluwer, Dordrecht, 19–36.
- DEUFLHARD, P. AND HEINDL, G. 1979. Affine invariant convergence theorems for Newton's method and extensions to related methods. *SIAM J. Numer. Anal.* 16, 1–10.
- ELSTER, C. AND NEUMAIER, A. 1995. A grid algorithm for bound constrained optimization of noisy functions. *IMA J. Numer. Anal.* 15, 585–608.
- FIACCO, A. AND MCCORMICK, G. P. 1990. *Sequential Unconstrained Minimization Techniques*. Classics in Applied Mathematics 4. SIAM, Philadelphia.
- GURATZSCH, R. F. 2007. Sensor placement optimization under uncertainty for structural health monitoring systems of hot aerospace structures. Ph.D. thesis, Dept. Civil Engineering, Vanderbilt University, Nashville.
- HIRSCH, M. J., MENESES, C. N., PARDALOS, P. M., AND RESENDE, M. G. C. 2007. Global optimization by continuous grasp. *Optim. Lett.* 1, 201–212.

- HOCK, W. AND SCHITTKOWSKI, K. 1981. *Test Examples for Nonlinear Programming*. Lecture Notes in Economics and Mathematical Systems 187. Springer-Verlag, Berlin Heidelberg New York.
- HUYER, W. AND NEUMAIER, A. 1999. Global optimization by multilevel coordinate search. *J. Global Optim.* 14, 331–355.
- JONES, D. R. 2001. A taxonomy of global optimization methods based on response surfaces. *J. Global Optim.* 21, 345–383.
- JONES, D. R., PERTTUNEN, C. D., AND STUCKMAN, B. E. 1993. Lipschitzian optimization without the Lipschitz constant. *J. Optim. Theory Appl.* 79, 157–181.
- JONES, D. R., SCHONLAU, M., AND WELCH, W. J. 1998. Efficient global optimization of expensive black-box functions. *J. Global Optim.* 13, 455–492.
- MCKAY, M. D., BECKMAN, R. J., AND CONOVER, W. J. 1979. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* 21, 239–245.
- NOCEDAL, J. AND WRIGHT, S. J. 1999. *Numerical Optimization*. Springer Series in Operations Research. Springer, Berlin.
- OWEN, A. B. 1992. Orthogonal arrays for computer experiments, integration and visualization. *Statistica Sinica* 2, 439–452.
- OWEN, A. B. 1994. Lattice sampling revisited: Monte Carlo variance of means over randomized orthogonal arrays. *Ann. Stat.* 22, 930–945.
- POWELL, M. J. D. 1998. Direct search algorithms for optimization calculations. *Acta Numerica* 7, 287–336.
- POWELL, M. J. D. 2002. UOBYQA: unconstrained optimization by quadratic approximation. *Math. Program.* 92B, 555–582.
- SACKS, J., WELCH, W. J., MITCHELL, T. J., AND WYNN, H. P. 1989. Design and analysis of computer experiments. With comments and a rejoinder by the authors. *Statist. Sci.* 4, 409–435.
- SCHEINBERG, K. 2003. *Manual for Fortran software package DFO v2.0*. [https://projects.coin-or.org/Dfo/browser/trunk/manual2\\_0.ps](https://projects.coin-or.org/Dfo/browser/trunk/manual2_0.ps).
- SCHONLAU, M. 1997. Computer experiments and global optimization. Ph.D. thesis, University of Waterloo, Waterloo, Ontario, Canada.
- SCHONLAU, M. 1997–2001. *SPACE Stochastic Process Analysis of Computer Experiments*. <http://www.schonlau.net>.
- SCHONLAU, M., WELCH, W. J., AND JONES, D. R. 1998. Global versus local search in constrained optimization of computer models. In *New Developments and Applications in Experimental Design*, N. Flournoy, W. F. Rosenberger, and W. K. Wong, Eds. Institute of Mathematical Statistics Lecture Notes – Monograph Series 34. Institute of Mathematical Statistics, Hayward, CA, 11–25.
- TANG, B. 1993. Orthogonal array-based Latin hypercubes. *J. Amer. Statist. Assoc.* 88, 1392–1397.
- TROSSET, M. W. AND TORCZON, V. 1997. Numerical optimization using computer experiments. Tech. Rep. 97-38, ICASE.
- VANDEN BERGHEN, F. AND BERSINI, H. 2005. CONDOR, a new parallel, constrained extension of Powell’s UOBYQA algorithm: experimental results and comparison with the DFO algorithm. *J. Comput. Appl. Math.* 181, 157–175.
- WELCH, W. J., BUCK, R. J., SACKS, J., WYNN, H. P., MITCHELL, T., AND MORRIS, M. D. 1992. Screening, predicting, and computer experiments. *Technometrics* 34, 15–25.