Finding optimal algorithmic parameters using a mesh adaptive direct search

Charles Audet*

Charles.Audet@gerad.ca, www.gerad.ca/Charles.Audet

Dominique Orban†

Dominique.Orban@gerad.ca, www.mgi.polymtl.ca/Dominique.Orban

GERAD and École Polytechnique de Montréal
Département de Mathématiques et de Génie Industriel
C.P. 6079, Succ. Centre-ville, Montréal (Québec), H3C 3A7 Canada

December 1, 2004

Abstract: The objectives of this paper are twofold; we first demonstrate the flexibility of the mesh adaptive direct search (MADS) in identifying locally optimal algorithmic parameters. This is done by devising a general framework for parameter tuning. The framework makes provision for surrogate objectives. Parameters are sought so as to minimize some measure of performance of the algorithm being fine-tuned. This measure is treated as a black-box and may be chosen by the user. Examples are given in the text.

The second objective illustrates this framework by specializing it to the identification of locally optimal trust-region parameters in unconstrained optimization. Parameters are identified that minimize, in a certain sense, the computational time or the number of function evaluations required to solve a set of problems from the CUTEr collection. Each function call may take several hours and may not always return a predictable result. A surrogate function, taylored to the experiment at hand, is used to guide the MADS towards a local solution.

*Supported by FCAR grant NC72792, NSERC grant 239436-01, AFOSR F49620-01-1-0013 and ExxonMobil R62291.

†Supported by NSERC grant 299010-04

The parameters thus identified differ from traditionally used values, and are used to solve problems from the CUTEr collection that remained otherwised unsolved in a reasonable time using traditional values.

Key words: Trust-region methods, unconstrained optimization, mesh adaptive direct search algorithms, black-box optimization, surrrogate functions, parameter estimation.

1 Introduction

Most algorithms, be it in optimization or any other field, depend more or less critically on a number of parameters. Some parameters may be continuous, such as an initial trust-region radius, or a scalar that dictates the precision at which a subproblem needs to be solved. These parameters may be required to remain between two, possibly infinite bounds, or may be constrained in a more complex way. They may also be discrete, such as the maximal number of iterations or the number of banned directions in a taboo search heuristic, or even categorical, such as a boolean indicating whether exact or inexact Hessian is used, or which preconditioner to use. The overall behaviour of the algorithm is influenced by the values of these parameters. Unfortunately, for most practical cases, it remains unclear how a user should proceed to determine *good*, let alone optimal, values for those parameters. We devise a framework for fine-tuning such parameters which is general enough to encompass most numerical algorithms from engineering, numerical analysis and optimization. The design of the framework relies on the observation that measures of *performance* can be derived from the dependency of the algorithm on its parameters. These measures are context- and problem-dependent and for this reason, we wish to treat them as black boxes in the remainder of this paper. We shall however give examples in the context of a particular application.

An optimization problem is formulated where such a measure of performance is minimized as a function of the parameters, over a domain of acceptable values. We use the recent mesh adaptive direct search (MADS) [3] class of nonsmooth optimization algorithms to solve this problem. MADS may use a *surrogate* objective function to guide its search strategy. A surrogate function is a simplification of the real objective function that possesses similar behaviour, but is believed to be less costly to evaluate, or easier to manipulate, in some sense. For example, suppose that the evaluation of the objective function requires a costly numeri-

cal solution of a subproblem. Then, a natural surrogate might use only a crude approximation of its solution. Surrogate functions may also be constructed by approximation surfaces, such as Kriging models. The reader is invited to consult [5] for a general framework for the use of surrogates in an optimization context. With an application in mind, we shall restrict ourselves in the present framework to the case where Ω is polyhedral.

As an illustration of how to use this framework, we address the study of standard parameters present in trust-region algorithms for unconstrained optimization and try to identify some locally optimal values. We note that optimality is not a well-defined concept in this case and the quality of a final solution depends on the underlying method used to minimize the measure of performance. In this illustration, the quality of a set of parameters is measured by the overall computational time or the overall number of function calls required by a trust-region algorithm to solve to a prescribed precision a significant number of test problems. In the numerical tests presented, the test problems originate from the CUTEr [17] collection. We therefore formulate an optimization problem, where each evaluation of the objective function requires solving a collection of test problems. The objective function is therefore time-consuming to compute, is highly nonlinear and no derivative is available or even proved to exist. Moreover, evaluating the objective twice with the same arguments may lead to slightly different function values since the computational time is influenced by the current machine load and fluctuates with network activity. In our context, a surrogate function is obtained by applying the same trust-region algorithm to a set of easier problems.

The trust-region parameters obtained by MADS allow the solution of problems which remained otherwise unsolved in reasonable time by the trust-region method.

Related work on locally optimal parameter identification in a similar trust-region framework is presented in [15], where the parameter space is discretized and a thorough search is carried out. However, even for a modest number of discretized values, devising a mechanism able to compare so many variants of an algorithm becomes an issue. In recent years, performance profiles [12] have been extensively used to compare algorithms, but it remains unclear how to use them to efficiently compare more than five or six. We circumvent this issue in the present paper by delegating this task to another optimization algorithm.

The paper is structured as follows. In Section 2, we describe a specific implementation of the MADS class of algorithms, and highlight the main convergence results. We also describe how a surrogate function can be used inside this algorithm. Section 3 describes a standard trust-region algorithm, and discusses the four algorithmic parameters which will be fine-tuned. In Section 4, we apply MADS to identify locally optimal values for these parameters. Results are presented in §5, discussed in §6 and we give concluding remarks in §7.

2 Mesh adaptive direct search algorithms

A general optimization problem may be stated as

$$\min_{p\in\Omega} \quad \psi(p). \tag{1}$$

with $\psi : \Omega \subseteq \mathbb{R}^\ell \to \mathbb{R} \cup \{+\infty\}$. The nature and structure of the function ψ and the domain Ω limit the type of algorithms that may be used to attempt to solve this problem. Global optimization is possible when the problem structure is sufficiently rich and exploitable, and when the problem size is reasonable. However, global optimization is frequently out of reach in acceptable time, and, under appropriate smoothness assumptions, we are content with algorithms providing a first-order critical solution. For example, when ψ is continuously differentiable over Ω, an appropriate variant of Newton's method combined with a globalizing scheme would yield a critical point under reasonable assumptions. When ψ is non-differentiable, discontinuous or fails to evaluate for some values of its argument $p \in \Omega$, problem (1) cannot be satisfactorily approached by such a method. This is often the case when evaluating the objective entices running a computer code. In order to evaluate, the code may, for instance, have to solve a coupled system of differential equations, and may for some internal reasons fail to return a meaningful value. In this case, the function value is simply considered to be infinite. In a helicopter rotor blade design application [4], the objective function failed to return a value two times out of three. Randomness may also be present in the evaluation of a function, as in [26], where two evaluations of ψ at the same point p return slightly different values. In this less optimistic case, the best optimality condition which can be hoped for is to find a *refined point*. We shed light on this concept in §2.3.

The MADS class of algorithms, introduced in [3], is designed for such nonsmooth optimization problems, and its convergence properties rely on Clarke's

nonsmooth calculus [6]. MADS extends the generalized pattern search [2, 34] (GPS) class of algorithms by allowing a more general exploration of the space of variables.

2.1 A general overview of MADS with a surrogate function

The MADS algorithm attempts to locate a minimizer of the function ψ over Ω by means of the *barrier* function

$$\psi_\Omega(p) = \begin{cases} +\infty & \text{if } p \notin \Omega \\ \psi(p) & \text{otherwise.} \end{cases} \tag{2}$$

We will refer to ψ as being the *truth function* or, sometimes, simply the truth.

As is usual in nonlinear programming, a second function, playing the role of a *model*, may be used to guide the algorithm and steer the iterates towards promising regions. Suppose that $\sigma : \mathbb{R}^\ell \to \mathbb{R} \cup \{+\infty\}$ is a given function which may be used as a model for ψ. In the context of non-differentiable optimization and MADS-type methods, a model is often referred to as a *surrogate*. The surrogate may be an approximation to the truth, or it may be a simplified function whose behaviour is similar to that of the truth. An important feature of the surrogate is that it must be *easier* to evaluate than the truth, in some sense, be it less costly in terms of time or other. The previous sentences are left intentionally vague, since the formal convergence analysis is independent of the quality of the approximation of ψ by σ. However, in practice, appropriate surrogates may improve the convergence speed. A *barrier* surrogate σ_Ω is defined similarly to (2).

MADS is an iterative algorithm, where each iteration essentially consists of two steps. First, a global exploration of the space of variables is conducted in hopes of improving the *incumbent* $p_k \in \mathbb{R}^\ell$ at iteration k. This flexible stage, called the SEARCH step, returns a set of candidates but the truth function need not be evaluated at all of them. To decide at which of these the truth ψ_Ω will be evaluated, the value of the barrier surrogate function σ_Ω is computed at each of them and they are subsequently ordered in increasing function values. We may thus consider without loss of generality that the set of candidates has the form $\mathcal{L} = \{q^1, q^2, \ldots, q^m\}$ and satisfies $\sigma_\Omega(q^1) \leq \sigma_\Omega(q^2) \leq \ldots \leq \sigma_\Omega(q^m)$. A candidate $q^j \in \mathcal{L}$ will be considered promising if $\sigma_\Omega(q^j) \leq \sigma_\Omega(p_k) + v|\sigma_\Omega(p_k)|$, where $v \in \mathbb{R}_+ \cup \{+\infty\}$ is a threshold supplied by the user. Candidates which are not promising are eliminated from the SEARCH list.

The truth function is then evaluated at the promising candidates in $\mathcal{L}$. This can be done in an opportunistic way, evaluating $\psi_\Omega(q^i)$ with increasing values of i and terminating the process as soon as $\psi_\Omega(q^i) < \psi_\Omega(p_k)$. In this case, an improved incumbent is found and we set $p_{k+1} = q^i$.

Secondly, in the event where the SEARCH fails to identify an improved iterate, a local exploration about p_k is performed. This is called the POLL step. Again, the surrogate function is used to order the trial points. The convergence theory prohibits pruning candidates given by the POLL step. The convergence analysis relies mostly on this step and it must obey stricter rules. In GPS, the POLL is confined at each iteration to a fixed finite set of directions, while the set of directions for the MADS POLL may vary at each iteration, and in the limit the union of these poll directions over all iterations is dense in the whole space.

2.2 An iteration of a MADS algorithm

We now present a lower-level description of the method. The reader is invited to consult [3] for a complete algorithmic description and a detailed convergence analysis. The version presented in the present work is specialized for our purposes and some algorithmic choices were made. As a consequence, this allows the use of much lighter notation.

Let $S_0 \subset \Omega$ denote a finite set of initial guesses, provided by the user (a strategy exploiting the surrogate to determine S_0 is presented in §4.4). Set p_0 to be the best initial guess in S_0. A MADS algorithm is constructed in such a way that any trial point generated by the SEARCH or POLL step is required to lie on the *current mesh*, whose coarseness is governed by a *mesh size parameter* $\Delta_k \in \mathbb{R}_+$. The mesh is formally defined in Definition 2.1.

Definition 2.1 *At iteration k, the current mesh is defined to be the union*

$$M_k = \bigcup_{p \in S_k} \left\{ p + \Delta_k z : z \in \mathbb{Z}^{2\ell} \right\},$$

where S_k is the set of points where the objective function ψ has been evaluated by the start of iteration k and $\mathbb{Z}$ denotes the set of integers.

The objective of the iteration is to find a trial mesh point with a lower objective function value than the current incumbent value $\psi_\Omega(p_k)$. Such a trial point is called an *improved mesh point*, and the iteration is called a *successful iteration*. There are no sufficient decrease requirements: any improvement in

ψ_Ω leads to a successful iteration. The iteration is said to be *unsuccessful* if no improved point is found.

Most of the flexibility in the MADS algorithm lies in the SEARCH step. On the other hand, the POLL step evaluates ψ_Ω at 2ℓ trial points surrounding the current incumbent. These neighbouring points are called the *frame*, and are denoted by :

$$F_k \quad = \quad \{p_k \pm \Delta_k d : d \in D_k\} \ , \tag{3}$$

where $D_k = \{d^1, d^2, \ldots, d^\ell\}$ is a basis in $\mathbb{Z}^\ell$. To ensure convergence, the radii of successive frames must converge to zero at a slower rate than the mesh size parameter. The construction of the basis D_k proposed in [3] ensures that

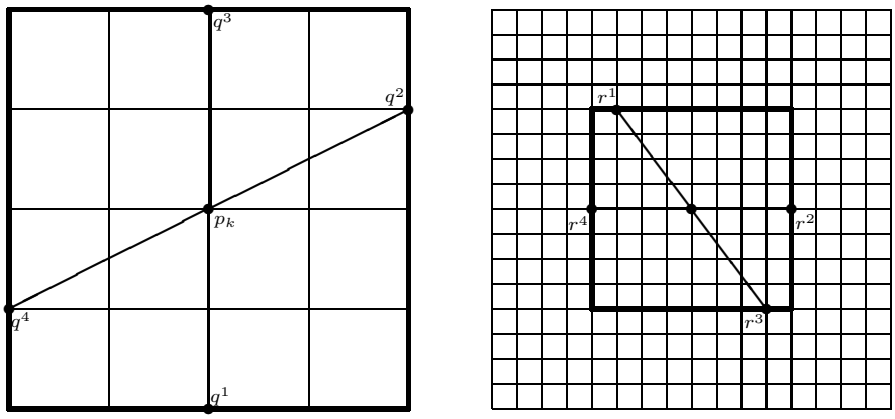
$$\|\Delta_k d\|_\infty = O(\sqrt{\Delta_k}) \quad \text{for all } d \in D_k. \tag{4}$$

Both the SEARCH (when $k \geq 1$) and the POLL may be opportunistic: the iteration may be terminated as soon as an improved mesh point is detected.

Figure 1 shows an example of two consecutive frames in $\mathbb{R}^2$. The figure on the left represents iteration k. The mesh M_k is represented by the intersection of all lines. Suppose that $\Delta_k = \frac{1}{2}$. The thick lines delimits the frame, i.e., the region in which all four POLL points must lie. In this example, the frame points q_1 and q_3 are obtained by the randomly generated direction $d_1 = (0, -2)$, and q_2 and q_4 are obtained by $d_2 = (2, 1)$. The figure on the right displays a possible frame if iteration k is unsuccessful. The mesh is finer at iteration $k+1$ than it was at iteration k, and there are more possibilities in choosing a frame. More precisely, $\Delta_{k+1} = \frac{1}{4}\Delta_k = \frac{1}{8}$ and, as in (4), the distance from the boundary of the frame to the incumbent is reduced by a factor of 2 ; $\|r^i - p_{k+1}\|_\infty = \sqrt{\frac{1}{4}}\|q^j - p_k\|_\infty$ for all i, j. The directions used to construct the frame points r_1 and r_3 are $d_1 = (-3, 4)$, and the directions for q_2 and q_4 are $d_2 = (4, 0)$. Moreover, suppose that iteration $k+1$ is successful at q_3. Then iteration $k+2$ would be initiated at $p_{k+1} = q_3$ with a smaller mesh size parameter $\Delta_{k+1} = \frac{1}{2}$.

When the POLL step fails to generate an improved mesh point then the frame is called a *minimal frame*, and the frame center p_k is said to be a *minimal frame center*. At iteration k, the rule for updating the mesh size parameter is

$$\Delta_{k+1} = \begin{cases} \Delta_k/4 & \text{if } p_k \text{ is a minimal frame center,} \\ 4\Delta_k & \text{if an improved mesh point is found, and } \Delta_k \leq \frac{1}{4}, \\ \Delta_k & \text{otherwise.} \end{cases} \tag{5}$$



Iteration k (unsuccessful) Iteration $k+1$ with $p_{k+1} = p_k$

Figure 1: Two consecutive frames F_k, F_{k+1} in $\mathbb{R}^2$ with $\Delta_k = \frac{1}{2}$, $\Delta_{k+1} = \frac{1}{8}$.

The above description is summarized in Algorithm 2.1.

Given the functions ψ and σ, the only steps that are not completely defined in Algorithm 2.1 are the selection of the set of initial guesses S_0 and the SEARCH strategy. Particular choices in the framework of an application are discussed in §4.4.

2.3 Convergence properties of MADS

Since we are concerned in the present work with bound constraints on parameters, the convergence analysis of [3] greatly simplifies. We present here the specialized results. The proofs of these results may all be found in [3]. The analysis relies on Assumption 1.

Assumption 1 *At least one initial guess $p_0 \in S_0 \subseteq \Omega$ has finite $\psi(p_0)$ value and all iterates $\{p_k\}$ produced by Algorithm 2.1 lie in a compact set.*

The mechanism of Algorithm 2.1 ensures the following property.

Lemma 2.2 *The sequence of mesh size parameters satisfies*

$$\liminf_{k \to +\infty} \Delta_k \;=\; 0.$$

Moreover, since Δ_k shrinks only at minimal frames, it follows that there are infinitely many minimal frame centers.

Definition 2.3 specifies important subsequences of iterates and limit directions.

Algorithm 2.1: [**A** MADS **algorithm**]

Step 0. [Initialization] Let S_0 be given, $p_0 \in \arg\min\{\psi(p) : p \in S_0\}$, $\Delta_0 > 0$ and $v \in \mathbb{R}_+ \cup \{+\infty\}$. Set the iteration counter $k = 0$, and go to Step 1.

Step 1. [SEARCH step] Let $\mathcal{L} = \{q^1, q^2, \ldots q^m\} \subset M_k$ be a finite (possibly empty) set of mesh points such that $\sigma_\Omega(q^i) \leq \sigma_\Omega(q^j) \leq \sigma_\Omega(p_k) + v|\sigma_\Omega(p_k)|$ when $1 \leq i < j \leq m$.

Let i_0 be the smallest $i \in \{1, \ldots, m\}$ such that $\psi_\Omega(q^i) < \psi_\Omega(p_k)$.
If no such index i_0 exists, go to Step 2.
Otherwise, declare k successful, set $p_{k+1} = q^{i_0}$ and go to Step 3.

Step 2. [POLL step:] Construct the frame $F_k = \{q^1, q^2, \ldots q^{2\ell}\}$ as in (3) and order the points so that $\sigma_\Omega(q^i) \leq \sigma_\Omega(q^j)$ when $1 \leq i < j \leq 2\ell$.

Let i_0 be the smallest $i \in \{1, \ldots, 2\ell\}$ such that $\psi_\Omega(q^i) < \psi_\Omega(p_k)$.
If no such index i_0 exists, declare k unsuccessful and go to Step 3.
Otherwise, declare k successful, set $p_{k+1} = q^{i_0}$ and go to Step 3.

Step 3. [Parameter update] If iteration k was declared unsuccessful, then p_k is a minimal frame center and p_{k+1} is set to p_k. Otherwise p_{k+1} is an improved mesh point.

Update Δ_{k+1} according to (5). Increase $k \leftarrow k + 1$ and go back to Step 1.

Definition 2.3 *A subsequence of the* MADS *iterates consisting of minimal frame centers, $\{p_k\}_{k\in K}$ for some subset of indices K, is said to be a* refining subsequence *if $\{\Delta_k\}_{k\in K}$ converges to zero. Any accumulation point of $\{p_k\}_{k\in K}$ will be called a refined point.*

Let $\{p_k\}_{k\in K}$ be a convergent refining subsequence, with refined point $\hat{p}$, and let v be any accumulation point of the set $\{\frac{d_k}{\|d_k\|} : p_k + \Delta_k d_k \in \Omega, k \in K\} \subset \mathbb{R}^\ell$. Then v is said to be a refining direction for $\hat{p}$.

Note that under Assumption 1, there always exists at least one convergent refining subsequence, one refining point and a positive spanning set of refining directions. We first present a basic result on refining directions, based on the concept of Clarke generalized directional derivative [6, 19] in the tangent cone (recall that the tangent cone to Ω at some $\hat{p} \in \Omega$, noted $T_\Omega(\hat{p})$, is defined to be the closure of the set $\{\mu(v - \hat{p}) : \mu \in \mathbb{R}_+,\ v \in \Omega\}$).

Theorem 2.4 *Let ψ be Lipschitz near a limit $\hat{p} \in \Omega$ of a refining subsequence,*

and $v \in T_\Omega(\hat{p})$ be a refining direction for $\hat{p}$. Then the Clarke generalized directional derivative $\psi^\circ(\hat{p}; v)$ of ψ at $\hat{p}$ in the direction v is nonnegative, i.e.,

$$\psi^\circ(\hat{p}; v) \equiv \limsup_{\substack{y \to \hat{p},\ y \in \Omega, \\ t \downarrow 0,\ y + tv \in \Omega}} \frac{\psi(y + tv) - \psi(y)}{t} \geq 0.$$

The next result states that MADS produces a limit point that satisfies some necessary optimality conditions. The results are presented in a hierarchical way; the weaker conclusion results from the weaker assumption on ψ which is lower semicontinuity. In this case, we obain a locally minimal function value along the directions explored by the refining subsequence. The main result is that the Clarke derivatives of ψ at a refined point $\hat{p}$ are nonnegative for all directions in the tangent cone.

Theorem 2.5 *Let $\hat{p} \in \Omega$ be a refined point of a refining subsequence $\{p_k\}_{k \in K}$, and assume that the set of refining directions for $\hat{p}$ is dense in $T_\Omega(\hat{p})$.*

- *If ψ is lower semicontinuous at $\hat{p}$, then $\psi(\hat{p}) \leq \lim_{k \in K} \psi(p_k)$,*
- *If ψ is Lipschitz near $\hat{p}$, then $\psi^\circ(\hat{p}, v) \geq 0$ for every $v \in T_\Omega(\hat{p})$,*
- *If ψ is Lipschitz near $\hat{p} \in int(\Omega)$, then $0 \in \partial\psi(\hat{p}) \equiv \{s \in \mathbb{R}^\ell : \psi^\circ(\hat{x}; v) \geq v^T s,\ \forall v \in \mathbb{R}^\ell\}$,*
- *If ψ is strictly differentiable [20] at $\hat{p}$, then $\hat{p}$ is KKT stationary point of ψ over Ω.*

The density assumption in Theorem 2.5 is not restrictive ; the LTMADS implementation ensures that it is true with probability one [3] when the sequence of iterates produces by the algorithm converges.

Note that the above convergence results rely only on the POLL step, and are independent of the surrogate function and of the SEARCH step. Furthermore, even though Algorithm 2.1 is applied to ψ_Ω instead of ψ, the convergence results are linked to the local smoothness of ψ and not ψ_Ω, which is obviously discontinuous on the boundary of Ω.

3 Trust-region methods

To successfully tackle a smooth nonlinear nonconvex programming problem from a remote starting guess, the iteration must often be embedded into a *globalization* method. The two most popular globalization methods are the linesearch and the trust region. Their philosophies may be seen as dual; a linesearch strategy computes a step length along a predetermined direction, while a trust-region strategy considers all acceptable directions but limits the maximal step length. In this section, we briefly review the latter.

3.1 A basic trust-region algorithm

Trust-region methods appear to date back to a 1944 paper in which they were used to solve nonlinear least-squares problems [21]. For similar purposes, they were independently used by Morrison [25] and Marquardt [22]. Later, Goldfeldt, Quandt and Trotter [13] introduced updating rules for the size of the region, a crucial step towards modern trust-region methods. In 1970, Powell [27] proved global convergence of a particular trust-region algorithm. Different terminologies are used in the community, but substantial standardization appears as a result of the survey [24]. Trust-region methods now form one of the most popular globalization schemes and are often praised for their robustness and flexibility. They are used throughout optimization, from regularization problems to derivative-free and interior-point methods. We trust that the reader's appetite for lists of references together with more historical notes and thorough theoretical developments across the whole optimization spectrum will be satisfied by the recent book [9].

For simplicity and because our later numerical tests will only concern such problems, assume we wish to solve the unconstrained programming problem

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{6}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a twice-continuously differentiable function. For problem (6) to be well defined, we assume throughout that f is bounded below. The philosophy of trust-region methods is to assume that f might be highly nonlinear and/or costly to evaluate. At iteration k, instead of manipulating f directly, f is replaced by a suitable local model m_k which is easier and cheaper to evaluate. However, from the very nature of a local model, it might not be wise to trust that m_k accurately represents f far from the current iterate x_k. A region $\mathcal{B}_k \subset \mathbb{R}^n$,

referred to as the *trust region*, is therefore defined around x_k to represent the extent to which m_k is believed to reasonably model f. The trust region is defined as the ball

$$\mathcal{B}_k \equiv \{x_k + s \in \mathbb{R}^n \ : \ \|s\| \leq \delta_k\},$$

where $\delta_k > 0$ is the current trust-region radius and $\|\cdot\|$ represents any norm on $\mathbb{R}^n$. To simplify the exposition, we choose the Euclidean norm but other choices are acceptable [9].

Instead of applying a procedure to minimize f starting from x_k, the model m_k is approximately minimized within $\mathcal{B}_k$. If the decrease thus achieved is sufficient and if the agreement between f and m_k at the trial point is satisfactory, the step is accepted and the radius δ_k is possibly increased. Otherwise, the step is rejected and the radius is decreased. This last option indicates that m_k might have been trusted in too large a neighbourhood of x_k.

Global convergence of trust-region schemes is ensured by mild assumptions on m_k and on the decrease that should be achieved at each iteration. In practice, one of the most popular models is the quadratic model

$$m_k(x_k + s) = f(x_k) + \nabla f(x_k)^T s + \frac{1}{2} s^T H_k s,$$

where $H_k = \nabla^2 f(x_k)$, or a symmetric approximation to it. For simplicity, we assume in the remainder of this paper that H_k is the exact Hessian of f at x_k.

Sufficient decrease in the model is established by considering the decrease obtained at two points of $\mathcal{B}_k$. The first is the *Cauchy point* x_k^{C}—the minimizer of m_k along the steepest descent direction $d = -\nabla m_k(x_k)$. The second is the minimizer of m_k along a direction of approximately minimal negative curvature, referred to as the *eigen point* and noted x_k^{E}. Sufficient decrease is achieved if the decrease in m_k is at least a fraction of that obtained at the best of these two points:

$$m_k(x_k) - m_k(x_k + s) \geq \theta \left[m_k(x_k) - \min \left\{ m_k(x_k^{\mathrm{C}}), m_k(x_k^{\mathrm{E}}) \right\} \right], \tag{7}$$

where $0 < \theta < 1$ is a fixed value, independent of the iteration number.

Combining all of the above, a typical trust-region framework for problem (6) may be stated as Algorithm 3.1.

We should alert the reader that the updating rule (8) at Step 4 of Algorithm 3.1 is not the only one used in practice, but most likely the most common one. Other rules involve polynomial interpolation of $\rho_k = \rho(s_k)$ as a function of

Algorithm 3.1: [Basic Trust-Region Algorithm]

Step 0. [Initialization] An initial point $x_0 \in \mathbb{R}^n$ and an initial trust-region radius $\delta_0 > 0$ are given, as well as parameters η_1, η_2, α_1 and α_2 satisfying

$$0 \leq \eta_1 < \eta_2 < 1 \quad \text{and} \quad 0 < \alpha_1 < 1 < \alpha_2.$$

Compute $f(x_0)$ and set $k = 0$.

Step 1. [Step calculation] Define a model $m_k(x_k + s)$ of $f(x_k + s)$ in $\mathcal{B}_k$ and compute a step $s_k \in \mathcal{B}_k$ which satisfies (7).

Step 2. [Acceptance of the trial point] Compute $f(x_k + s_k)$ and

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}.$$

If $\eta_1 \leq \rho_k$, then set $x_{k+1} = x_k + s_k$; otherwise, set $x_{k+1} = x_k$.

Step 3. [Trust-region radius update] Set

$$\delta_{k+1} = \begin{cases} \alpha_1 \|s_k\| & \text{if} \quad \rho_k < \eta_1 \\ \delta_k & \text{if} \quad \eta_1 \leq \rho_k < \eta_2 \\ \max[\alpha_2 \|s_k\|, \delta_k] & \text{if} \quad \eta_2 \leq \rho_k. \end{cases} \tag{8}$$

Increment k by one, and go to Step 1.

the step s_k [11], while others devise more sophisticated functions to obtain the new radius [18, 35].

Requirements on the function f and each model m_k are gathered in Assumption 2.

Assumption 2 *The function f is bounded below and its Hessian matrix remains bounded over a set containing all iterates x_k. The model m_k coincides up to first order with f at x_k, i.e., $m_k(x_k) = f(x_k)$ and $\nabla m_k(x_k) = \nabla f(x_k)$.*

The lengthy subject of how to solve the subproblems at Step 1 of Algorithm 3.1 while ensuring (7) is out of the scope of this paper. We shall however return to this issue in Section 4 and argue that the method we have chosen in our implementation to solve trust-region subproblems ensures this.

3.2 Convergence properties of the basic algorithm

We recall in this section the important global convergence properties of Algorithm 3.1 without proof. The proofs may be found in [9]. Quite remarkably, Assumption 2 is all that is required to prove global convergence of the basic trust-region framework. Note that no assumption is made on *how* the model should be minimized within $\mathcal{B}_k$ but rather, (7) imposes a condition on the *quality* of the resulting trial point.

Step 3 of Algorithm 3.1 is often referred to as the computation of achieved versus predicted reduction. Achieved reduction is the actual reduction in the objective f, defined by $\text{ared}_k = f(x_k) - f(x_k + s_k)$. Predicted reduction is the reduction suggested by the model, defined by $\text{pred}_k = m_k(x_k) - m_k(x_k + s_k)$. The quotient ρ_k is thus simply $\text{ared}_k/\text{pred}_k$. The step s_k will be accepted whenever $\text{ared}_k \geq \eta_1 \text{pred}_k$, an iteration we refer to as *successful*. If additionally, $\text{ared}_k \geq \eta_2 \text{pred}_k$, we shall say that the iteration is *very successful*. Otherwise, it is *unsuccessful*.

Critical to the global convergence is the fact that the difference between the objective and model values at the trial point decreases quadratically with δ_k, i.e., $|\text{ared}_k - \text{pred}_k| \leq \kappa \delta_k^2$, where $\kappa > 0$ is a constant, possibly dependent on k [9, Theorem 6.4.1]. This fact ensures that after a finite number of unsuccessful steps, a successful step will be generated [9, Theorem 6.4.2]. A first result considers the situation where there are only a finite number of successful iterations.

Theorem 3.1 *Suppose that Assumption 2 holds. If there are only finitely many successful iterations, then $x_k = x^*$ for all sufficiently large k where x^* is first-order critical for (6).*

The first stage in the global convergence analysis of Algorithm 3.1 is usually summarized by Theorem 3.2, which addresses the case where an infinite number of successful iterations occur.

Theorem 3.2 *Suppose that Assumption 2 is satisfied. Then*

$$\liminf_{k \to \infty} \|\nabla f(x_k)\| = 0.$$

Theorem 3.2 was first proved by Powell [27] in a framework where $\eta_1 = 0$, i.e., where all trial points are accepted as soon as they produce a decrease in the

objective. This result proves that if $\{x_k\}$ has limit points, at least one of them is critical. In fact, this is as good a convergence result as we can obtain when $\eta_1 = 0$ [36]. The framework of Algorithm 3.1 differs in that it is more demanding on the trial point—*sufficient* reduction must be achieved. This sheds some light on the importance of the value of η_1 in the framework, for as the next result shows, a much stronger conclusion holds in this case.

Theorem 3.3 *Suppose Assumption 2 is satisfied, and that $\eta_1 > 0$. Then*

$$\lim_{k\to\infty} \|\nabla f(x_k)\| = 0.$$

In other words, Theorem 3.3 shows that *all* limit points are first-order critical for (6). The distinction between Theorem 3.2 and Theorem 3.3 was reinforced by the careful example of [36], where it is shown that an algorithm with $\eta_1 = 0$ may very well produce limit points which are not critical.

The importance of the parameters η_1 and η_2, but also α_1 and α_2 of Algorithm 3.1 will be of interest to us in the remainder of this paper. In particular, we shall come back to the issue of reduction versus sufficient reduction.

4 Methodology

An objective of the paper is to address a long standing question of identifying four optimal parameters found in a trust-region update (8), namely η_1, η_2, α_1 and α_2. In this section, we present a general methodology to address this issue.

4.1 A black-box approach to parameter estimation

Suppose that Algorithm $\mathcal{A}$ depends on a set of continuous parameters p restricted to lie in $\Omega \subset \mathbb{R}^\ell$, where ℓ is typically small. Let $\mathcal{P}_O = \{\mathcal{P}_i \mid i \in \mathcal{O}\}$ be a set of $n_\mathcal{O} \geq 1$ problem instances believed to be representative of the class of problems for which Algorithm $\mathcal{A}$ was designed, or to be of particular interest in the context of Algorithm $\mathcal{A}$. Define a function $\psi : \Omega \to \mathbb{R}$ so that for any $p \in \Omega$, $\psi(p)$ is some measure of the performance of Algorithm $\mathcal{A}$ in solving the set of problems $\mathcal{P}_i \in \mathcal{P}_O$ and such that the smaller the value of $\psi(p)$, the better the performance of the algorithm, in a context-dependent sense.

In an optimization context, examples of a function ψ would include the total CPU time required to solve the complete test set, or the cumulative number of

iterations, of function evaluations or the number of problems unsuccessfully solved. In other contexts, any appropriate measure may be used.

The above description qualifies as a black-box optimization problem in the sense that a computer program must in general be run in order to evaluate $\psi(\cdot)$ at a given parameter value $p \in \Omega$. For all allowed values of the parameters p, we seek to minimize a global measure of the performance of Algorithm $\mathcal{A}$. In other words, we wish to solve problem (1).

Problem (1) is usually a small-dimensional nondifferentiable optimization problem with expensive black-box function evaluation. It therefore seems natural to use Algorithm MADS to approach it. As an additional difficulty, evaluating the objective function of (1) twice at the same value of p might produce two sligthly different results.[1]

In the present context, there is a natural way to define a less expensive surrogate function that would have an overall behaviour similar to that of ψ. Let $\mathcal{P}_S = \{\mathcal{P}_j \mid j \in \mathcal{S}\}$ be a set of $n_{\mathcal{S}} \geq 1$ *easy* problems and for any $p \in \Omega$, define $\sigma(p)$ to be the same measure (as with ψ) of performance of Algorithm $\mathcal{A}$ in solving the set $\mathcal{P}_S$ of problems. The quality of an approximation of the behaviour of ψ by the surrogate function σ depends on $n_{\mathcal{S}}$ and on the features of the problems in $\mathcal{P}_S$. The more problems, the better the approximation, but the cost of surrogate evaluations will increase. There therefore is a trade-off between the quality and the cost of a surrogate function. It would thus make sense to include in $\mathcal{P}_S$ problems that are less expensive to solve and possibly, we may choose $\mathcal{S} \subset \mathcal{O}$.

Note that this framework is sufficiently general to encompass algorithmic parameter estimation in almost any branch of applied mathematics or engineering.

4.2 An implementation of Algorithm 3.1

Our implementation of the trust-region method is relatively conventional and relies on the building blocks of the GALAHAD Library for Optimization [16]. Trust-region subproblems are solved by means of the Generalized Lanczos method for Trust Regions GLTR [14, 16]. This method is attractive for its ability to naturally handle negative curvature and to stop at the Steihaug-Toint point, i.e., the intersection of the trust-region boundary with a direction of sufficiently negative

[1] And thus technically, ψ is not a function in the mathematical sense.

curvature [31, 33]. It also ensures satisfaction of (7).

We now discuss the termination criteria of applying the trust-region Algorithm 3.1 on the unconstrained problem

$$(\mathcal{P}_i) \;\equiv\; \left\{ \quad \min_{x\in\mathbb{R}^{n_i}} \quad f_i(x). \right.$$

where $f_i : \mathbb{R}^{n_i} \to \mathbb{R}$, for $i \in \mathcal{O} \cup \mathcal{S}$. The algorithm stops as soon as an iterate x_k satisfies

$$\|\nabla f_i(x_k)\|_2 \leq 10^{-5}.$$

The trust-region algorithm is also terminated when this criterion was not met in the first 1000 iterations. The measure ψ or σ is then set to $+\infty$.

The trust-region subproblem

$$\begin{array}{ll} \min\limits_{s\in\mathbb{R}^n} & \nabla f_i(x_k)^T s + \frac{1}{2} s^T \nabla^2 f_i(x_k) s \\ \text{s.t.} & \|s\|_2 \leq \delta_k \end{array}$$

is successfully solved as soon as $x_k + s$ satisfies

$$\|\nabla f_i(x_k + s)\|_2 \;\leq\; \min\left[\frac{1}{10}, \|\nabla f_i(x_k)\|_2^{1/2}\right] \quad \text{or} \quad \|s\|_2 = \delta_k.$$

The initial trust-region radius was chosen according to

$$\delta_0 \;=\; \max\left(\frac{1}{10}\|\nabla f_i(x_0)\|_2,\; 1\right).$$

4.3 Measures of performance

In our experiments below, several functions $\psi(\cdot)$ are considered, but the evaluation of each of them involves running a computer program which has a given trust-region algorithm for unconstrained optimization solve a series of problems with given values of the parameters. The parameters are $p = (\eta_1, \eta_2, \alpha_1, \alpha_2)$ from Step 4 of Algorithm 3.1, $\ell = 4$ and

$$\Omega = \left\{p \in \mathbb{R}^4 \mid 0 \leq \eta_1 < \eta_2 < 1 \text{ and } 0 < \alpha_1 < 1 < \alpha_2 \leq 10\right\}. \tag{9}$$

Note that Assumption 1 is satisfied since the domain Ω is a full-dimensional polyhedral bounded set. The upper bound on α_2 was introduced on the one hand to satisfy Assumption 1, and on the other hand because it does not appear intuitive, or useful, to enlarge the trust-region by an arbitrarily large factor on very successful iterations. Note the presence of the linear constraint $\eta_1 <$

η_2 in (9). In order to agree with the bound-constrained framework of §2.3, this linear constraint was hardcoded into MADS , leaving only bounds in (9). If a parameter value violates this linear constraint, the resulting value of the objective is infinite.

To ensure that a change in Δ in Algorithm 2.1 is comparable for all four parameters, the latter are scaled using

$$\begin{bmatrix} \tilde{\eta}_1 \\ \tilde{\eta}_2 \\ \tilde{\alpha}_1 \\ \tilde{\alpha}_2 \end{bmatrix} = \begin{bmatrix} 1000 & & & \\ & 100 & & \\ & & 100 & \\ & & & 10 \end{bmatrix} \begin{bmatrix} \eta_1 \\ \eta_2 \\ \alpha_1 \\ \alpha_2 \end{bmatrix}. \tag{10}$$

MADS then works with the variables $(\tilde{\eta}_1, \tilde{\eta}_2, \tilde{\alpha}_1, \tilde{\alpha}_2)$.

Suppose a set of $n_{\mathcal{O}}$ unconstrained problems from the CUTEr [17] collection is chosen. An evaluation of the black-box function $\psi(\cdot)$ is defined by the solution of these problems using Algorithm 3.1 and the current parameter values $p \in \Omega$. The outcome of this evaluation is either a real number—our measure of performance—or an infinite value, resulting from a computer or algorithmic failure in one or more problems. Failures may occur because the maximum number of iterations has been exceeded, because of memory or system-dependent errors or perhaps because of a floating-point exception.

Noticeably, in the context of Algorithm 3.1, the same parameter values

$$p^{\mathrm{C}} = (\eta_1, \eta_2, \alpha_1, \alpha_2) = \left(\frac{1}{4}, \frac{3}{4}, \frac{1}{2}, 2\right) \tag{11}$$

are often recommended in the literature [7, 8, 10, 28, 29, 30]. We shall refer to those as the *classical* parameter values. Our contention is to show that the values (11) are arbitrary and that much better options are available. We use Algorithm MADS to identify them. In our tests, the first trial point considered by MADS is p^{C}. The measures of performance which we choose are the number of function evaluations and the total CPU time. If we denote by $\varphi_i(p)$ and $\tau_i(p)$ the number of function evaluations and CPU time, respectively, which were necessary to solve problem $\mathcal{P}_i$ with the parameters p, we may define

$$\psi_1(p) = \sum_{i \in \mathcal{O}} \varphi_i(p). \tag{12}$$

Note that since we are only considering unconstrained minimization, this measure is equivalent to the number of iterations and is justified in the frequent case where objective function evaluations are computationally costly and dominate

the other internal work of the algorithm. If evaluating the objective function is relatively cheap compared to the algorithm's internal work, which might be the case when the dimension of the problem is large and the linear algebra therefore more expensive, then the overall cpu-time is an obvious choice for measuring performance. For this purpose, we define

$$\psi_2(p) = \sum_{i \in \mathcal{O}} \tau_i(p). \tag{13}$$

From the MADS point of view, each objective function evaluation requires the computation of $n_{\mathcal{O}}$ values : either $\varphi_i(\cdot)$ or $\tau_i(\cdot)$ for $i \in \mathcal{O}$.

4.4 The surrogate function σ

In the present context, a function evaluation consists in solving a list of problems with Algorithm 3.1 and combining the results on each problem into a unique real number. A surrogate function σ, as described in §4.1, was defined through a set of relatively simple unconstrained problems. The objective function on the other hand, was defined by a list of much harder problems. They were chosen as follows.

The trust-region algorithm described in Section 4.2 was run on the 163 unconstrained regular problems from the CUTEr collection, using the default dimensions. From those, some problems failed to be solved for memory reasons and some reached the maximum number of iterations of 1000. Two test lists were extracted from the results. The first consists in those problems for which $n_i < 1000$ and $0.01 \leq \tau_i(p^{\text{C}}) \leq 30$ (measured in seconds), and we shall refer to it as the *surrogate* list $\mathcal{S}$. The surrogate list contains 54 problems of small to moderate dimension, $2 \leq n_i \leq 500$ and such that $\sum_i \tau_i(p^{\text{C}}) = 68.1999$ seconds. In other words, we may expect that running through the whole surrogate list, i.e., evaluating the surrogate, should not take much longer than two minutes. Problems in this list and their characteristics are summarized in Table 5. The second list, or *objective* list $\mathcal{O}$, consists in those problems for which $n_i \geq 1000$ and $\tau_i(p^{\text{C}}) \leq 3600$. This yields a list of 55 problems with $1000 \leq n_i \leq 20000$ and such that $\sum_i \tau_i(p^{\text{C}}) = 13461$ seconds, which amounts to 3 hours, 44 minutes and 35 seconds. The latter duration is the time that *one* objective function evaluation may be expected to take. Problems in this list and their characteristics

are summarized in Table 4. The surrogate functions are simply

$$\sigma_1(p) = \sum_{j \in \mathcal{S}} \varphi_j(p) \quad \text{and} \quad \sigma_2(p) = \sum_{j \in \mathcal{S}} \tau_j(p). \tag{14}$$

A surrogate function plays three important roles in the present context. First, it is used as if it were the real objective function, for the sole purpose of obtaining a better starting point than p^{C} before restarting the procedure with the objective function defined by the objective list. Its purpose is thus to postpone the long computations until a neighbourhood of a local minimizer is reached. No surrogate approximation of σ was used. Starting from p^{C} given by (11), MADS terminates with some solution p^{S}. After having obtained these parameters, we can now apply MADS on the truth function ψ and use the surrogate function σ to guide the algorithm. The set of initial guesses was chosen as $S_0 = \{p^{\text{C}}, p^{\text{S}}\}$.

Secondly, the surrogate is used to order trial points generated by the MADS POLL or SEARCH steps, as described in §2.1. If the surrogate is appropriate, the ordering should produce a sucessful iterate for the real objective function before all directions have been explored.

Finally, the third role of the surrogate is to eliminate from consideration the trial SEARCH points at which the surrogate function value exceeds the user threshold value v, which is in our case set to 0.1.

In the present application, the SEARCH strategy differs from one iteration to another, and goes as follows. When $k = 0$, the SEARCH consists of a 64 points latin hypercube sampling of Ω in hopes of identifying promising basins [23, 32]. At iteration $k \geq 1$, the SEARCH consists in evaluating the surrogate barrier function at 8 randomly generated mesh points.

In addition, the SEARCH step conducts the *dynamic search* described in [3]. It is only called after a successful iteration, and essentially consists in evaluating ψ at the next mesh point in a previously successful direction.

5 Numerical results

All tests were run on a 500 MHz Sun Blade 100 running SunOS 5.8. The implementation of MADS is a C++ package called NOMAD.[2]

[2]May be downloaded from `www.gerad.ca/NOMAD`.

5.1 Improving the initial solution

The first run consisted in applying MADS to the surrogate function σ_2 from the classical parameters p^{C}. Results are reported in Table 1. The first column contains the number of MADS function evaluations required to improve the measure to the value in the second column. The other columns contain the corresponding parameter values.

$\#f$ evals	$\sigma_2(p)$	$\tilde{\eta}_1$	$\tilde{\eta}_2$	$\tilde{\alpha}_1$	$\tilde{\alpha}_2$
1	67.6099	250	75	50	20
3	66.11	538.86719	77.480469	43.765625	17.914062
57	56.94	221.65625	89.175781	39.511719	23.042969
138	56.53	221.65625	89.300781	39.402344	23.136719
139	54.42	221.65625	89.675781	39.074219	23.417969
141	53.88	221.65625	89.675781	39.074219	22.917969
154	53.67	221.65625	90.175781	39.074219	22.667969
194	52.57	221.6875	90.175781	38.996094	22.792969
224	52.43	221.625	90.203125	38.996094	22.792969
289	52.35	221.625	90.207031	38.996094	22.792969

Table 1: Minimization of the surrogate function to improve the initial solution. The measure $\sigma_2(p)$ is in seconds.

MADS stopped after 310 function evaluations as the mesh size parameter Δ_k dropped below the stopping tolerance of 10^{-6}. The best set of parameters

$$p^{\mathrm{S}} = (0.221625, 0.90207031, 0.38996094, 2.2792969)$$

was identified at the 289th evaluation. This strategy allowed to improve the truth initial value from 13461 to 11498.026, i.e., down to 3 hours and 12 minutes.

Note that the value of the surrogate at p^{C} had to be evaluated again during the first iteration and that its differred by approximately 1% from the one we had first obtained §4.4 before building the surrogate. This is an illustration of the non-deterministic aspect of such objective functions.

5.2 Performance profiles

Comparison between the initial and final values of the MADS objective function, i.e., the benchmarking of the trust-region method on a set of nonlinear unconstrained programs for the initial and final values of the parameters, will be

presented using performance profiles. Originally introduced in [12], we briefly recall here how to read them.

Suppose that a given algorithm $\mathcal{A}_i$ from a competing set $\mathcal{A}$ reports a statistic $u_{ij} \geq 0$ when run on problem j from a test set $\mathcal{S}$, and that the smaller this statistic the better the algorithm is considered. Let the function

$$\omega(u, u^*, \alpha) = \begin{cases} 1 & \text{if } u \leq \alpha u^* \\ 0 & \text{otherwise} \end{cases}$$

be defined for all u, u^* and all $\alpha \geq 1$. The *performance profile* of algorithm $\mathcal{A}_i$ is the function

$$\pi_i(\alpha) = \frac{\sum_{j \in \mathcal{S}} \omega(u_{i,j}, u_j^*, \alpha)}{|\mathcal{S}|} \quad \text{with} \quad \alpha \geq 1,$$

where $u_j^* = \min_{i \in \mathcal{A}} u_{ij}$. Thus $\pi_i(1)$ gives the fraction of the number of problems for which algorithm $\mathcal{A}_i$ was the most effective, according to the statistics u_{ij}, $\pi_i(2)$ gives the fraction for which algorithm $\mathcal{A}_i$ is within a factor of 2 of the best, and $\lim_{\alpha \to \infty} \pi_i(\alpha)$ gives the fraction of examples for which the algorithm succeeded.

5.3 Minimizing the total computing time

From the parameter values suggested by the surrogate function in Table 1 used as new starting point, Algorithm 2.1 was restarted using the objective function. The stopping condition this time was to perform a maximum of 150 truth evaluations. Based on the estimate of roughly 3 hours and 45 minutes per function evaluation, this amounts to an expected total running time of just about three weeks. Fragments of the evolution of $\psi_2(\cdot)$ are given in Table 2.

The final iterate

$$p^* = (0.22125, 0.94457031, 0.37933594, 2.3042969)$$

produced by MADS gives a value $\psi_2(p^*) = 10192.305$, and therefore reduces the total computing time to just under 2 hours and 50 minutes ; a reduction of 25% of the computing time. This p^* is clearly in favour of a sufficient decrease condition rather than of $\eta_1 \approx 0$.

The values $\psi_2(p^c)$ and $\psi_2(p^*)$ can be visualized in the profile of Fig. 2 which compares the cpu-time profiles of Algorithm 3.1 applied to the objective list for the initial and final parameter values. The profile of Fig. 3 presents a similar comparison, using the number of function evaluations.

$\#f$ evals	$\psi_2(p)$	$\tilde{\eta}_1$	$\tilde{\eta}_2$	$\tilde{\alpha}_1$	$\tilde{\alpha}_2$
1	11498.026	221.625	90.207031	38.996094	22.792969
5	11241.236	221.375	90.207031	38.871094	22.917969
7	10757.544	221.375	90.207031	38.871094	23.417969
13	10693.04	219.375	90.207031	38.871094	24.417969
40	10691.432	219.25	90.207031	38.871094	24.417969
42	10617.409	219.25	90.207031	38.621094	24.417969
73	10617.409	219.25	90.207031	38.621094	24.417969
74	10279.853	221.25	94.457031	37.996094	23.042969
77	10183.954	221.25	94.457031	37.933594	23.042969
97	10183.954	221.25	94.457031	37.933594	23.042969
98	10195.392	221.25	94.457031	37.933594	23.042969
115	10195.392	221.25	94.457031	37.933594	23.042969
116	10192.305	221.25	94.457031	37.933594	23.042969
142	10192.305	221.25	94.457031	37.933594	23.042969

Table 2: Minimization of the objective function from improved starting point. The measure $\psi_2(p)$ is in seconds.

6 Discussion

The above results must be interpreted in the light of the objective function used in the minimization procedure. Fig. 2 and Fig. 3 result from a minimization of $\psi_2(\cdot)$. Almost certainly, a minimization of $\psi_1(\cdot)$ would have produced a different set of parameters. Moreover, the simplicity of $\psi_1(\cdot)$ and $\psi_2(\cdot)$ is counterbalanced by their disadvantage of computing *global* measures. More sophisticated objectives in the present application could penalize the fact that a particular problem took a long time to fail for some parameter values while for others, failure was quickly detected. Similarly, they do not treat differently problems which are uniformly solved in a fraction of a second for nearly all parameter values and problems whose running time varies with great amplitude. Such effects might, and do, cause MADS to elect against exploring certain regions.

To illustrate this point, we note that the parameter values recommended by MADS differ significantly from those recommended in [15]. In a separate, preliminary, series of tests, the objective function $\sigma_2(\cdot)$ was minimized over (9) without using the scaling (10) and without using surrogates. The final iterate thus produced turned out to be

$$p^{\mathrm{A}} = (0.000008, 0.9906, 0.3032, 3.4021),$$

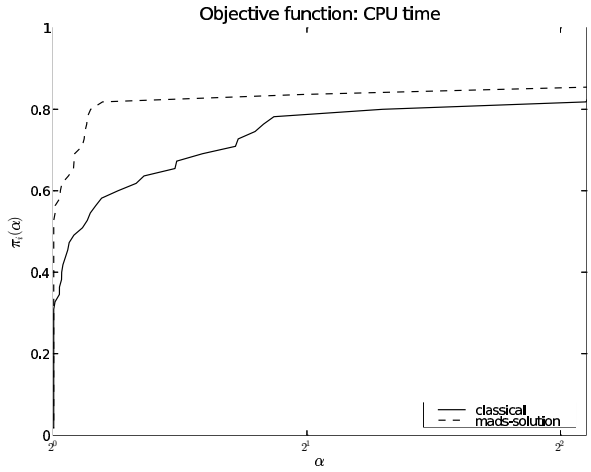


Figure 2: Profile comparing the CPU time required for one evaluation of the MADS objective for the initial and final parameter values.

which is rather close to the recommendations of [15] and seems to indicate that enforcing sufficient descent is not particularly beneficial in practice. The value $\psi_2(p^{\mathrm{A}}) = 13707$ is surprisingly higher than $\psi_2(p^{\mathrm{C}}) = 13461$. However, the corresponding profile appears significantly better than the reference algorithm using p^{C} as illustrated by Fig. 4 and Fig. 5, where we refer to p^{A} as the *alternative* parameter value. Problems with long solution times are gathered in Table 3. The first three of those do not appear to influence the behaviour of MADS by much, as their solution time varies little. Some problems failed to be solved for any values of the parameter. Among those, `GENHUMPS` is particularly detrimental to the measure $\psi_2(p^{\mathrm{A}})$ as the failure takes 10 times longer to be detected than at p^*. Likely, the value p^{A} would produce much better results if `GENHUMPS` were not present. We see nonetheless in the present case that MADS performed its task as it should have and that, perhaps, it is the objective function $\psi_2(\cdot)$ which should take such outliers into account, since the presence of problems like `GENHUMPS` cannot be anticipated.

A phenomenon of a much more optimistic kind is revealed by problem `CRAGGLVY` which could not be solved in less than 1000 iterations using p^{c}, which took 83.3 seconds, but was solved in 20 iterations and 6.59 seconds using p^*.

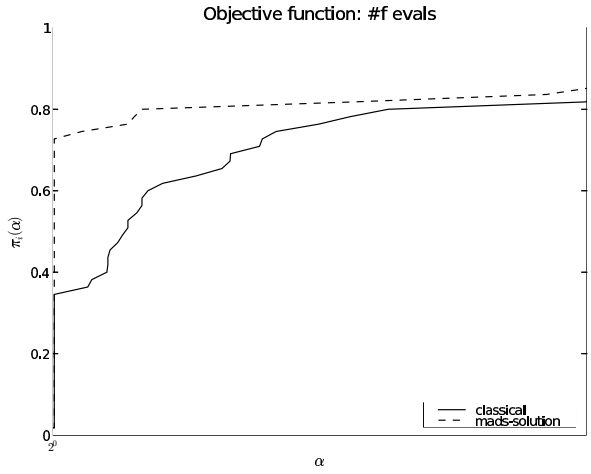


Figure 3: Profile comparing the number of function evaluations required for one evaluation of the MADS objective for the initial and final parameter values.

Problem $\mathcal{P}_i$	$\tau_i(p^{\mathrm{C}})$	$\tau_i(p^{\mathrm{A}})$	$\tau_i(p^*)$
DIXON3DQ	2728.24	1286.14	1572.75
EIGENALS	1768.25	1119.76	1177.19
NCB20B	1444.47	964.25	1152.8
CHAINWOO	1224.25 F	1372.78 F	1224.1 F
GENHUMPS	615.29 F	4028.99 F	444.96 F

Table 3: Problems with relatively high and varying solution times. A 'F' indicates a failure. CPU times are in seconds.

7 Conclusion

We presented a general framework for the optimization of algorithmic parameters, which is general enough to be applied to many branches of engineering and computational science. Using the algorithm presented in [1], this framework may also be extended to the case where some parameters are categorical. The framework is illustrated on an example which at the same time addresses the long-standing question of determining locally optimal trust-region parameters in unconstrained minimization. The MADS algorithm for non-smooth optimization of expensive functions [3] is at the core of the framework.

The very notion of optimality for such problems is not well defined. Hence,

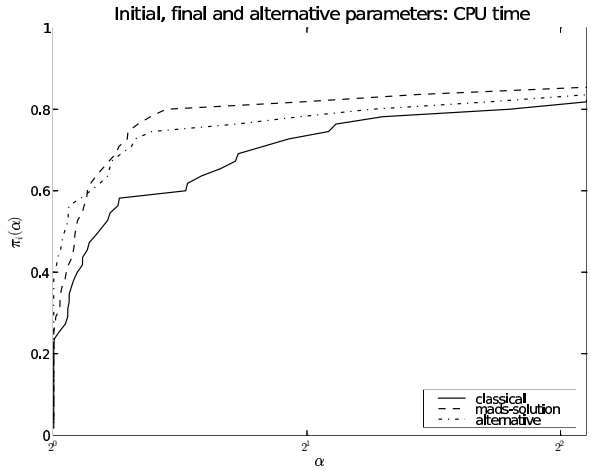


Figure 4: Profile comparing the CPU time required for one evaluation of the MADS objective for the initial, final and alternative parameter values.

our aim in designing this framework was to suggest values for the parameters which seem to perform *better*, in a sense specified by the user, on a set of problems which are context-dependent and can also be specified by the user. In real applications, we believe this black-box approach is beneficial since it allows users to take full advantage of their knowledge of the context to design appropriate test sets and performance measures. As our numerical experience indicates, the choice of objective to be optimized will likely influence the results.

We reserve the exploration of more elaborate objective functions, making provision for outliers, and the study of scaling strategies for MADS for future work. We also wish to explore modifications of the algorithm to accept integer and categorical parameters and more general constraints.

Appendix: Tables

Tables 4 and 5 report numerical results on the objective and surrogate lists respectively using traditional values of the parameters. The headers of the columns are as follows. The column titled n gives the number of variables, $\#f$ eval is the number of function evaluations, Time is the CPU time in seconds, $\|\nabla f(x^*)\|_2$ is the Euclidean norm of the gradient at the final point, and $f(x^*)$

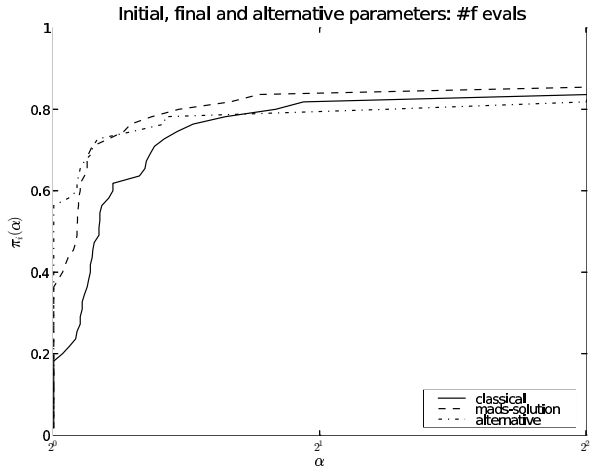


Figure 5: Profile comparing the number of function evaluations required for one evaluation of the MADS objective for the initial, final and alternative parameter values.

is the final objective value. The Exit column gives the exit code of the solver. An exit code of 0 indicates a success and an exit code of 2 indicates that the maximal number of evaluations, set to 1000, was reached.

Acknowledgments We wish to thank Gilles Couture for developping NOMAD, the C++ implementation of MADS.

References

[1] C. Audet and J. E. Dennis, Jr. Pattern search algorithms for mixed variable programming. *SIAM Journal on Optimization*, 11(3):573–594, 2000.

[2] C. Audet and J. E. Dennis, Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.

[3] C. Audet and J. E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. Technical Report G–2004–04, Les Cahiers du GERAD, Montréal, 2004.

[4] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. B. Serafini, and V. Torczon. Optimization using surrogate objectives on a helicopter test example.

Name	n	$\#f$ eval	Time (s)	$\|\nabla f(x^*)\|_2$	$f(x^*)$	Exit
ARWHEAD	5000	7	0.9500	0.1475E-13	0.0000E+00	0
BDQRTIC	5000	18	5.2400	0.1404E-06	0.2001E+05	0
BROYDN7D	5000	331	443.6999	0.7049E-06	0.1749E+04	0
BRYBND	5000	15	5.8300	0.4633E-07	0.6970E-16	0
CHAINWOO	4000	1001	1224.2506	0.1818E+03	0.1217E+05	2
COSINE	10000	14	1.8300	0.1258E-06	-0.9999E+04	0
CRAGGLVY	5000	1001	83.3004	0.3276E-05	0.1688E+04	2
DIXMAANA	3000	12	0.5300	0.9559E-06	0.1000E+01	0
DIXMAANB	3000	12	0.4600	0.4180E-08	0.1000E+01	0
DIXMAANC	3000	14	0.5500	0.3559E-06	0.1000E+01	0
DIXMAAND	3000	15	0.6400	0.3850E-06	0.1000E+01	0
DIXMAANE	3000	15	7.1900	0.4101E-06	0.1000E+01	0
DIXMAANF	3000	26	37.2900	0.4084E-06	0.1000E+01	0
DIXMAANG	3000	24	30.9100	0.1895E-06	0.1000E+01	0
DIXMAANH	3000	22	12.8400	0.4658E-07	0.1000E+01	0
DIXMAANI	3000	17	178.4300	0.2117E-06	0.1000E+01	0
DIXMAANJ	3000	35	455.6000	0.2268E-06	0.1000E+01	0
DIXMAANL	3000	40	461.4600	0.3689E-06	0.1000E+01	0
DIXON3DQ	10000	10	2728.2402	0.4521E-07	0.7349E-13	0
DQDRTIC	5000	13	1.2500	0.2305E-17	0.5191E-36	0
DQRTIC	5000	53	2.3300	0.4194E-06	0.7316E-08	0
EDENSCH	2000	21	0.9800	0.7168E-06	0.1200E+05	0
EG2	1000	4	0.0600	0.5958E-08	-0.9989E+03	0
EIGENALS	2550	98	1768.2498	0.5615E-06	0.1189E-10	0
ENGVAL1	5000	18	2.4700	0.4160E-07	0.5549E+04	0
EXTROSNB	1000	1001	78.2900	0.4567E-05	0.3825E-06	2
FLETCBV3	5000	1001	314.5186	0.3291E+02	-0.6676E+07	2
FLETCHBV	5000	1001	314.9600	0.3083E+10	-0.6562E+15	2
FLETCHCR	1000	1001	77.1401	0.4758E+01	0.2897E+03	2
FMINSRF2	5625	140	151.8100	0.9511E-06	0.1000E+01	0
FMINSURF	5625	122	134.0901	0.7822E-07	0.1000E+01	0
FREUROTH	5000	18	2.3700	0.6173E-06	0.6082E+06	0
GENHUMPS	5000	1001	615.2895	0.6048E+04	0.8634E+08	2
INDEF	5000	1001	260.5709	0.7193E+02	-0.3090E+14	2
LIARWHD	5000	20	1.7400	0.5457E-08	0.7421E-17	0
MODBEALE	20000	23	38.4600	0.4206E-07	0.6602E-15	0
NCB20	5010	86	639.8300	0.3638E-07	-0.1456E+04	0
NCB20B	5000	23	1444.4702	0.2534E-06	0.7351E+04	0
NONCVXU2	5000	1001	647.5209	0.2864E+02	0.7246E+05	2
NONCVXUN	5000	1001	708.7709	0.3753E+02	0.6178E+05	2
NONDIA	5000	8	0.6100	0.1947E-08	0.1173E-16	0
NONDQUAR	5000	164	415.1200	0.8056E-06	0.2349E-06	0
PENALTY1	1000	62	0.4300	0.9703E-07	0.9686E-02	0
POWELLSG	5000	25	1.8500	0.4753E-06	0.2520E-08	0
POWER	10000	44	54.7400	0.8258E-06	0.7139E-10	0
QUARTC	5000	53	2.3600	0.4194E-06	0.7316E-08	0
SCHMVETT	5000	11	3.8500	0.3894E-07	-0.1499E+05	0
SINQUAD	5000	16	2.7200	0.5097E-06	-0.6757E+07	0
SPARSQUR	10000	28	17.1000	0.3063E-06	0.2756E-09	0
SROSENBR	5000	11	0.5600	0.1871E-06	0.1774E-16	0
TESTQUAD	5000	18	37.9700	0.4908E-07	0.3760E-16	0
TOINTGSS	5000	23	3.6000	0.2688E-06	0.1000E+02	0
TQUARTIC	5000	15	1.2000	0.3883E-09	0.1884E-15	0
TRIDIA	5000	17	28.4000	0.4698E-07	0.8546E-17	0
WOODS	4000	66	6.0900	0.7967E-06	0.4410E-12	0

Table 4: Results on the objective list

Name	n	$\#f$ eval	Time (s)	$\|\nabla f(x^*)\|_2$	$f(x^*)$	Exit
3PK	30	74	0.1400	0.3417E-06	0.1720E+01	0
ARGLINA	200	6	0.4500	0.1373E-12	0.2000E+03	0
BIGGS6	6	27	0.0200	0.8514E-06	0.5656E-02	0
BOX3	3	9	0.0300	0.2102E-08	0.2384E-14	0
BROWNAL	200	7	0.3000	0.1280E-08	0.3614E-19	0
BROWNBS	2	26	0.0200	0.0000E+00	0.0000E+00	0
BROWNDEN	4	13	0.0200	0.1079E-09	0.8582E+05	0
CHNROSNB	50	69	0.3200	0.2874E-07	0.6269E-16	0
CLIFF	2	30	0.0200	0.1494E-07	0.1998E+00	0
CUBE	2	48	0.0200	0.1135E-08	0.3223E-17	0
DECONVU	61	93	2.9100	0.2916E-06	0.4045E-09	0
DENSCHND	3	34	0.0300	0.4029E-07	0.3540E-09	0
DIXMAANK	15	12	0.0200	0.2982E-10	0.1000E+01	0
DJTL	2	171	0.0800	0.7967E-07	-0.8952E+04	0
ERRINROS	50	74	0.2500	0.2803E-07	0.3990E+02	0
GENROSE	500	465	28.5399	0.1275E-06	0.1000E+01	0
GROWTHLS	3	186	0.1200	0.2505E-07	0.1004E+01	0
GULF	3	58	0.1800	0.3461E-06	0.4802E-12	0
HAIRY	2	82	0.0400	0.1126E-10	0.2000E+02	0
HEART6LS	6	1001	0.5900	0.5826E+03	0.6271E-01	2
HEART8LS	8	263	0.2500	0.2651E-07	0.2172E-18	0
HIELOW	3	15	2.7300	0.2429E-07	0.8742E+03	0
HIMMELBF	4	205	0.1100	0.2911E-07	0.3186E+03	0
HUMPS	2	1001	0.4400	0.2628E+02	0.9715E+04	2
LOGHAIRY	2	1001	0.4400	0.1496E-02	0.6201E+01	2
MANCINO	100	24	11.6100	0.1149E-06	0.1679E-20	0
MARATOSB	2	1001	0.3400	0.6104E+02	-0.8601E+00	2
MEYER3	3	1001	0.5400	0.8330E+00	0.8820E+02	2
OSBORNEA	5	74	0.0900	0.3196E-07	0.5465E-04	0
OSBORNEB	11	25	0.1900	0.4666E-06	0.4014E-01	0
PALMER1C	8	1001	0.6800	0.1781E-02	0.9761E-01	2
PALMER1D	7	54	0.0600	0.9458E-06	0.6527E+00	0
PALMER2C	8	1001	0.5900	0.8995E-04	0.1449E-01	2
PALMER3C	8	651	0.3100	0.3740E-07	0.1954E-01	0
PALMER4C	8	85	0.0800	0.2544E-06	0.5031E-01	0
PALMER6C	8	234	0.1300	0.8616E-06	0.1639E-01	0
PALMER7C	8	1001	0.4100	0.6120E-03	0.6020E+00	2
PALMER8C	8	273	0.1400	0.9022E-06	0.1598E+00	0
PENALTY2	200	18	0.3900	0.4437E-07	0.4712E+14	0
PFIT1LS	3	630	0.2800	0.8950E-06	0.4115E-08	0
PFIT2LS	3	247	0.1000	0.3223E-06	0.1512E-13	0
PFIT3LS	3	280	0.1100	0.2982E-08	0.4987E-18	0
PFIT4LS	3	515	0.2100	0.4567E-06	0.5412E-14	0
SENSORS	100	22	11.1300	0.3931E-12	-0.1967E+04	0
SISSER	2	15	0.0200	0.5015E-06	0.4738E-09	0
SNAIL	2	80	0.0400	0.2978E-09	0.2217E-19	0
TOINTGOR	50	12	0.0400	0.5451E-07	0.1374E+04	0
TOINTPSP	50	21	0.0200	0.3395E-07	0.2256E+03	0
TOINTQOR	50	11	0.0200	0.4696E-07	0.1175E+04	0
VARDIM	200	31	0.0200	0.1863E-09	0.3238E-26	0
VAREIGVL	50	15	0.0900	0.4279E-06	0.2535E-10	0
VIBRBEAM	8	1001	2.3900	0.1001E-03	0.1564E+00	2
WATSON	12	11	0.0400	0.2012E-07	0.6225E-09	0
YFITU	3	77	0.0600	0.5824E-08	0.8460E-12	0

Table 5: Results on the surrogate list

In J. Borggaard, J. Burns, E. Cliff, and S. Schreck, editors, *Optimal Design and Control*, Progress in Systems and Control Theory, pages 49–58, Cambridge, Massachusetts, 1998. Birkhäuser.

[5] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13, February 1999.

[6] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Wiley, New York, 1983. Reissued in 1990 by SIAM Publications, Philadelphia, as Vol. 5 in the series Classics in Applied Mathematics.

[7] T. F. Coleman and Y. Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on Optimization*, 6(2):418–445, 1996.

[8] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50:399–430, 1988.

[9] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. SIAM, Philadelphia, USA, 2000.

[10] J. E. Dennis, M. Heinkenschloss, and L. N. Vicente. Trust-region interior-point SQP algorithms for a class of nonlinear programming problems. *SIAM Journal on Control and Optimization*, 36(5):1750–1794, 1998.

[11] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Classics in Applied Mathematics. SIAM, Philadelphia PA, USA, 1996.

[12] E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming, Series B*, 91:201–213, 2002.

[13] S. M. Goldfeldt, R. E. Quandt, and H. F. Trotter. Maximization by qua. r.tic hill-climbing. *Econometrica*, 34:541–551, 1966.

[14] N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM Journal on Optimization*, 9(2):504–525, 1999.

[15] N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint. Sensitivity of trust-region algorithms. Technical Report 04/07, Mathematics Department, University of Namur, Belgium, 2004.

[16] N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. *Transactions of the ACM on Mathematical Software*, 29(4):353–372, December 2003.

[17] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr and SifDec, a Constrained and Unconstrained Testing Environment, revisited. *Transactions of the ACM on Mathematical Software*, 29(4):373–394, December 2003.

[18] L. Hei. A self-adaptive trust region algorithm. Technical Report, ECE Department, Northwestern University, Evaston IL, USA, 2000.

[19] J. Jahn. *Introduction to the Theory of Nonlinear Optimization*. Springer, Berlin, 1994.

[20] E. B. Leach. A note on inverse function theorem. In *Proceedings of the AMS*, volume 12, pages 694–697, 1961.

[21] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly Journal on Applied Mathematics*, 2:164–168, 1944.

[22] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11:431–441, 1963.

[23] M. D. McKay, W. J. Conover, and R. J. Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[24] J. J. Moré. Recent developments in algorithms and software for trust region methods. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 258–287, Heidelberg, Berlin, New York, 1983. Springer Verlag.

[25] D. D. Morrison. Methods for nonlinear least squares problems and convergence proofs. In J. Lorell and F. Yagi, editors, *Proceedings of the Seminar on Tracking Programs and Orbit Determination*, pages 1–9, Pasadena, USA, 1960. Jet Propulsion Laboratory.

[26] M. S. Ouali, H. Aoudjit, and C. Audet. Optimisation des stratégies de maintenance. *Journal Européen des Systèmes Automatisés*, 37(5):587–605, 2003.

[27] M. J. D. Powell. A new algorithm for unconstrained optimization. In J. B. Rosen, O. L. Mangasarian, and K. Ritter, editors, *Nonlinear Programming*, pages 31–65, London, 1970. Academic Press.

[28] A. Sartenaer. Armijo-type condition for the determination of a generalized Cauchy point in trust region algorithms using exact or inexact projections on convex constraints. *Belgian Journal of Operations Research, Statistics and Computer Science*, 33(4):61–75, 1993.

[29] Ch. Sebudandi and Ph. L. Toint. Nonlinear optimization for seismic travel time tomography. *Geophysical Journal International*, 115:929–940, 1993.

[30] J. S. Shahabuddin. *Structured trust-region algorithms for the minimization of nonlinear functions*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, USA, 1996.

[31] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[32] M. Stein. Large sample properties of simulations using latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987.

[33] Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. In I. S. Duff, editor, *Sparse MA. R.ces and Their Uses*, pages 57–88, London, 1981. Academic Press.

[34] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, February 1997.

[35] J. M. B. Walmag and E. J. M. Delhez. A note on trust-region radius update. Technical Report, Modélisation et Méthodes Mathématiques, Université de Liège, Belgium, 2004.

[36] Y. Yuan. An example of non-convergence of trust region algorithms. In Y. Yuan, editor, *Advances in Nonlinear Programming*, pages 205–218, Dordrecht, The Netherlands, 1998. Kluwer Academic Publishers.