An active-set trust-region method for derivative-free nonlinear bound-constrained optimization

by S. Gratton, Ph. L. Toint and A. Tröltzsch

Report NAXYS-01-2010 9 July 2010

Namur Research Center On Complex Systems

University of Namur
61, rue de Bruxelles, B5000 Namur (Belgium)

http://www.fundp.ac.be/sciences/naxys

An active-set trust-region method for derivative-free nonlinear bound-constrained optimization

Serge Gratton[*], Philippe L. Toint[†], Anke Tröltzsch[‡]

9 July 2010

Abstract

We consider an implementation of a recursive model-based active-set trust-region method for solving bound-constrained nonlinear non-convex optimization problems without derivatives using the technique of self-correcting geometry proposed in [24]. Considering an active-set method in model-based optimization creates the opportunity of saving a substantial amount of function evaluations when maintaining smaller interpolation sets while proceeding optimization in lower dimensional subspaces. The resulting algorithm is shown to be numerically competitive.

Keywords: derivative-free optimization, bound constraints, nonlinear optimization, active-set methods, trust region, numerical experiments.

1 Introduction

Derivative-free optimization has enjoyed renewed interest over the past years, mostly motivated by the ever growing need to solve optimization problems defined by functions whose values are computed by simulation. Model-based methods have been pioneered by Powell [19] and several such methods for solving unconstrained optimization problems without derivatives (and associated software implementations) are available today [19, 20, 7, 8, 22, 17] and have been shown to be numerically efficient [18]. Algorithms of this type are discussed extensively in the recent book [10] by Conn, Scheinberg and Vicente.

Many of these methods construct local polynomial interpolation-based models of the objective function and compute steps by minimizing these models inside a region using the standard trust-region methodology (see [6] for detailed information). The models are built to interpolate previously computed function values at past iterates or at specially constructed points. For the model to be well-defined, the interpolation points must be poised [9, 21], meaning that the geometry of this set of points has to "cover the space" sufficiently well to stay safely away from degeneracy of the interpolation conditions. To maintain a good poisedness of the set, geometry improving steps are included in many model-based DFO algorithms, but their necessity has recently been questioned (see [13]) in that a simple method not using them at all has shown surprisingly good performance. However, it was also shown in [24] that convergence from arbitrary starting points may then be lost, but that a new algorithm can be designed to substantially reduce the need of such geometry improving steps by exploiting a self-correcting property of the interpolation set geometry.

The purpose of this paper is to describe a particular implementation of this algorithm (see Algorithm 1 on page 3 of this paper) in a bound-constrained setting, where bounds on the variables are handled by an active set strategy.

The paper is organized as follows. We present the basic framework of our algorithm in Section 2. After recalling elements of polynomial interpolation theory, we discuss algorithmic concepts in Section 3, while Section 4 is concerned with practical implementation issues. Section 5 reports numerical experiments

[*]ENSEEIHT, 2, rue Charles Camichel, 31000 Toulouse, France. Email: serge.gratton@enseeiht.fr

[†]Namur Research Center for Complex Systems (NAXYS), FUNDP-University of Namur, 61, rue de Bruxelles, B-5000 Namur, Belgium. Email: philippe.toint@fundp.ac.be (corresponding author)

[‡]CERFACS, 42 avenue G. Coriolis, 31057 Toulouse, France. Email: anke@cerfacs.fr

with the new algorithm and compares it to NEWUOA [22] and BOBYQA [23], two state-of-the-art packages. We finally present some conclusions and perspectives in Section 6.

2 A recursive active-set trust-region DFO algorithm

We consider the bound-constrained optimization problem

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} f(x), \\ &\text{subject to} \quad l \le x \le u, \end{aligned} \tag{1}$$

where f is a nonlinear function from $\mathbb{R}^n$ into $\mathbb{R}$, which is bounded below, and where l and u are vectors of (possibly infinite) lower and upper bounds on x. We denote the feasible domain of this problem by $\mathcal{F}$.

Our approach uses an iterative trust-region method. At each iteration of such a method, a model of the form

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

(where g_k and H_k are the model's gradient and Hessian, respectively) is minimized inside a trust region

$$\mathcal{B}_\infty(x_k, \Delta_k) = \{x \in \mathbb{R}^n \mid \|x - x_k\|_\infty \le \Delta_k\}, \tag{3}$$

where $\|\cdot\|_\infty$ denotes the infinity norm.

This (possibly approximate) minimization yields a trial point $x_k + s_k$, which is accepted as the new iterate if the ratio

$$\rho_k \stackrel{\text{def}}{=} \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \tag{4}$$

is larger than a constant $\eta_1 > 0$. In this case, the model is updated and the trust-region radius is possibly increased. If $\rho \le \eta_1$, the trial point is rejected and the trust-region radius is decreased. Methods of this type have long been considered for the solution of numerical optimization problems, and we refer the reader to [6] for an extensive coverage of this topic.

In our context, the model (2) will be[1] determined by interpolating known objective function's values at a given set $\mathcal{Y}_k$ of *interpolation points*, meaning that

$$m_k(y) = f(y) \;\text{ for all }\; y \in \mathcal{Y}_k. \tag{5}$$

The set $\mathcal{Y}_k$, known as the *interpolation set*, contains (in our case) at least $n + 1$ points and is chosen as a subset of $\mathcal{X}_k$, the set of all points where the value of the objective function f is known. How to choose this interpolation set is of course one of the main issues we have to address below, as not every set $\mathcal{Y}_k$ is suitable. We also propose to handle the bound constraints by an "active set" approach in the sense that our method keeps track of all such constraints which are (nearly) active and then performs minimization in the subspace of the remaining free variables.

An outline of the algorithm is given in Algorithm 1 on the following page. This outline is purposedly schematic and many more of its details needs to be discussed. This discussion constitutes the body of Section 3. At this stage, we only need to mention that the initial call is performed with $\mathcal{S}_0 = \mathbb{R}^n$ and $\mathcal{X}_0 = \mathcal{Z}_0 = \{x_0\}$ (typically). It is also assumed that

$$\Delta \le \frac{1}{2} \min_{i=1,\ldots,n} (u(i) - l(i)) \;\text{ and }\; l(i) + \Delta \le x_0(i) \le u(i) - \Delta, \tag{6}$$

where $x(i)$ denotes the i-th component of the vector x. We finally note that the functions values associated with $\mathcal{X}_k$ and $\mathcal{Z}_k$ are considered implicitly.

3 Algorithmic details

We now need to make the steps of our algorithmic outline more precise, and discuss all the relevant ingredients.

[1] Except for the case of "dummy points", see Section 3.3 below.

Algorithm 1 ***BC-DFO*** $(\mathcal{S}_0, \mathcal{X}_0, x_0, \mathcal{Z}_0, \Delta_0, \epsilon)$

Step 0: Initialization. A trust-region radius Δ_0 and an accuracy threshold ϵ are given. $\mathcal{X}_0$, the set of all points, and a tentative interpolation set $\mathcal{Z}_0$, which contains the initial point x_0, are also given. Set $k = 0$.

Step 1: Ensure the suitability of $\mathcal{Z}_0$ and build the initial model.
Update $\mathcal{Z}_0$ to an interpolation set $\mathcal{Y}_0$ suitable for building an interpolation model with $|\mathcal{Y}_0| \geq \dim(\mathcal{S}_0)+1$. Then build the corresponding interpolation model m_0.

Step 2: Possibly restrict minimization to a subspace $\mathcal{S}_\mathbf{k}$.
Step 2.1: Check for (nearly) active bounds.
Determine active and nearly active bounds, as well as the corresponding subspace $\mathcal{S}_k$ spanned by the remaining free variables. If there is no active or nearly active bound or if $\mathcal{S}_k$ has already been explored, go to Step 3.
Step 2.2: Project information on the subspace of free variables.
Project points in $\mathcal{X}_k$ which lie close to the (nearly) active bounds on $\mathcal{S}_k$ and associate with them suitable function values estimates.
Step 2.3: Build a tentative interpolation set in the subspace.
Build a new tentative interpolation set $\mathcal{Z}_k$ in $\mathcal{S}_k$ including the projected points, if any.
Step 2.4: Solve in $\mathcal{S}_\mathbf{k}$ by a recursive call.
Call algorithm

$$\textbf{\textit{BC-DFO}}(\mathcal{S}_k, \mathcal{X}_k, x_k, \mathcal{Z}_k, \Delta_k, \epsilon),$$

yielding a solution $x^*_{\mathcal{S}}$ of the subspace problem.
Step 2.5: Return to the full space.
If $\dim(\mathcal{S}_k) < n$, return $x^*_{\mathcal{S}}$. Otherwise, redefine $x_k = x^*_{\mathcal{S}}$, construct a new interpolation set $\mathcal{Y}_k$ around x_k and build the corresponding model m_k.

Step 3: Criticality test.
If $\|P_{\mathcal{F}}(x_k - \nabla m_k(x)) - x_k\|_\infty \leq \epsilon$ (where $P_{\mathcal{F}}$ is the projection onto $\mathcal{F}$) and the model m_k is sufficiently accurate, return x_k.

Step 4: Compute a trial point and evaluate the objective function.
Compute $x^+_k = x_k + s_k$ by applying a projected truncated conjugate gradient algorithm. Evaluate f at x^+_k and compute the ratio ρ_k from (4).

Step 5: Define the next iterate and update the trust-region radius.
Take a decision how to possibly incorporate the current trial point x^+_k into the set $\mathcal{Y}_{k+1}$, define x_{k+1} and determine Δ_{k+1}.

Step 6: Update the model.
If $\mathcal{Y}_{k+1} \neq \mathcal{Y}_k$, compute the interpolation model m_{k+1} around x_{k+1} using $\mathcal{Y}_{k+1}$. Update $\mathcal{X}_{k+1} = \mathcal{X}_k \cup \{x_{k+1}\}$. Increment k by one and go to Step 2.

3.1 Polynomial interpolation and poisedness

The first question is to discuss when a model can be associated (numerically safely) to a given interpolation set $\mathcal{Y}$ (we drop the subscript k in this description for clarity). In order to provide a formal answer, we have to briefly recall some basic material about multivariate interpolation and Lagrange polynomials (the reader is referred to [10] for further details and definitions). Consider $\mathcal{P}_n^d$, the space of polynomials of degree $\leq d$ in $\mathbb{R}^n$. A polynomial basis $\phi = \{\phi_1(x), \phi_2(x), ..., \phi_p(x)\}$ of $\mathcal{P}_n^d$ is a set of p polynomials of degree $\leq d$ that span $\mathcal{P}_n^d$. Well-known examples of such bases are the basis of monomials and bases of Lagrange or Newton fundamental polynomials. For any basis ϕ, any polynomial $m(x) \in \mathcal{P}_n^d$ can be written as

$$m(x) = \sum_{j=1}^{p} \alpha_j \phi_j(x),$$

where α_j are real coefficients.

Given an interpolation set $\mathcal{Y} = \{y^1, y^2, ...y^p\} \subset \mathbb{R}^n$ and a polynomial $m(x)$ of degree d in $\mathbb{R}^n$ that interpolates $f(x)$ at the points of $\mathcal{Y}$, the coefficients $\alpha_1, ..., \alpha_p$ can be determined by solving the linear system

$$M(\phi, \mathcal{Y})\alpha_\phi = f(\mathcal{Y}),$$

where

$$M(\phi, \mathcal{Y}) = \begin{pmatrix} \phi_1(y^1) & \phi_2(y^1) & \cdots & \phi_p(y^1) \\ \phi_1(y^2) & \phi_2(y^2) & \cdots & \phi_p(y^2) \\ \vdots & \vdots & & \vdots \\ \phi_1(y^p) & \phi_2(y^p) & \cdots & \phi_p(y^p) \end{pmatrix}, \quad f(\mathcal{Y}) = \begin{pmatrix} f(y^1) \\ f(y^2) \\ \vdots \\ f(y^p) \end{pmatrix}. \tag{7}$$

If the coefficient matrix $M(\phi, \mathcal{Y})$ of the system is nonsingular for some basis ϕ in $\mathcal{P}_n^d$, the set of points $\mathcal{Y} = \{y^1, y^2, ...y^p\}$ is called *poised* for polynomial interpolation in $\mathbb{R}^n$. If the interpolation set $\mathcal{Y}$ is poised, the basis of Lagrange polynomials (see [10, page 39]) $\{\ell_i(x)\}_{i=1}^p$ exists and is uniquely defined. The unique polynomial $m(x)$ which interpolates $f(x)$ on $\mathcal{Y}$ using this basis of Lagrange polynomials can be expressed as

$$m(x) = \sum_{i=1}^{p} f(y^i)\, \ell_i(x). \tag{8}$$

The quality of poisedness of $\mathcal{Y}$ can be measured using the following notion [10]. Let a set $\mathcal{B} \in \mathbb{R}^n$ be given and let ϕ be a basis in $\mathcal{P}_n^d$. A poised set $\mathcal{Y} = \{y^1, y^2, ..., y^p\}$ is said to be Λ-poised in $\mathcal{B}$ for some $\Lambda > 0$ if and only if for the basis of Lagrange polynomials associated with $\mathcal{Y}$

$$\Lambda \geq \max_{1 \leq i \leq p} \max_{x \in \mathcal{B}} |\ell_i(x)|.$$

This upper bound on the absolute value of the Lagrange polynomials in a region $\mathcal{B}$ can be interpreted as a measure of the distance to a nonpoised set or equivalently to a singular system matrix [10, page 49]. Importantly for our purposes, Lagrange polynomial values and Λ-poisedness are also used to bound the model function and model gradient accuracy: given the sphere

$$\mathcal{B}_2(x, \Delta) \stackrel{\text{def}}{=} \{v \in \mathbb{R}^n \mid \|v - x\|_2 \leq \sqrt{n}\Delta\},$$

a poised interpolation set $\mathcal{Y} \in \mathcal{B}_2(x, \Delta)$ and its associated basis of Lagrange polynomials $\{\ell_i(x)\}_{i=1}^p$, there exists constants $\kappa_{ef} > 0$ and $\kappa_{eg} > 0$ such that, for any interpolation polynomial $m(x)$ of degree one or higher of the form (8) and any point $y \in \mathcal{B}_2(x, \Delta)$,

$$|f(y) - m(y)| \leq \kappa_{ef} \sum_{i=1}^{p} \|y_i - y\|_2^2 |\ell_i(y)| \tag{9}$$

and

$$\|\nabla_x f(y) - \nabla_x m(y)\|_2 \leq \kappa_{eg} \Lambda \Delta, \tag{10}$$

where

$$\Lambda = \max_{i=1,...,p} \max_{x \in \mathcal{B}_2(x,\Delta)} |\ell_i(x)| \tag{11}$$

(see [4]). From a practical point of view, it is often important to compute the global maximum in (11) relatively accurately, which can be done using the Hebden-Moré-Sorensen algorithm (see [6], Section 7.3) in the Euclidean norm and motivates our choice of $\mathcal{B}_2(x,\Delta)$. Note that our definition of this last neighbourhood guarantees that $\mathcal{B}_\infty(x,\Delta) \subset \mathcal{B}_2(x,\Delta)$, and the error bounds (9)-(10) therefore hold in $\mathcal{B}_\infty(x,\Delta)$.

An alternative measure of poisedness may be derived, albeit indirectly, from the matrix $M(\phi,\mathcal{Y})$. First note that the condition number of this matrix is in general not a satisfactory measure of poisedness of $\mathcal{Y}$ since it can be made arbitrarily large by changing the basis ϕ and scaling $\mathcal{Y}$. However, [10] have shown that a relation between the condition number of $\hat{M} = M(\bar{\phi},\hat{\mathcal{Y}})$ and the measure of Λ-poisedness can be established when considering the basis of monomials $\bar{\phi}$ and $\hat{\mathcal{Y}}$, a shifted and scaled version of $\mathcal{Y}$. This new matrix is computed as follows. Given a sample set $\mathcal{Y} = \{y^1, y^2, ..., y^p\}$, a shift of coordinates is first performed to center the interpolation set $\mathcal{Y}$ at the origin, giving $\{0, y^2 - y^1, ..., y^p - y^1\}$, where y^1 denotes the current best iterate which is usually the center of the interpolation. The region $\mathcal{B}$ is then fixed to be $\mathcal{B}_2(0,\Delta(\mathcal{Y}))$ and the radius

$$\Delta = \Delta(\mathcal{Y}) = \max_{2\le i\le p} \|y^i - y^1\|_2$$

is used to scale the set, yielding

$$\hat{\mathcal{Y}} = \{0, \hat{y}^2, ..., \hat{y}^p\} = \{0, (y^2 - y^1)/\Delta, ..., (y^p - y^1)/\Delta\}.$$

The resulting scaled interpolation set $\hat{\mathcal{Y}}$ is then contained in a ball of radius one centered at the origin. The following result is then derived in [10, page 51].

Theorem 3.1 *If $\hat{M}$ is nonsingular and $\|\hat{M}^{-1}\|_2 \le \Lambda$, then the set $\hat{\mathcal{Y}}$ is $\sqrt{p}\Lambda$-poised in the unit ball $\mathcal{B}(0,1)$ centered at 0. Conversely, if the set $\hat{\mathcal{Y}}$ is Λ-poised in the unit ball $\mathcal{B}(0,1)$ centered at 0, then*

$$\kappa(\hat{M}) = \|\hat{M}\|_2\|\hat{M}^{-1}\|_2 \le \theta p^2 \Lambda, \tag{12}$$

where $\theta > 0$ is dependent on n and d, but independent of $\hat{\mathcal{Y}}$ and Λ.

This means that this condition number of $M(\bar{\phi},\hat{\mathcal{Y}})$ can also be used to monitor poisedness of the interpolation set without computing Lagrange polynomials and Λ. Conversely, we can conclude that if the set $\hat{\mathcal{Y}}$ is reasonably well-poised, then there is virtually no risk of numerical difficulties when using $M(\bar{\phi},\hat{\mathcal{Y}})$.

One may then wonder which measure of poisedness is more appropriate. In our experience, both have their advantages. The measure in terms of the Lagrange polynomials is more convenient for estimating the (crucial) accuracy of the model's gradient near convergence (see Section 4.3), while the condition number of $\hat{M}$ is cheaper to compute and suitable for the update of interpolation sets (see Section 3.2).

If we now consider using interpolation models in the framework of a trust-region method for optimization, we observe that interpolation models of varying degree are possible and indeed desirable in the course of the complete minimization. In early stages, a more economical linear model (using $p = n + 1$ function values) is often sufficient while faster progress to a close solution may be achieved with quadratic ones (which uses $p = \frac{1}{2}(n+1)(n+2)$ values). It is then natural to consider models evolving from linear to fully quadratic as minimization progresses. In our algorithm, models become progressively "more quadratic" by considering banded matrices H_k with increasing semi-bandwidth. The number of interpolation conditions p that are imposed on a model $m(x)$ therefore varies in the interval $[n+1, \frac{1}{2}(n+1)(n+2)]$. Note that this "expanding band" strategy is particularly efficient if the true Hessian of the objective function is itself banded.

3.2 Ensuring suitability of a tentative interpolation set

At the start (in Step 1) of our algorithm, we are given a tentative interpolation set $\mathcal{Z}_0$ and have to build an interpolation model using this set as much as possible. As is clear from the previous subsection, using the whole of $\mathcal{Z}_0$ is only possible if this set is suitable in the sense that it is (sufficiently well) poised. We now describe the procedure used in our algorithm to verify this condition and/or to modify $\mathcal{Z}_0$ to form

$\mathcal{Y}_0$ if necessary. This procedure distinguishes two cases, depending whether or not $\mathcal{Z}_0$ contains more than a single point.

If $|\mathcal{Z}_0| = 1$, our objective is then to build a poised interpolation set $\mathcal{Y}_0$ containing $\{x_0\} = \mathcal{Z}_0$ and contained in the initial trust region $\mathcal{B}(x_0, \Delta_0)$. This is achieved by choosing the interpolation points at the vertices of an n-dimensional simplex, as given by the formula

$$y_{i+1} = x_0 \pm \Delta_0 e_i, i = 1, 2, ..., n$$

where e_i is the i-th coordinate vector in $\mathbb{R}^n$ and the sign is negative for the initial computation of an interpolation set but, in an attempt to diversify the set of interpolation points, alternates whenever applied again during the calculation.

If $|\mathcal{Z}_0| > 1$, an obvious choice would be to search first for the set $\mathcal{Y}_0 \subseteq \mathcal{Z}_0$ of size $p \leq n+1$ for which the condition number of the shifted and scaled system matrix $M(\bar{\phi}, \hat{\mathcal{Y}}_0)$ is the smallest out of all matrices associated with subsets of $\mathcal{Z}_0$ consisting of at most $\min(n+1, |\mathcal{Z}_0|)$ points. However, this procedure is quite costly due to its combinatorial nature, and we have decided to use a cheaper technique adapted from [1].

Suppose that there exists a subset of points $\mathcal{W}_p = \{x^1, x^2, ..., x^p\}$ in $\mathcal{Z}_0$ spanning a p-dimensional linear manifold L. Our selection problem in $\mathcal{Z}_0$ can be seen as an optimal basis problem in $\mathcal{W}_p$ where we have to find p vectors out of $\mathcal{Z}_0$ (of the form $x^i - x^j$) which are as linearly independent as possible. This problem can be formalized by regarding the points in $\mathcal{W}_p$ as nodes of a graph and the vectors $x^i - x^j$ as edges e_{ij} in this graph. It can then be shown that any set of p linearly independent vectors of the form $x^i - x^j$ that generate L corresponds to a tree spanning all nodes of $\mathcal{W}_p$, and conversely. In addition, Burdakov shows that the optimal basis problem can be reduced to finding the spanning tree t which minimizes the functional

$$\phi(t) = \sum_{e_{ij} \in t} \|x^i - x^j\|_2. \tag{13}$$

This author proposes in [2] a greedy algorithm for the solution of this minimization problem, in which the measure of linear independence given by $\Gamma(\{x^1, ..., x^p\}) = det(A^T A)$ is exploited, where

$$A = \left[\frac{x^1 - x^2}{\|x^1 - x^2\|_2}, \cdots, \frac{x^{p-1} - x^p}{\|x^{p-1} - x^p\|_2} \right] \in R^{n \times p-1}.$$

It can be shown that Γ is a scaling invariant measure of linear independence of the columns of A and thus also measures the general position of $\{x^1, ..., x^p\}$. It is always included in the interval $[0, 1]$ and takes the value 0 and 1 for linearly dependent or orthogonal columns, respectively. For a given threshold $\kappa_{th} \in (0, 1)$, we thus consider as sufficiently well-poised those sets of points, for which $\Gamma(\{x^1, ..., x^p\}) \geq \kappa_{th}$. It also turns out that $\Gamma(\{x^1, ..., x^p\})$ can be updated to $\Gamma(\{x^1, ..., x^p, x^{p+1}\})$ by a simple algebraic formula, thereby avoiding the repetitive computation of determinants.

As we do not know a subset of $\mathcal{Z}_0$ containing points in general position and not even the final number p of linearly independent points in $\mathcal{W}_p$, a modified version of Burdakov's greedy algorithm is proposed. In our version, the desired set is built incrementally in a sequence $\mathcal{W}_1$, ..., $\mathcal{W}_p$, where $\mathcal{W}_p$ is chosen over all sets of the form $\mathcal{W}_{p-1} \cup \{y\}$ for $y \in \mathcal{Z}_0 \setminus \mathcal{W}_{p-1} \setminus \mathcal{T}_k$, where $\mathcal{T}_k$ contains the points which were tried but couldn't be included in $\mathcal{W}_p$ while keeping Γ sufficiently large. The algorithm is formalized as Algorithm 2 on the next page.

Note that p, the number of points selected from $\mathcal{Z}_0$ may be smaller than $n+1$. In this case, we propose to complete $\mathcal{W}_p$ by $n+1-|\mathcal{W}_p|$ points selected randomly in the trust region to form the final interpolation set $\mathcal{Y}_0$. To ensure a good geometry of $\mathcal{Y}_0$, these random points $\{y_{p+1}, ..., y_{n+1}\}$ are then successively improved using the observation that replacing an interpolation point by the maximum of its associated Lagrange polynomial in the trust region ameliorates poisedness of the interpolation set (see [24], for instance). More precisely, for each $j = p+1, \ldots, n+1$, the absolute value of the Lagrange polynomial $\ell_j(x)$ associated with y_j is maximized inside $\mathcal{B}(x_k, \Delta_k)$ and y_j is then replaced by the computed maximizer $\tilde{y}_j$. This finally yields the "optimized" interpolation set $\mathcal{Y}_0 = \mathcal{W}_p \cup \{\tilde{y}_{p+1}\} \cup \ldots \cup \{\tilde{y}_{n+1}\}$.

3.3 Recursive call in the subspace $\mathcal{S}_k$

As we have mentioned above, our algorithm is of the active-set type and proceeds by exploring the subspace $\mathcal{S}_k$ defined by fixing active or nearly active variables at their bound. This choice is intended

Algorithm 2 Modified greedy algorithm for selecting a well-poised interpolation set (Inputs: $x_0, \mathcal{Z}_0$, Outputs: $\mathcal{W}_p, p$)

Step 1: Compute distances $\|x^i - x^j\|_2$ for $i, j = 1, ..., |\mathcal{Z}_0|$.
Step 2: Define $p = 1$, $\mathcal{W}_1 = \{x_0\}$ and $\mathcal{T}_0 = \emptyset$. Set $\Gamma(\mathcal{W}_1) = 1$ and $k = 0$.
while $(p < n + 1)$ and $(k < |\mathcal{Z}_0|)$ **do**
Step 3: Find $x^i \in \mathcal{W}_p$ and $x^j \in \mathcal{Z}_0 \setminus (\mathcal{W}_p \cup \mathcal{T}_k)$, such that $\|x^i - x^j\|_2$ is minimal.
Step 4: Compute the measure of degeneracy

$$\Gamma(\mathcal{W}_p \cup \{x^j\}) = \Gamma(\mathcal{W}_p) \frac{\|x^j_\perp\|_2^2}{\|x^j - x^i\|_2^2}$$

where $x^j_\perp = x^j - P_p x^j$, and P_p is the orthogonal projector on the linear manifold spanned by $\{x^i\}_1^p$.
Step 5: If $\Gamma(\mathcal{W}_p \cup \{x^j\}) \geq \kappa_{th}$, then set $\mathcal{W}_{p+1} = \mathcal{W}_p \cup \{x^j\}$, $\mathcal{T}_{k+1} = \mathcal{T}_k$, and increment p by one, else $\mathcal{T}_{k+1} = \mathcal{T}_k \cup \{x^j\}$.
Step 6: Increment k by one.
end while

to prevent the interpolation set from degenerating as would happen when points belonging to such a subspace are included in $\mathcal{Y}$. This section is devoted to the description of the mechanism for selecting (nearly) active bounds and then restarting the minimization in the associated subspace.

The lower und upper (nearly) active bounds at the current iterate x_k are defined, at iteration k, by those whose index is in one of the sets

$$\mathcal{L}_k = \{\, i \;\mid\; x_k(i) - \nabla m_k(i) < l(i) \text{ and } x_k(i) - l(i) \leq \epsilon_b\},$$
$$\mathcal{U}_k = \{\, i \;\mid\; x_k(i) - \nabla m_k(i) > u(i) \text{ and } u(i) - x_k(i) \leq \epsilon_b\},$$

where $i = 1, ..., n$ and $\epsilon_b = \min\{\epsilon, |P_{\mathcal{S}_k}[(\nabla m_k)(i)]|\}$, the minimum of the required accuracy for termination ϵ and the absolute value of the appropriate model gradient component projected onto $\mathcal{S}_k$. Considering the combined measure ϵ_b on the bounds l and u indeed enables us to define not only currently active bounds but also "nearly-active" bounds which are presumed to become active in the next local minimization problem. If the set $\mathcal{L}_k \cup \mathcal{U}_k$ is non-empty, the minimization is then restricted to the affine subspace

$$\mathcal{S}_k = \{x \in \mathcal{F} \mid x_k(i) = l(i) \;\text{ for }\; i \in \mathcal{L}_k \;\text{ and }\; x_k(i) = u(i) \;\text{ for }\; i \in \mathcal{U}_k\},$$

and the number of free variables consequently reduces to $n_{free} = n - |\mathcal{L}_k \cup \mathcal{U}_k|$.

To pursue minimization in $\mathcal{S}_k$, a new linear model has to be built to initiate the computation of the first step, and the interpolation set for this model must consist of points lying exactly in this subspace. In an attempt to use all the available information when entering the subspace, all points of $\mathcal{X}_k$ lying inside a $\pm\epsilon_b$-region of the active bounds are projected on $\mathcal{S}_k$. To save function evaluations, function values corresponding to these projected points are not recomputed but replaced by the relevant current values, giving rise to points with approximate function values that we call "dummy" points. Specifically, we define

$$\mathcal{A}_k = \{y \in \mathcal{X}_k \mid 0 < |y(i) - l(i)| \leq \epsilon_b, \forall i \in \mathcal{L}_k \text{ and } 0 < |u(i) - y(i)| \leq \epsilon_b, \forall i \in \mathcal{U}_k\},$$

the set of points that are close to the active bounds (but not on these). All points $y \in \mathcal{A}_k$ are then projected on $\mathcal{S}_k$, yielding $y_s = P_{\mathcal{S}_k}(y)$, and these "dummy" points $\{y_s\}$ are added to $\mathcal{X}_k$ with associated function values given by $\{m_k(y_s)\}$. An exception is made when the current best point x_k belongs to $\mathcal{A}_k$ and is thus projected onto $\mathcal{S}_k$: the objective function is then evaluated at the projected point $P_{\mathcal{S}_k}(x_k)$, rather than m_k. If the new function value is such that the projected point is not the current best point, we refrain from further exploring the subspace $\mathcal{S}_k$, otherwise the current best point consequently changes to the projected point in $\mathcal{S}_k$.

The technique described above in Section 3.2 is then used to select a well-poised interpolation set $\mathcal{Y}_0$ in $\mathcal{Z}_k = \mathcal{X}_k \cap \mathcal{S}_k$. The algorithm then proceeds by recursively calling itself in the subspace $\mathcal{S}_k$, as indicated in Step 2.4 of Algorithm 1.

While minimizing in $\mathcal{S}_k$, dummy points are successively replaced by real points with high priority (see Section 3.5 below). Moreover, we ensure that there is no dummy point in the interpolation set at a potential subspace solution: if the interpolation set still contains dummy points at this stage, the true function values are computed to ensure that convergence in $\mathcal{S}_k$ is solely based on real function values.

Once the algorithm has converged to an approximate solution of the problem restricted to $\mathcal{S}_k$, it must be verified whether it is also an approximate solution for the full-dimensional problem (after adding the fixed components). Therefore, a safely nondegenerate full-space interpolation set of degree $n+1$ in an ϵ-neighbourhood around the suspected solution x^* is constructed following the technique described in Section 3.2. After computing the associated model, its gradient is checked for convergence. If convergence can not be declared the minimization is then restarted in $\mathbb{R}^n$.

3.4 Local solver

To minimize the interpolation model m_k inside the intersection of the trust region and the bounds at each iteration, a simple projected truncated conjugate-gradient algorithm is used, as in the LANCELOT package [5]. As is standard for such techniques, the set of active bounds is never reduced and a piecewise linesearch is performed on the path defined by the projection of the current search direction onto the feasible set $\mathcal{F}$ if a new bound is hit in the course of the conjugate-gradient calculation. The computation is also stopped as soon as the iterates leave the trust region.

3.5 Defining the next iterate

At each iteration of a classical trust-region method, a new trial point x_k^+ is computed by minimizing the interpolation model m_k inside the trust-region Δ_k. The point x_k^+ is accepted to be the new iterate x_{k+1} if the ratio ρ between achieved and predicted reduction (4) exceeds a constant η_1. In this case, the iteration is declared successful. Otherwise, the iteration is unsuccessful. Following [24], we note that an unsuccessful iteration can either result from a too large trust region or from a badly poised $\mathcal{Y}_k$. For this reason, we always try first to improve the geometry by replacing an appropriate point from the set and only if we cannot find such a point, is the trust-region radius decreased. We now describe the details of this replacement/updating procedure (Step 5 of Algorithm 1).

The first step (after unsetting the `illcond` flag if necessary) is to check whether the current model degree is already quadratic. If this is not the case, the size of the interpolation set is augmented by the new trial point when possible, i.e. when appending the trial point doesn't deteriorate the poisedness of $\mathcal{Y}_k$ too much. This verified by checking that $\kappa(\hat{M})$, the condition of the system matrix after appending the trial point, does not exceed a certain threshold κ_{illcond}. If this threshold is exceeded, the trial is not added to $\mathcal{Y}_k$ and the flag `illcond` is set. If the iteration is successful, we also have to update the current best iterate and thus the center of the interpolation set.

When the model is either quadratic or it is not yet quadratic but the flag `illcond` is set, we try to replace the dummy points in the current interpolation set to avoid keeping approximate information in the model for too long. If there are any dummy points in the current interpolation set for which $\ell_{k,j}(x_k^+)$, the value of the associated Lagrange polynomial evaluated at the trial point, is nonzero, the dummy point for which this last value is largest in absolute value is replaced by x_k^+. If the current iteration is successful, we update the iterate by $x_{k+1} = x_k^+$ and the trust-region radius by

$$\Delta_{k+1} = \min(\max(\gamma_3 \|s_k\|_\infty, \Delta_k), \Delta_{\max}). \tag{14}$$

Otherwise we define $x_{k+1} = x_k$ and keep $\Delta_{k+1} = \Delta_k$.

If the trial point could not yet be included in the interpolation set, we apply the strategy described by Scheinberg and Toint in [24, Algorithm 2] for the unconstrained case, which we now recall. If the iteration is successful, we define, as above, $x_{k+1} = x_k^+$ and update the radius by (14). In the interpolation set, one point $y_{k,r}$ is then replaced by the trial point $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ for

$$y_{k,r} = \arg \max_{y_{k,j} \in \mathcal{Y}_k} \|y_{k,j} - x_k^+\|_2^2 |\ell_{k,j}(x_k^+)|.$$

In the unsuccessful case, i.e. when $\rho < \eta_1$, we still attempt to include the trial point in the interpolation set to improve its geometry. To do so, a point from $\mathcal{Y}_k \setminus \{x_k\}$ has to be replaced by x_k^+ and we first

attempt to replace a far interpolation point. If the set

$$\mathcal{F}_k \stackrel{\rm def}{=} \{y_{k,j} \in \mathcal{Y}_k \ \mid \ \|y_{k,j} - x_k\|_\infty > \beta\Delta_k \text{ and } \ell_{k,j}(x_k^+) \neq 0\} \tag{15}$$

is non-empty, where $\beta \geq 1$, then we set $x_{k+1} = x_k$, $\Delta_{k+1} = \Delta_k$ and define the new interpolation set by $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is an index of any point in $\mathcal{F}_k$, for instance such that

$$y_{k,r} = \arg \max_{y_{k,j} \in \mathcal{F}_k} \|y_{k,j} - x_k^+\|_2^2 |\ell_{k,j}(x_k^+)|$$

or

$$y_{k,r} = \arg \max_{y_{k,j} \in \mathcal{F}_k} \|y_{k,j} - x_k^+\|_2^2. \tag{16}$$

If the set $\mathcal{F}_k$ is empty and the set

$$\mathcal{C}_k \stackrel{\rm def}{=} \{y_{k,j} \in \mathcal{Y}_k \setminus \{x_k\} \text{ such that } \|y_{k,j} - x_k\|_\infty \leq \beta\Delta_k \text{ and } \ell_{k,j}(x_k^+) > \Lambda_C\} \tag{17}$$

is non-empty, where $\Lambda_C > 1$ is defined by the user, we then set $x_{k+1} = x_k, \Delta_{k+1} = \Delta_k$ and define the new interpolation set $\mathcal{Y}_{k+1} = \mathcal{Y}_k \setminus \{y_{k,r}\} \cup \{x_k^+\}$ where r is the index of any point in $\mathcal{C}_k$, for instance such that

$$y_{k,r} = \arg \max_{y_{k,j} \in \mathcal{C}_k} \|y_{k,j} - x_k^+\|_2^2 |\ell_{k,j}(x_k^+)|$$

or

$$y_{k,r} = \arg \max_{y_{k,j} \in \mathcal{C}_k} |\ell_{k,j}(x_k^+)|. \tag{18}$$

(The current default in our algorithm, based on our numerical experience, is to choose (16) and (18).)

If the trial point could finally not be included into the interpolation set under the above conditions, it implies that the interpolation set must be reasonably poised, as otherwise we could have improved it. As a consequence, we set $x_{k+1} = x_k$, $\mathcal{Y}_{k+1} = \mathcal{Y}_k$ and reduce the trust-region radius such that $\Delta_{k+1} \in [\gamma_1\Delta_k, \gamma_2\Delta_k]$, where $0 < \gamma_1 \leq \gamma_2 < 1$. In practice, interpolation is used to define the new trust-region radius as described in [5, page 116].

3.6 Re-entering a subspace

We have stated in Step 2.1 of Algorithm 1 that we never re-enter a subspace $\mathcal{S}_k$ which has already been explored. We now justify that feature.

Suppose that convergence is declared in $\mathcal{S}_k$ and that a new model of the required degree—the default is linear—is built at x^* in $\mathbb{R}^n$. Assume also that x^* is an acceptable solution of the full-space problem. It may then happen that the model gradient $\nabla m_k(x^*)$ is too large to declare convergence in $\mathbb{R}^n$, because m_k is not a sufficiently accurate model even if the interpolation set is well poised. Since convergence is not detected, the algorithm has to proceed, re-enter the subspace it just left and thus loop without progressing. We now show that this can only happen if the trust-region radius is too large, in which case reducing it is the appropriate strategy. Indeed, we know (Theorem 2.11, p. 29, in [10]) that, for linear models,

$$\|\nabla_x f(y) - \nabla_x m(y)\|_2 \leq \kappa_{eg}\Delta, \tag{19}$$

where κ_{eg} is given by

$$\kappa_{eg} = \nu(1 + n^{\frac{1}{2}}\|\hat{L}^{-1}\|_2/2), \tag{20}$$

with $\hat{L} = \frac{L}{\Delta} = \frac{1}{\Delta}[y_2 - y_1, ..., y_n - y_1]$ and ∇f is Lipschitz continuous with constant $\nu > 0$. As a consequence, a big difference between ∇m_k and ∇f can only occur either because the Lipschitz constant or the trust region radius are too large. As the size of Lipschitz constant is beyond our control, reducing Δ_k must solve the problem because (19) implies that the model gradient will converge to the true one and either convergence will be declared or the algorithm may proceed with a more accurate model.

4 Practical implementation issues

The description of our algorithm in the previous chapter leaves a number of practical questions open. In this section, we briefly report some further details about the particular implementation of the algorithm whose numerical performance is reported in Section 5 of this paper.

4.1 Representation of the Lagrange polynomials

As $\mathcal{Y}$ varies, the code maintains a QR factorization

$$M(\phi, \mathcal{Y})^T = Q_{\mathcal{Y}} R_{\mathcal{Y}}$$

of the matrix of the system (7) (or of its shifted version $\hat{M}$ if appropriate), where the basis ϕ is that of the monomials. If the vector ℓ_i contains the coefficients of the Lagrange polynomial $\ell_i(x)$ associated to $\mathcal{Y}$, their definition implies that they have to satisfy $M(\phi, \mathcal{Y})\ell_i = e_i$ and hence may be retrieved from the formula $\ell_i = Q_{\mathcal{Y}} R_{\mathcal{Y}}^{-T} e_i$.

4.2 Handling fixed variables

In practice, it is often very convenient for users of an optimization package, to be able to fix the value of certain variables. Hence, we have that

$$l(i) = x_0(i) = u(i).$$

In order to handle such a case, we check for variables where $u(i) - l(i) = 0$. The corresponding indices i together with their fixed values $x_0(i)$ are then stored in a vector for use when evaluating the objective function throughout the calculation, but the associated variables are otherwise excluded from the minimization process.

4.3 Model gradient as stopping criterion

As indicated above, the model gradient $\nabla_x m_k(x)$ is used to check convergence to a first-order critical point, in the sense that we verify the inequality

$$\|P_{\mathcal{F}}(x_k - \nabla m_k(x_k)) - x_k\|_\infty \leq \epsilon, \tag{21}$$

which [15] have shown to correspond to a suitable measure of backward error for bound-constrained problems. Moreover, we have, that

$$\begin{aligned}
&\|P_{\mathcal{F}}(x_k - \nabla f(x_k)) - x_k\|_\infty \\
&\qquad = \|P_{\mathcal{F}}(x_k - \nabla f(x_k) - \nabla m_k(x_k) + \nabla m_k(x_k)) - x_k\|_\infty \\
&\qquad \leq \|P_{\mathcal{F}}(x_k - \nabla m_k(x_k)) - x_k\|_\infty + \|\nabla m_k(x_k) - \nabla f(x_k)\|_\infty \\
&\qquad \leq \|P_{\mathcal{F}}(x_k - \nabla m_k(x_k)) - x_k\|_\infty + \|\nabla m_k(x_k) - \nabla f(x_k)\|_2,
\end{aligned}$$

and, using (19), we deduce that the left-hand side of this inequality can be made small if (21) holds and Δ_k is sufficiently small. In practice, we require the interpolation points $y_i, i = 1, ..., p$ used to build $m_k(x)$ to be contained in the ball $\mathcal{B}(x_k, \epsilon)$ and $\mathcal{Y}_k$ is poised enough to ensure $\kappa_{eg}\Lambda\Delta_k \leq \epsilon$ for some user-defined constant $\kappa_{eg} > 0$.

5 Numerical experiments

The algorithm described has been implemented in the BC-DFO Matlab code and all numerical experiments reported below were run on a single processor workstation. As the time to compute the objective function values in derivative-free optimization typically dominates other costs of the algorithm, our results will be presented in terms of number of function evaluations. In what follows, BC-DFO is compared with the excellent packages NEWUOA [22] and BOBYQA [23] developed by M.J.D. Powell, where BOBYQA is able to handle bound constraints and NEWUOA supersedes BOBYQA in solving unconstrained problems.

5.1 Default parameters

In BC-DFO, we fixed the trust-region parameters to $\eta_1 = 0.0001, \eta_2 = 0.9, \gamma_1 = 0.01, \gamma_2 = 0.5$ and $\gamma_3 = 1.5$. The initial trust-region radius Δ_0 is set to 1, as suggested in Section 17.2 of [6]. To build a sufficiently well-poised set in the modified greedy algorithm, we set the threshold $\kappa_{th} = 0.005$. After appending a point to an incomplete interpolation set, we check the condition of the shifted and scaled system matrix $\kappa(\hat{M})$ to be smaller than $\kappa_{\text{illcond}} = 10^{15}$. To divide the interpolation set into far and close points when incorporating the new trial point, we set $\beta = 1$. When replacing a close interpolation point, we use the parameter $\Lambda_C = 1.2$ to ensure an improvement of the interpolation set geometry. For declaring convergence, the desired accuracy on the projected gradient norm is set to $\epsilon = 10^{-5}$ and the parameter of the tolerated error on the gradient is set to $\kappa_{eg} = 0.1$.

We always used the default parameters for the codes NEWUOA and BOBYQA. We run BOBYQA with a number of $m = 2n + 1$ interpolation points using the Frobenius norm approach and NEWUOA with a full quadratic model, as these two options give the best results for these solvers, out of the choice $m \in [n + 1, 2n + 1, \frac{1}{2}(n + 1)(n + 2)]$.

5.2 Test problems

The CUTEr test environment (see [14]) is used in our experiments. To compare BC-DFO and NEWUOA on unconstrained problems, we chose to use the test problems from the CUTEr test collection which were selected in [13]. Two problems[2] were excluded from the test set because they contain fixed variables and NEWUOA does not provide facilities to handle such cases and one listed problem[3] contains bounds. After running all problems in this test set, three problems[4] were removed because the solvers converged to different solutions, making a comparison meaningless. A total of 54 unconstrained problems were thus considered.

For the bound-constrained case, we took all bound-constrained problems provided by the CUTEr collection with a size of at most 30 variables. We could not consider problems containing fixed variables because BOBYQA, as NEWUOA, does not provide the required facilities. Furthermore, in order to avoid too many problems of the same kind, we chose randomly four of the 26 bound-constrained PALMER problems provided. After running BC-DFO and BOBYQA on these 53 remaining problems, six of them[5] had to be excluded from our comparison because the two algorithms converged to different solutions, giving a final test set of 47 bound-constrained problems.

The detailed list of all considered bound-constrained problems and their characteristics is provided in Table 1 in the Appendix of this paper.

5.3 A common stopping criterion

As BOBYQA and NEWUOA use different stopping criteria from those of BC-DFO, an independent criterion needs to be applied for the comparison. For this reason, we use the optimal objective function value computed by the TRON package [16] (using first and second derivatives) as a reference for our bound-constrained experiments. In the experiments with unconstrained problems we take the optimal objective function value computed by the KNITRO package [3] used in the paper of Fasano, Morales and Nocedal [13][6]. We take the number of function evaluations needed until a prescribed number of correct significant figures in the objective value was attained.

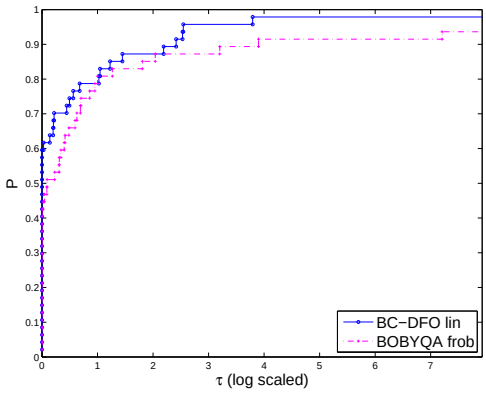
To provide a fair comparison, we followed the testing framework proposed by Dolan, Moré, and Munson in [12]. In this framework, the solvers are run first with their own default stopping criterion. If, for a given problem, convergence of one of the solvers to the common stopping criterion can't be declared with this configuration, the stopping criterion for this solver is strenghtened and the run repeated using the more stringent criterion. For a few test problems, BOBYQA and NEWUOA were run several times while decreasing its own stopping criterion (namely ρ_{end}) after each run, trying to attain the commonly required accuracy in the objective function value. This procedure was a successful for a subset of the

[2] BIGGS3, BOX2
[3] CHEBYQAD
[4] ENGVAL2, HATFLDD, HATFLDE
[5] EG1, EXPLIN, HART6, KOEBHELB, MAXLIKA, WEEDS
[6] Unfortunately, we could not conduct a detailed comparison of our results with the method proposed by these authors.

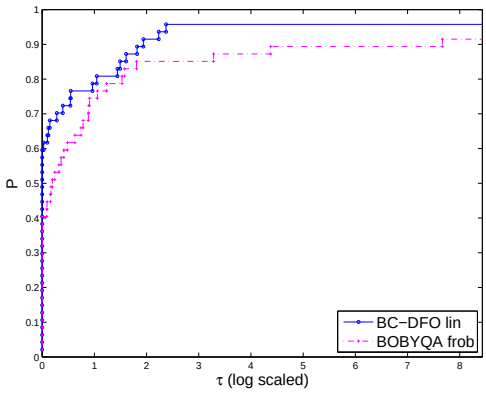
problems, for others the limit of function evaluations (15000) was reached or, in some cases, NEWUOA reported failure to reduce the model in a trust region step. No time limitation was set.

5.4 Performance of BC-DFO

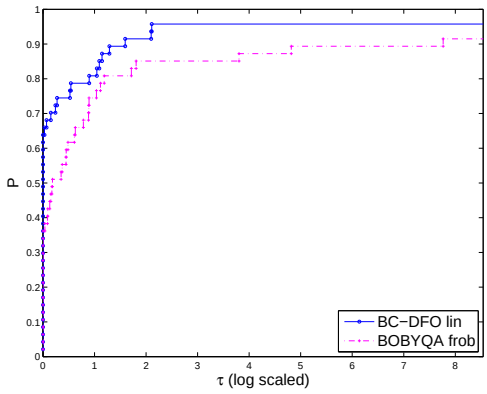
We now report our results using performance profiles (see [11]). Such profiles compare the number of function evaluations needed by each solver to achieve the desired accuracy in the objective function value. We use four different levels of accuracy: 2, 4, 6 and 8 significant figures in $f(x^*)$.



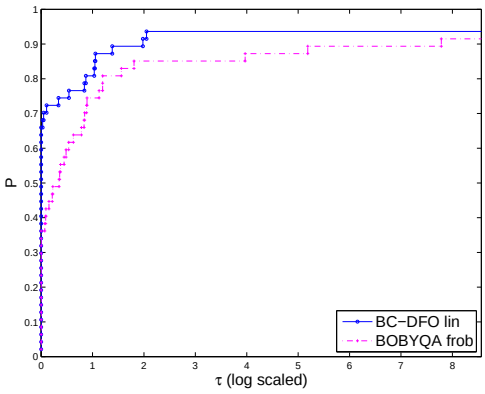
(a) 2 digits of accuracy in final function value



(b) 4 digits of accuracy in final function value



(c) 6 digits of accuracy in final function value



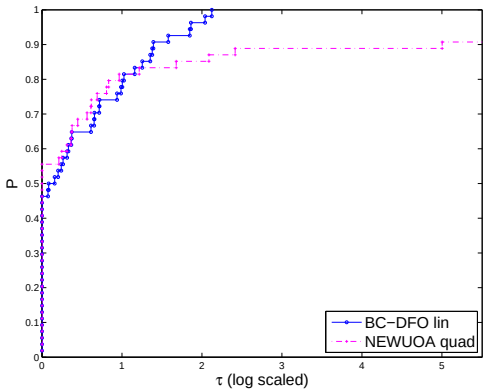
(d) 8 digits of accuracy in final function value

Figure 1: Experiments on a bound-constrained subset of CUTEr

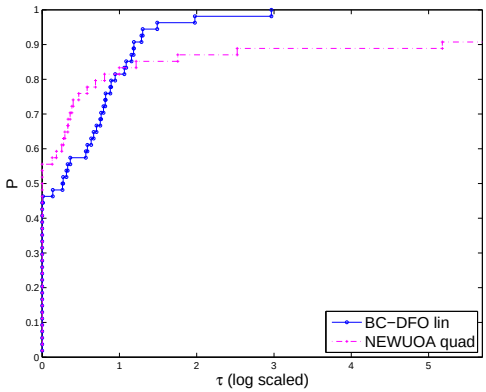
The profiles reported in Figures 1(a)-1(d) show that BC-DFO compares well with BOBYQA in the bound-constrained experiments. For low accuracy (in Figure 1(a)), BOBYQA manages to solve 44 of the 47 test problems including PALMER3E which it couldn't solve in 15000 function evaluations to a more accurate level. BC-DFO fails to solve two test problems in all four cases. The overall conclusion is that both solvers are equally robust, but that BC-DFO's dominance increases with the desired level of accuracy. For the case where 2 correct significant figures are required, BC-DFO solves 60% of the test cases faster than BOBYQA and BOBYQA solves 42 % of the problems faster. For 8 correct significant figures, BC-DFO solves 66% of the test cases faster, and BOBYQA solves 36 % of the problems faster. The performance of both codes does not vary significantly between requiring 4 or 6 correct significant figures.

In the six cases where BOBYQA and BC-DFO converged to different critical points, BC-DFO converged to a lower optimal value than BOBYQA in three cases and BOBYQA found a lower function value in three cases. This is due to the existence of multiple minima, but also to the fact that BC-DFO sometimes checks convergence in the full-space without taking second order information into account (after converging inside a subspace). This creates the possibility to declare convergence at a saddle point

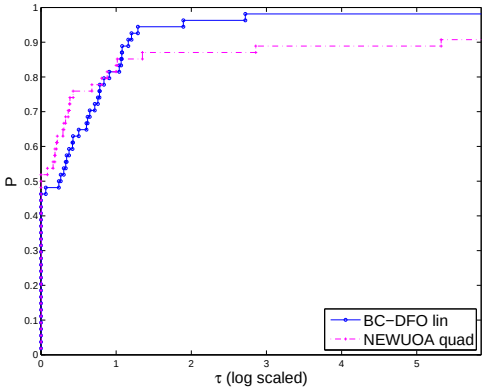
which is a minimum in the explored subspace. This is a slight disadvantage of our current implementation and can be circumvented (at some cost) by requiring that a full quadratic model is built before declaring termination.



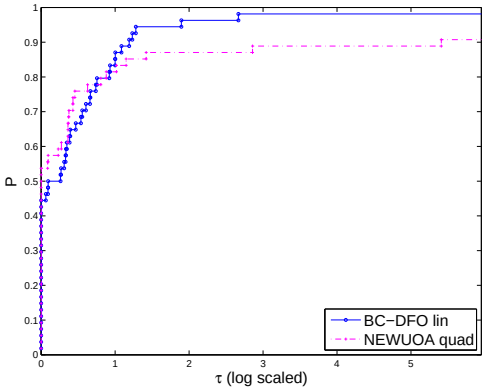
(a) 2 digits of accuracy in final function value



(b) 4 digits of accuracy in final function value



(c) 6 digits of accuracy in final function value



(d) 8 digits of accuracy in final function value

Figure 2: Experiments on an unconstrained subset of CUTEr

The conclusions are different for unconstrained problems. As Figure 2 shows, BC-DFO appears to be more robust but less efficient than NEWUOA, irrespective of the accuracy required. For instance, NEWUOA solves 52 % of the problems faster and BC-DFO solves 46 % of the test cases faster, when 6 digits of accuracy are requested.

Tables 3 and 4 in the appendix contain the detailed results for each code and each problem for the various accuracy levels.

6 Concluding remarks

We have presented an implementation of an active-set trust-region method for bound-constrained optimization without derivatives which uses the self-correction mechanism presented by [24]. The numerical experiments reported suggest that such an algorithm may prove to be both efficient and reliable.

Continued development of the BC-DFO code is expected. In particular, developments making use of the structure of the model's Hessian or extending the methodology to larger problem classes are considered.

Acknowledgements

The second author gratefully acknowledges the partial support of the ADTAO project funded by the "Sciences et Technologies pour l'Aéronautique et l'Espace (STAE)" Fundation (Toulouse, France) within the "Réseau Thématique de Recherche

Avancée (RTRA)".

References

[1] O. Burdakov. An MST-type algorithm for the optimal basis problem. Technical Report TR/PA/95/22, CERFACS, 42, av. G. Coriolis, 31057 Toulouse, France, 1995.

[2] O. Burdakov. A greedy algorithm for the optimal basis problem. *BIT*, 37(3):591–599, 1997.

[3] R. H. Byrd, J. Nocedal, and R. A. Waltz. KNITRO: An integrated package for nonlinear optimization. In *Large Scale Nonlinear Optimization*, pages 35–59, Heidelberg, Berlin, New York, 2006. Springer Verlag.

[4] P. G. Ciarlet and P. A. Raviart. General Lagrange and Hermite interpolation in $\mathbb{R}^n$ with applications to finite element methods. *Arch. Ration. Mech. Anal.*, 46:177–199, 1972.

[5] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*. Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

[6] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. Number 01 in MPS-SIAM Series on Optimization. SIAM, Philadelphia, USA, 2000.

[7] A. R. Conn, K. Scheinberg, and Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In A. Iserles and M. Buhmann, editors, *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, pages 83–108, Cambridge, England, 1997. Cambridge University Press.

[8] A. R. Conn, K. Scheinberg, and Ph. L. Toint. A derivative free optimization algorithm in practice. Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, Missouri, September 2-4, 1998.

[9] A. R. Conn, K. Scheinberg, and L. N. Vicente. Geometry of interpolation sets in derivative free optimization. *Mathematical Programming, Series B*, 111(1-2), 2008.

[10] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-free Optimization*. MPS-SIAM Optimization series. SIAM, Philadelphia, USA, 2008.

[11] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[12] E. D. Dolan, J. J. Moré, and T. S. Munson. Optimality measures for performance profiles. *SIAM Journal on Optimization*, 16(3):891–909, 2006.

[13] G. Fasano, J. Nocedal, and J.-L. Morales. On the geometry phase in model-based algorithms for derivative-free optimization. *Optimization Methods and Software*, (to appear), 2009.

[14] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr, a constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, 2003.

[15] S. Gratton, M. Mouffe, and Ph. L. Toint. Stopping rules and backward error analysis for bound-constrained optimization. Technical Report 09/13, Department of Mathematics, FUNDP - University of Namur, Namur, Belgium, 2009.

[16] C. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[17] M. Marazzi and J. Nocedal. Wedge trust region methods for derivative free optimization. *Mathematical Programming, Series A*, 91(2):289–300, 2002.

[18] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. Technical Report ANL-MCS-P1471-1207, Mathematics and Computer Science, Argonne National Laboratory, Argonne, Illinois, USA, 2007.

[19] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In S. Gomez and J. P. Hennart, editors, *Advances in Optimization and Numerical Analysis, Proceedings of the Sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico*, volume 275, pages 51–67, Dordrecht, The Netherlands, 1994. Kluwer Academic Publishers.

[20] M. J. D. Powell. A direct search optimization method that models the objective by quadratic interpolation. Presentation at the 5th Stockholm Optimization Days, Stockholm, 1994.

[21] M. J. D. Powell. The use of band matrices for second derivative approximations in trust region algorithms. In Y. Yuan, editor, *Advances in Nonlinear Programming*, pages 3–28, Dordrecht, The Netherlands, 1998. Kluwer Academic Publishers.

[22] M. J. D. Powell. Developments of NEWUOA for minimization without derivatives. *IMA Journal of Numerical Analysis*, 28(4):649–664, 2008.

[23] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, Department of Applied Mathematics and Theoretical Physics, Cambridge University, Cambridge, England, 2009.

[24] K. Scheinberg and Ph. L. Toint. Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. Technical Report TR09/06, Department of Mathematics, FUNDP - University of Namur, Namur, Belgium, 2009.

A Test problems

Table 1 depicts the bound-constrained test problems taken from the CUTEr testing environment for running our numerical experiments. It shows the name and dimension of the problem and gives specific details on the number of free variables, the number of variables which are bounded from below, those which are bounded from above and the number of variables which are bounded from below and above.

name	n	free vars	lbound	ubound	l+ubound	f^*
3PK	30		30			1.72011856739612E+00
BIGGSB1	25	1			24	1.50000000000000E-02
BQP1VAR	1				1	0.00000000000000E+00
CAMEL6	2				2	-1.03162845348988E+00
CHARDIS0	18				18	3.95170009709151E-27
CHEBYQAD	4				4	2.56057805386809E-22
CHENHARK	10		10			-2.00000000000000E+00
CVXBQP1	10				10	2.47500000000000E+00
EXPLIN2	12				13	-7.09247239439664E+03
EXPQUAD	12	6			6	-4.20107186489211E+03
HARKERP2	10		10			-5.00000000000000E-01
HATFLDA	4		4			1.61711062151584E-25
HATFLDB	4		3		1	5.57280900008425E-03
HATFLDC	25	1			24	3.43494690036517E-27
HIMMELP1	2				2	-6.205393553382574E+01
HS1	2	1	1			7.13660798093435E-24
HS110	10				10	-4.57784755318868E+01
HS2	2	1	1			4.94122931798918E+00
HS25	3				3	1.81845940377455E-16
HS3	2	1	1			1.97215226305253E-36
HS38	4				4	2.02675622883580E-28
HS3MOD	2	1	1			0.00000000000000E+00
HS4	2		2			2.66666666400000E+00
HS45	5				5	1.00000000040000E+00
HS5	2				2	-1.91322295498104E+00
LINVERSE	19	9	10			6.00000000022758E+00
LOGROS	2		2			0.00000000000000E+00
MCCORMCK	10				10	-9.59800619474625E+00
MDHOLE	2	1	1			7.52316384526264E-35
NCVXBQP1	10				10	-2.20500000000000E+04
NCVXBQP2	10				10	-1.43818650000000E+04
NCVXBQP3	10				10	-1.19578050000000E+04
NONSCOMP	25				25	4.42431972353647E-14
OSLBQP	8		5			6.25000000000000E+00
PALMER1A	6	4	2			8.98830583652624E-02
PALMER2B	4	2	2			6.23266904205002E-01
PALMER3E	8	7	1			1.38380645324899e-01
PALMER4	4	1	3			2.28538322742966E+03
PALMER4A	6	4	2			4.06061409159725E-02
PROBPENL	10				10	-2.11912948080046E+05
PSPDOC	4	3		1		2.41421356237309E+00
QUDLIN	12				12	-7.20000000000000E+03
S368	8				8	-9.37500000000000E-01
SIMBQP	2	1			1	0.00000000000000E+00
SINEALI	4				4	-2.83870492243045E+02
SPECAN	9				9	1.64565541040970E-13
YFIT	3	2	1			6.66972055747565E-13

Table 1: Bound-constrained CUTEr test problems

name	n	f^*
ALLINITU	4	5.74438491032034E+00
ARGLINB	10	4.63414634146338E+00
ARGLINC	10	6.13513513513513E+00
ARWHEAD	16	5.32907051820075E-15
BARD	3	8.21487730657899E-3
BDQRTIC	10	1.82811617535935E+01
BEALE	2	1.03537993810258E-30
BIGGS6	6	5.49981608181981E-16
BOX3	3	1.85236429640516E-20
BRKMCC	2	1.69042679196450E-1
BROWNAL	10	1.49563496755546E-16
BROWNDEN	4	8.58222016263563E+04
CHNROSNB	10	1.21589148855346E-19
CRAGGLVY	10	1.88656589666311E+00
CUBE	2	5.37959996529976E-25
DENSCHND	3	2.15818302178292E-4
DENSCHNE	3	1.29096866601748e-18
DENSCHNF	2	6.51324621983021E-22
DIXMAANC	15	1.00000000000000E+00
DIXMAANG	15	1.00000000000000E+00
DIXMAANI	15	1.00000000000000E+00
DIXMAANK	15	1.00000000000000E+00
DIXON3DQ	10	2.95822839457879E-31
DQDRTIC	10	5.91645678915759E-29
ENGVAL1	2	0.00000000000000E+00
EXPFIT	2	2.40510593999058E-1
FREUROTH	10	1.01406407257452E+03
GENHUMPS	5	9.31205762089110E-33
GULF	3	5.70816776659866E-29
HAIRY	2	2.00000000000000E+01
HELIX	3	1.81767515239766E-28
HILBERTA	2	1.51145573593758E-20
HIMMELBF	4	3.18571748791125E+02
HIMMELBG	2	1.17043537660229E-27
JENSMP	2	1.24362182355615e+02
KOWOSB	4	3.07505603849238E-4
MANCINO	10	1.24143266331958E-19
MARATOSB	2	-1.00000006249999E+00
MEXHAT	2	-4.01000000000000E-2
MOREBV	10	1.85746736253704E-24
NASTY	2	1.53409170790554e-72
OSBORNEB	11	4.01377362935478E-2
PALMER1C	8	9.75979912629838E-2
PALMER3C	8	1.95376385131058E-2
PALMER5C	6	2.12808666605511E+00
PALMER8C	8	1.59768063470262E-1
POWER	10	6.03971630559837E-31
ROSENBR	2	3.74397564313947E-21
SINEVAL	2	7.09027697800298E-20
SINGULAR	4	6.66638187151797e-12
SISSER	2	1.06051492721772E-12
VARDIM	10	1.59507305257139E-26
YFITU	3	6.66972048929030E-13
ZANGWIL2	2	-1.82000000000000E+01

Table 2: Unconstrained CUTEr test problems

B Test results

The results of the bound-constrained and unconstrained testing can be seen in Table 3 and 4 below. Both tables show the name and number of variables of the problem, and the number of function evaluations needed by each solver to attain six significant figures of the objective function value f^*, computed using the package TRON or KNITRO in the bound- or unconstrained case, respectively.

		nf BC-DFO				nf BOBYQA			
name	n	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig
3PK	30	8834	8834	8927	8959	failed	failed	failed	failed
BIGGSB1	25	35	35	35	35	144	341	489	548
BQP1VAR	1	2	2	2	2	7	7	7	7
CAMEL6	2	16	26	29	35	17	29	37	41
CHARDIS0	18	420	420	420	420	92	119	139	161
CHEBYQAD	4	208	235	259	265	15	50	60	64
CHENHARK	10	133	134	134	134	90	121	151	172
CVXBQP1	10	21	21	21	21	26	39	39	39
EXPLIN2	12	93	99	115	118	224	233	236	258
EXPQUAD	12	613	813	857	911	105	157	200	231
HARKERP2	10	62	63	63	63	64	67	67	67
HATFLDA	4	5	5	5	5	46	104	141	182
HATFLDB	4	64	76	81	91	40	52	67	72
HATFLDC	25	699	753	768	790	131	247	360	441
HIMMELP1	2	15	25	26	28	13	19	22	26
HS1	2	133	138	147	148	135	158	167	172
HS110	10	196	316	398	409	144	260	436	521
HS2	2	25	38	38	39	33	35	39	39
HS25	3	59	254	298	failed	107	734	978	995
HS3	2	3	8	9	9	6	9	10	10
HS38	4	322	342	347	347	408	440	474	503
HS3MOD	2	13	13	13	13	21	24	24	24
HS4	2	5	575	57	7	7			
HS45	5	15	15	15	15	16	16	16	16
HS5	2	8	8	20	20	13	15	18	21
LINVERSE	19	792	905	936	958	137	236	651	2828
LOGROS	2	3	3	3	3	443	609	652	661
MCCORMCK	10	59	152	166	178	29	54	75	87
MDHOLE	2	165	165	165	165	220	220	225	225
NCVXBQP1	10	16	18	18	18	31	31	31	31
NCVXBQP2	10	64	64	64	64	31	31	31	31
NCVXBQP3	10	50	50	50	50	49	49	49	49
NONSCOMP	25	859	907	921	937	779	1072	1406	1684
OSLBQP	8	31	32	32	32	22	22	22	22
PALMER1A	6	10842	11143	11190	11195	failed	failed	failed	failed
PALMER2B	4	1192	1198	1201	1206	1037	1118	1144	1166
PALMER3E	8	82	82	82	82	1223	failed	failed	failed
PALMER4	4	145	160	162	170	62	82	87	93
PALMER4A	6	2753	3445	3511	3536	2366	5767	7606	8105
PROBPENL	10	failed	failed	failed	failed	failed	failed	failed	failed
PSPDOC	4	35	44	44	50	41	55	57	65
QUDLIN	12	22	22	22	22	34	34	34	34
S368	8	90	136	154	158	33	50	63	76
SIMBQP	2	12	12	12	12	12	12	12	12
SINEALI	4	29	87	209	242	36	260	387	432
SPECAN	9	failed	failed	failed	failed	697	765	820	881
YFIT	3	897	965	981	985	1356	2011	2237	2257

Table 3: Results on bound-constrained test set with 2, 4, 6 and 8 significant figures attained in f^*

		nf BC-DFO				nf NEWUOA			
name	n	2 fig	4 fig	6 fig	8 fig	2 fig	4 fig	6 fig	8 fig
ALLINITU	4	53	66	103	106	50	60	67	73
ARGLINB	10	88	88	88	88	70	70	70	70
ARGLINC	8	84	84	84	84	70	70	70	70
ARWHEAD	15	16	16	16	16	513	579	641	680
BARD	3	62	73	80	83	26	46	51	60
BDQRTIC	10	347	435	578	593	181	236	276	296
BEALE	2	63	69	69	73	24	33	42	46
BIGGS6	6	442	653	715	820	148	265	400	624
BOX3	3	46	56	58	63	18	33	47	59
BRKMCC	2	7	13	13	16	7	7	15	15
BROWNAL	10	320	377	398	432	88	166	212	256
BROWNDEN	4	93	99	107	111	59	66	80	85
CHNROSNB	15	1180	1264	1335	1341	717	776	790	803
CRAGGLVY	10	919	1199	1302	1324	458	538	616	662
CUBE	2	77	103	110	112	105	130	138	151
DENSCHND	3	58	78	86	100	45	45	64	68
DENSCHNE	3	67	76	87	91	87	92	99	110
DENSCHNF	2	15	18	22	22	23	25	25	34
DIXMAANC	15	334	375	400	444	415	447	465	472
DIXMAANG	15	566	654	669	691	479	508	528	543
DIXMAANI	15	810	813	831	971	398	460	483	509
DIXMAANK	15	570	779	799	825	736	773	848	881
DIXON3DQ	10	31	31	31	31	72	72	79	83
DQDRTIC	10	44	44	44	44	71	71	81	81
ENGVAL1	2	4	4	4	4	17	23	29	29
EXPFIT	2	65	68	70	72	25	32	34	38
FREUROTH	10	345	553	560	571	174	197	229	243
GENHUMPS	5	705	1422	1602	1684	613	739	760	793
GULF	3	197	307	338	344	187	336	378	404
HAIRY	2	46	54	55	57	68	68	74	74
HELIX	3	57	66	68	70	66	75	84	90
HILBERTA	10	7	7	7	7	9	9	9	9
HIMMELBF	4	257	400	461	485	168	599	737	1076
HIMMELBG	2	22	32	35	35	14	19	23	23
JENSMP	2	36	44	48	49	10	25	28	31
KOWOSB	4	5	59	125	144	16	38	106	116
MANCINO	10	89	103	110	116	136	136	143	150
MARATOSB	2	2915	3240	3339	3361	5693	6471	6748	6813
MEXHAT	2	263	508	520	527	64	65	79	83
MOREBV	10	88	108	119	135	68	73	84	112
NASTY	2	3	3	3	3	failed	failed	failed	failed
OSBORNEB	11	1917	2564	2743	2764	858	1126	1190	1211
PALMER1C	8	68	68	failed	failed	failed	failed	failed	failed
PALMER3C	8	66	66	66	67	failed	failed	failed	failed
PALMER5C	6	46	46	46	46	37	37	44	44
PALMER8C	8	70	71	71	78	failed	failed	failed	failed
POWER	10	358	704	839	1117	218	289	375	460
ROSENBR	2	46	72	75	78	82	94	98	107
SINEVAL	2	171	177	178	182	203	217	218	234
SINGULAR	4	67	99	136	173	60	82	105	137
SISSER	2	3	8	16	26	16	27	32	35
VARDIM	10	2022	2278	2409	2412	464	579	648	648
YFITU	3	897	965	981	985	failed	failed	failed	failed
ZANGWIL2	2	4	4	4	4	7	7	7	7

Table 4: Results on unconstrained test set with 2, 4, 6 and 8 significant figures attained in f^*