Wedge Trust Region Methods for Derivative Free Optimization*

Marcelo Marazzi† Jorge Nocedal‡

26 October 2000

Report OTC 2000/10
Optimization Technology Center
Northwestern University, Evanston, IL 60208, USA

Abstract

A new method for derivative-free optimization is presented. It is designed for solving problems in which the objective function is smooth and the number of variables is moderate, but the gradient is not available. The method generates a model that interpolates the objective function at a set of sample points, and uses trust regions to promote convergence. The step-generation subproblem ensures that all the iterates satisfy a geometric condition and are therefore adequate for updating the model. The sample points are updated using a scheme that improves the accuracy of the interpolation model when needed. Two versions of the method are presented: one using linear models and the other using quadratic models. Numerical tests comparing the new approach with established methods for derivate-free optimization are reported.

*This work was supported by National Science Foundation grant CCR-9987818, and by Department of Energy grant DE-FG02-87ER25047-A004.

†Department of Industrial Engineering and Management Sciences, Northwestern University, 2145 Sheridan Road, Evanston, IL 60208-3119. marazzi@iems.northwestern.edu, www.iems.nwu.edu/~marazzi

‡Department of Electrical and Computer Engineering, Northwestern University, 2145 Sheridan Road, Evanston, IL 60208-3119. nocedal@ece.northwestern.edu, www.ece.northwestern.edu/~nocedal

1 Introduction

We are concerned with the problem of minimizing a smooth function f of several variables whose derivatives are unavailable. Formally,

$$\min_{x \in \mathbb{R}^n} \; f(x). \tag{1.1}$$

We restrict our attention to problems with a moderate number of variables, and assume that the cost of evaluating the function is much higher than the linear algebra required in the optimization iteration. Derivatives are not available in many applications for a variety of reasons. For example, the value $f(x)$ could be the result of a physical measurement, or the code that computes $f(x)$ could use different programming languages or include proprietary components that cannot be examined, making the use of automatic differentiation or the calculation of analytical derivatives impractical. An option for solving problems of this kind is to use gradient-based methods that employ finite-difference approximations to the gradient, and the algorithms we discuss here will be compared with that approach.

Several methods have been proposed, in addition to finite differences, for solving (1.1) when derivatives are not available. They include pattern-search, simulated annealing, and trust region methods based on interpolation models (see Powell [12] and Wright [14] for a survey of these techniques). Our approach belongs to the latter class: it forms a linear or quadratic model of the objective and makes use of trust regions to promote convergence. In contrast to the methods described in [4, 5, 11, 12], our method includes a constraint in the trust region problem that ensures that the position of all the points generated by the algorithm is such that they adequately define a linear or quadratic model. Since this additional constraint has the form of a wedge when the model is linear, we refer to our approach as a "wedge method".

Model-based trust region methods exploit the smoothness in the objective function and attempt to preserve the convergence properties of their gradient-based counterparts. A model $s \mapsto m_c(x_c + s)$ is created to approximate f around the current iterate x_c. The model is required to interpolate f at x_c, as well as at a set Σ_c of additional sample points, i.e.,

$$m_c(x_c) = f(x_c), \qquad m_c(y) = f(y) \quad \text{for all} \quad y \in \Sigma_c. \tag{1.2}$$

We can write these interpolation conditions as a linear system of equations whose unknowns are the coefficients of the model m_c.

For the linear system defined by (1.2) to be well defined, one must ensure that the position of the sample points is such that the rows of the linear system are linearly independent; we call this the *geometric condition*, and if it holds we say that the sample point set is *non-degenerate*. In the methods described in [4, 5, 11, 12], a step s_c is obtained by minimizing $m_c(x_c + s)$ subject to a trust region $\|s\|_2 \leq \Delta_c$, where the radius Δ_c is adjusted automatically according to established rules. After replacing one of the sample points by the new point $x_+ = x_c + s_c$ the geometric condition may, however, not be satisfied. To cope with this difficulty two types of iterations are performed to generate a new trial point x_+ (we use the nomenclature in [12]):

1. "minimization" iterations aimed at reducing f;

2. "simplex" iterations designed to define a model that approximates f more adequately.

If the trial point x_+ fails to reduce the value of f, the model m_c is considered to be a poor local approximation of f, and one of two courses of action is taken.

- If the set of sample points is nearly degenerate or some of the points interpolated by m_c are considered to be too far from x_c, then an improved sample set is required, and a simplex iteration is invoked. It returns a point that is close to x_c and that increases some measure of the goodness of the geometry of the simplex (e.g., the determinant of the system induced by (1.2)).

- Otherwise, the new point x_+ is considered to be too far from x_c for m_c to be an accurate approximation of f. Then the trust region radius is reduced and another minimization iteration is carried out.

Our method performs only one type of iteration. Instead of solving a standard trust region subproblem and taking special action if the new point $x_+ = x_c + s_c$ does not enjoy favorable geometric properties, we impose a geometric condition explicitly in the step computation procedure, thereby guaranteeing that the new set of points defines an adequate model. This, together with a mechanism that controls the accuracy of m_c in approximating f, make up the key components of the method, which has two versions, depending on whether we use linear or quadratic interpolation models m_c.

In section 2 we give a general description of the algorithm. In section 3 we consider the case when the model is linear, and in section 4 we discuss quadratic models. In section 5 we report the results of numerical tests comparing the new method with a finite-difference quasi-Newton method and with two model-based methods, DFO [3] and COBYLA [11].

Notation. Throughout the paper $\|\cdot\|$ denotes the Euclidean norm, and $\|\cdot\|_{\mathrm{F}}$ the Frobenius norm of a matrix.

2 The Algorithm

In this section we describe a general framework for the wedge trust region method. At the current iterate x_c we define the model

$$m_c(x_c + s) = f(x_c) + g_c^T s + \tfrac{1}{2} s^T G_c s, \tag{2.1}$$

where the vector $g_c \in I\!R^n$ and the $n \times n$ symmetric matrix G_c must be determined so that the model interpolates f at a set of sample points. (Of course, for linear models we define $G_c \equiv 0$.) The model (2.1) is minimized with respect to $s \in I\!R^n$, subject to a constraint of the form $||s|| \leq \Delta_c$, to generate a step $s_c \in I\!R^n$ that leads to the trial point $x_+ = x_c + s_c$. If x_+ reduces the objective function, it is accepted as the new iterate, and the trust region radius Δ may be increased; otherwise Δ is decreased and a new trial step is computed.

To define the model m_c uniquely we maintain, in addition to x_c, a set of m points

$$\Sigma_c = \{y^1, \ldots, y^m\},$$

which we call *satellites* of x_c. It is easy to see (see sections 3 and 4) that for a linear model we must define $m = n$, and for a quadratic model m should be chosen as $m = (n+1)(n+2)/2-1$. We then impose the interpolation conditions

$$m_c(x_c) = f(x_c), \qquad m(y^l) = f(y^l), \quad l = 1, \ldots, m.$$

When the model m_c is uniquely determined by these conditions, we say that the interpolation set $\{x_c\} \cup \Sigma_c$ is *non-degenerate*.

Let us suppose that we start the current iteration with a non-degenerate set of sample points $\{x_c\} \cup \Sigma_c$. Before computing a new trial point using the model m_c, the farthest satellite from the current iterate x_c, say $y^{l_{\rm out}}$, is identified as the point that will be removed from Σ_c. This choice promotes the conservation of points that provide local information of f around x_c. We then define a "taboo region" in $\mathbb{R}^n$ that contains all the points $x_c + s$ that, if included in the interpolation set in place of $y^{l_{\rm out}}$, would result in a degenerate set of sample points. We also define a set $\mathcal{W}_c$ that contains $\mathcal{T}_c$, and that is designed to avoid points that are very near $\mathcal{T}_c$. The description of $\mathcal{T}_c$ and $\mathcal{W}_c$ for the case of linear and quadratic models will be given in sections 3 and 4, respectively, where we also show that appropriate representations of these sets are inexpensive to compute.

Once the "wedge" $\mathcal{W}_c$ has been determined, we compute a trial step s_c by approximately solving

$$\begin{aligned} \min_s \ m_c(x_c + s) &= f(x_c) + g_c^T s + \tfrac{1}{2} s^T G_c s && \text{(2.2a)} \\ \text{subject to} \quad \|s\| &\le \Delta_c, && \text{(2.2b)} \\ s &\notin \mathcal{W}_c, && \text{(2.2c)} \end{aligned}$$

and define $x_+ = x_c + s_c$. If this trial point x_+ reduces f, then x_+ becomes the new iterate, and x_c becomes a satellite point, replacing $y^{l_{\rm out}}$. If, on the other hand, x_+ does not reduce f, the current iterate is not updated, and x_+ may or may not be discarded, depending on how far it is from the current iterate x_c compared with $y^{l_{\rm out}}$. This and other aspects of the algorithm are described below.

Algorithm 1
Choose the trust region parameters α, $\beta \in (0, 1)$, an initial trust region radius $\Delta_c > 0$, and an initial guess x_c. Select an initial set of satellites $\Sigma_c = \{y^1, y^2, \ldots, y^m\}$ such that $x_c \cup \Sigma_c$ is non-degenerate. Here $m = n$ for a linear model, and $m = \frac{1}{2}(n+1)(n+2) - 1$ for a quadratic model. (We assume that $f(x_c) \le f(y) \quad \forall y \in \Sigma_c$.)
Repeat

1. Find a satellite that is farthest from the current iterate (break ties arbitrarily):
   $y^{l_{\rm out}} = \arg\max_{y \in \Sigma_c} \|y - x_c\|$.

2. Form a model $m_c(x_c + s)$ that interpolates $\{x_c\} \cup \Sigma_c$, and define the wedge $\mathcal{W}_c$.

3. Compute s_c by approximately solving subproblem (2.2), and evaluate $f(x_c + s_c)$.

4. Set $\text{ared}(s_c) \equiv f(x_c) - f(x_c + s_c)$ and $\text{pred}(s_c) \equiv m_c(x_c) - m_c(x_c + s_c)$.

5. Update Δ_c:
   If $\text{ared}(s_c) > \alpha\, \text{pred}(s_c)$, choose Δ_+ such that $\Delta_+ \geq \Delta_c$,
   else set $\Delta_+ = \beta\Delta_c$.

6. **If** $f(x_c + s_c) < f(x_c)$ (successful iteration)
   Update the current iterate, and include x_c in the satellite set, discarding $y^{l_{\text{out}}}$:
   a. $x_+ = x_c + s_c$
   b. $\Sigma_+ = \{x_c\} \cup \Sigma_c \setminus \{y^{l_{\text{out}}}\}$.
   else (unsuccessful iteration)
   If the new trial point is not further to the current iterate than $y^{l_{\text{out}}}$,
   admit it to the satellite set, discarding $y^{l_{\text{out}}}$; otherwise, discard the new trial point:
   c. $x_+ = x_c$
   d. $\Sigma_+ = \begin{cases} \{x_c + s_c\} \cup \Sigma_c \cup \setminus \{y^{l_{\text{out}}}\} & \text{if } \|y^{l_{\text{out}}} - x_c\| \geq \|(x_c + s_c) - x_c\| \\ \Sigma_c & \text{otherwise.} \end{cases}$

7. $x_c \leftarrow x_+, \;\; \Sigma_c \leftarrow \Sigma_+, \;\; \Delta_c \leftarrow \Delta_+$.

End

In the next sections we discuss the definition of the model m_c and of the wedge $\mathcal{W}_c$, and the procedure for approximately solving the trust region subproblem.

Algorithm 1 is conceptually simple since it generates only one type of step, namely a minimization step that always attempts to decrease f. Its novelty lies in the use of the wedge constraint, and in in the acceptance strategy in step 6. This strategy ensures that the model is sufficiently accurate when needed, and therefore contributes significantly to the robustness of the iteration, as we now discuss.

Trust region methods for gradient-based optimization guarantee that a successful step will be generated whenever the trust region is small enough (and assuming that x_c is not a stationary point of f). In order to retain this important property in interpolation-based models, we deviate from the standard practice of discarding trial points that give rise to an increase the objective function, if these points help to improve the accuracy of the model in a vicinity of x_c. More specifically, suppose that the model m_c is poor and that, as a result, a sequence of unsuccessful trial points are computed. If these trial points were discarded, then the interpolation model would not change, and subsequent steps may still be poor in spite of the fact that the trust region has been reduced.

To ensure that the quality of model improves as steps are being rejected, we propose the mechanism described in Step 6 of the algorithm. Since an unsuccessful trial point $x_c + s_c$ will be retained as a satellite if it is no further from x_c than $y^{l_{\text{out}}}$ (Step 6d), we promote the generation of trial points in the vicinity of x_c, and also avoid wasting expensive

function evaluations. If a sequence of consecutive unsuccessful trial steps is generated and Δ decreases sufficiently, the trial points will eventually be admitted as satellites of x_c. As a result, the interpolation model m_c will become increasingly accurate, so that eventually a successful step will be computed. This endows Algorithm 1 with satisfactory global convergence properties [8] (see also [7] for a convergence analysis when linear models are used).

The values of the trust region parameters α and β, and the updating rule of the trust region radius in Step 5 used in our numerical tests are given in section 5.

3 Linear Models

Linear models can be useful in derivative-free optimization because they only require $n+1$ sample points—a useful feature when the number of variables is not very small. The model, at the current iterate x_c takes the form

$$m_c(x_c + s) = f(x_c) + g_c^T s, \tag{3.1}$$

where g_c is a vector in $\mathbb{R}^n$ to be determined. Since g_c has n components, we maintain, in addition to x_c, the set of n satellites

$$\Sigma_c = \{y^1, \ldots, y^n\},$$

and impose the interpolation conditions $m(y^l) = f(y^l), \;\; l = 1, ..., n$, which can be written as

$$g_c^T s^l = f(y^l) - f(x_c) \qquad l = 1, \ldots, n. \tag{3.2}$$

Here s^l is the displacement from x_c to y^l; i.e.

$$y^l = x_c + s^l \qquad l = 1, \ldots, n.$$

It follows from (3.2), that the linear model (3.1) is uniquely determined if and only if the set of sample points $\{x_c\} \cup \Sigma_c$ is such that the set

$$\{s^l : l = 1, \ldots, n\}$$

is linearly independent.

To compute a new iterate, we first select $y^{l_{\text{out}}}$, the satellite that is farthest from x_c. The taboo region $\mathcal{T}_c$, which is the region that we want to avoid when computing a new point so that the new sample set is non-degenerate, is therefore defined as the $(n-1)$-dimensional subspace spanned by the displacement vectors

$$\{s^l : l = 1, \ldots, n, \quad l \neq l_{\text{out}}\} \tag{3.3}$$

corresponding to the satellites that will remain in the sample set; see Figure 1. A more convenient representation is

$$\mathcal{T}_c = \{s \in \mathbb{R}^n : b_c^T s = 0\},$$

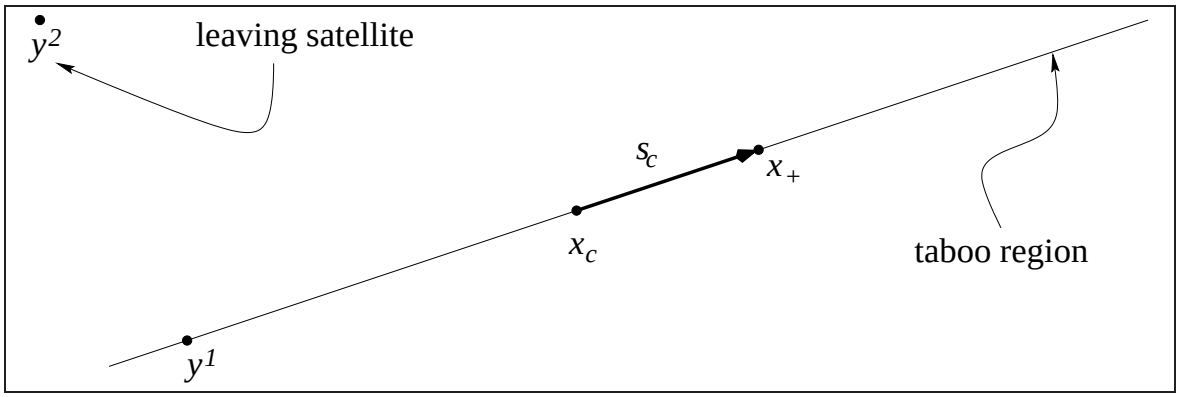

Figure 1: Example for $n = 2$. The set of sample points at the current iteration is $\{x_c\} \cup \Sigma_c = \{x_c, y^1, y^2\}$. Since y^2 is the farthest point from x_c, it will be removed from the sample set. If the trial step s_c lies on the taboo region the new set of sample points, $\{x_+\} \cup \Sigma_+ = \{x_+, x_c, y^1\}$, will be degenerate.

where $b_c \in \mathbb{R}^n$ is normal to the displacement vectors (3.3).

As mentioned in section 2, we would also like to avoid steps s_c that are very close to the taboo region, so that the system (3.2) is not too ill-conditioned, and to ensure that the sample points are reasonably spaced out. To achieve this, we demand that the magnitude of the cosine of the angle between the step s_c and the normal b_c is not less than a given constant $\gamma \in (0, 1)$, i.e.,

$$|b_c^T s| \geq \gamma \, \|b_c\| \|s\|. \tag{3.4}$$

We call inequality (3.4) the *wedge constraint*, and the parameter γ determines the "width" of the wedge; see Figure 2. We compute a trial step s_c by solving

$$\begin{aligned} \min_s \; m_c(x_c + s) &= f(x_c) + g_c^T s && \text{(3.5a)} \\ \text{subject to} \quad ||s|| &\leq \Delta_c && \text{(3.5b)} \\ |b_c^T s| &\geq \gamma \, ||b_c|| \, ||s||. && \text{(3.5c)} \end{aligned}$$

This problem can be easily solved. If we ignore the wedge constraint (3.5c), the solution is $s_{\text{TR}} = -(\Delta_c / \|g_c\|) g_c$. If s_{TR} satisfies (3.5c) (i.e., if it lies "outside" the wedge), then $s_c = s_{\text{TR}}$ is the solution of the subproblem (3.5). Otherwise, the wedge constraint is active, and it is easy to verify that an optimal solution lies in the span of g_c and b_c. By rotating s_{TR} in the plane $\text{span}\{s_{\text{TR}}, b_c\}$ we find the two points on this plane at which the wedge constraint is satisfied as an equality, and we chose the one with lowest model objective. This provides a global solution to subproblem (3.5), which is unique—except in the case when $b_c^T g_c = 0$, when there are exactly two global solutions.

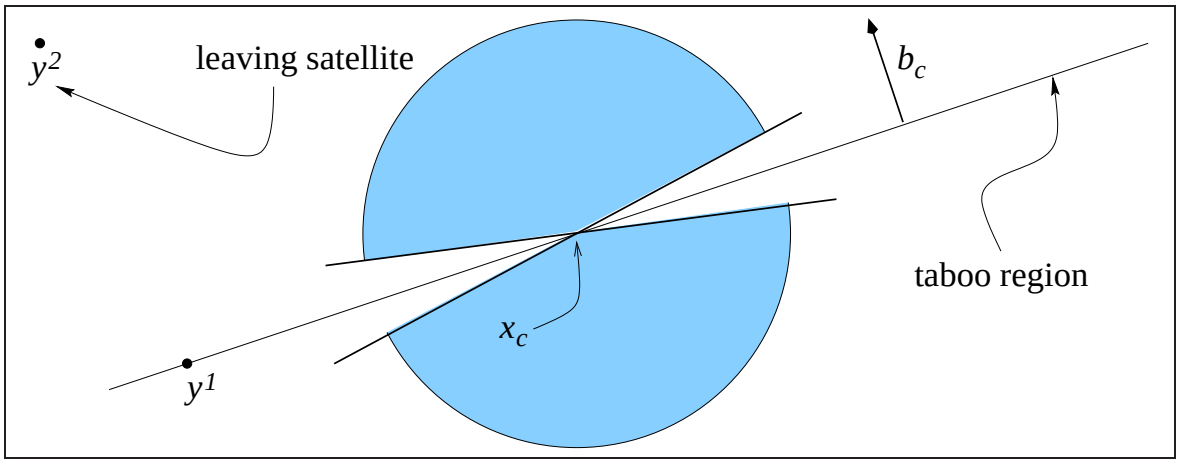

Figure 2: Example for $n = 2$. Steps s that fall in the shaded area will satisfy (3.4), and will not belong to the taboo region $\mathcal{T}_c$. The circle represents the trust region.

4 Quadratic Models

In order to uniquely define a quadratic model

$$m_c(x_c + s) = f(x_c) + g_c^T s + \tfrac{1}{2} s^T G_c s \tag{4.1}$$

that interpolates a set of points, we need to determine the coefficients $g_c \in I\!R^n$, and the symmetric $n \times n$ matrix G_c, a total of $m = \frac{1}{2}(n+1)(n+2) - 1$ scalar unknowns. Thus, in addition to the current approximation to the minimizer x_c, we will maintain m satellites

$$y^l = x_c + s^l \qquad l = 1, \ldots, m.$$

The quadratic model (4.1) can be expressed as (we drop the subscript c from the elements of G_c to keep the notation simple)

$$\begin{aligned} m_c(x_c + s) &= f(x_c) + g_c^T s + \sum_{i<j} G_{ij} s_i s_j + \tfrac{1}{2} \textstyle\sum_i G_{ii} s_i^2 \\ &\equiv f(x_c) + \widehat{g}_c^T \widehat{s}, \end{aligned} \tag{4.2}$$

where we have collected the elements of g_c and G_c in the m-vector of unknowns

$$\widehat{g}_c \equiv \left(g_c^T, \{G_{ij}\}_{i<j}, \left\{ \tfrac{1}{\sqrt{2}} G_{ii} \right\} \right)^T, \tag{4.3}$$

and defined the m-vector

$$\widehat{s} \equiv \left(s^T, \{s_i s_j\}_{i<j}, \left\{ \tfrac{1}{\sqrt{2}} s_i^2 \right\} \right)^T.$$

Since the model (4.2) has the same form as (3.1), the determination of the vector of unknown coefficients $\widehat{g}$ will be done as in the linear case. We deduce, from (4.2), that the interpolation conditions take the form

$$(\hat{s}^l)^T \hat{g}_c = f(y^l) - f(x_c) \qquad l = 1, \ldots, m, \tag{4.4}$$

where

$$\hat{s}^l \equiv \left((s^l)^T, \{s_i^l s_j^l\}_{i<j}, \left\{ \tfrac{1}{\sqrt{2}} (s_i^l)^2 \right\} \right)^T \qquad l = 1, \ldots, m.$$

The model (4.1) will thus be uniquely determined if and only if the system (4.4) has a unique solution, or equivalently, if and only if the set

$$\{\hat{s}^l : \, l = 1, \ldots, m\} \tag{4.5}$$

is linearly independent. This is the condition that the set of sample points must satisfy in order to be non-degenerate. The taboo region is defined as

$$\text{span}\{\hat{s}^l : \, l = 1, \ldots, m; \; l \neq l_{\text{out}}\}, \tag{4.6}$$

and can also be expressed as

$$\mathcal{T}_c = \{\hat{s} \in I\!R^m : \, \hat{b}_c^T \hat{s} = 0\}, \tag{4.7}$$

where $\hat{b}_c \in I\!R^m$ is perpendicular to the subspace (4.6). As in the linear case, we define a region that contains $\mathcal{T}_c$ by demanding that the magnitude of the cosine of the angle between $\hat{s}$ and the normal $\hat{b}_c$ be greater than or equal to a given scalar $\gamma \in (0, 1)$, i.e.,

$$|\hat{b}_c^T \hat{s}| \geq \gamma \|\hat{b}_c\| \|\hat{s}\|. \tag{4.8}$$

All that is left to do is to express this condition and the taboo region (4.7) in terms of the variables $s \in I\!R^n$ of the original problem. Writing $\hat{b}_c$ in the form (4.3) we have

$$\hat{b}_c \equiv \left(b_c^T, \; \{B_{ij}\}_{i<j}, \; \{\tfrac{1}{\sqrt{2}} B_{ii}\} \right)^T \tag{4.9}$$

where $b_c \in I\!R^n$. We now let B_c be the $n \times n$ symmetric matrix with upper triangular elements given by $\{B_{ij}\}_{i \leq j}$. The taboo region (4.7) is thus given by

$$\mathcal{T}_c = \{s \in I\!R^n : \, b_c^T s + \tfrac{1}{2} s^T B_c s = 0\}, \tag{4.10}$$

and is therefore the set of solutions to a quadratic equation. Let us assume without loss of generality that the vector $\hat{b}_c$ is normalized so that $\|\hat{b}_c\| = \sqrt{\|b_c\|^2 + \frac{1}{2}\|B_c\|_{\text{F}}^2} = 1$. Then the wedge condition (4.8) can be written as

$$|b_c^T s + \tfrac{1}{2} s^T B_c s| \geq \gamma \sqrt{\|s\|^2 + \tfrac{1}{2} \|s \, s^T\|_{\text{F}}^2}. \tag{4.11}$$

Since

$$\|s s^T\|_{\text{F}} = \|s\|^2,$$

inequality (4.11) can be written as

$$|b_c^T s + \tfrac{1}{2} s^T B_c s| \geq \gamma \|s\| \sqrt{1 + \tfrac{1}{2} \|s\|^2}.$$

This defines the wedge constraint when the model is quadratic.

To determine the coefficients g_c, G_c, b_c and B_c we compute the QR factorization of the matrix whose columns are the vectors $\{\hat{s}^l\}$ in (4.5). Using this factorization we can then solve for $\hat{g}_c$ the system (4.4), obtaining g_c and G_c through (4.3). The QR factorization also gives us the vector $\hat{b}_c$ in (4.7), and through (4.9), the vector b_c and the matrix B_c.

The step s_c is therefore defined as an approximate solution of

$$\begin{aligned} \min_s \; m_c(x_c + s) &= f(x_c) + g_c^T s + \tfrac{1}{2} s^T G_c s && \text{(4.12a)} \\ \text{subject to} \quad \|s\| &\leq \Delta_c && \text{(4.12b)} \\ |b_c^T s + \tfrac{1}{2} s^T B_c s| &\geq \gamma \|s\| \sqrt{1 + \tfrac{1}{2}\|s\|^2}. && \text{(4.12c)} \end{aligned}$$

The step s_c is thus dependent on the parameter γ, whose choice can have an impact on the efficiency of the wedge algorithm. If γ is too large, the wedge constraint may rule out steps that make significant progress toward the solution. To avoid these inefficiencies, we will include in the algorithm a procedure for decreasing γ, if necessary. The update of γ will be performed while computing an approximate solution to (4.12), as will be described in the next subsection.

4.1 Step Computation

It is difficult to compute an optimal solution of subproblem (4.12) since G_c may be indefinite and the feasible region is usually non-convex; see Figure 3. We will, however, content

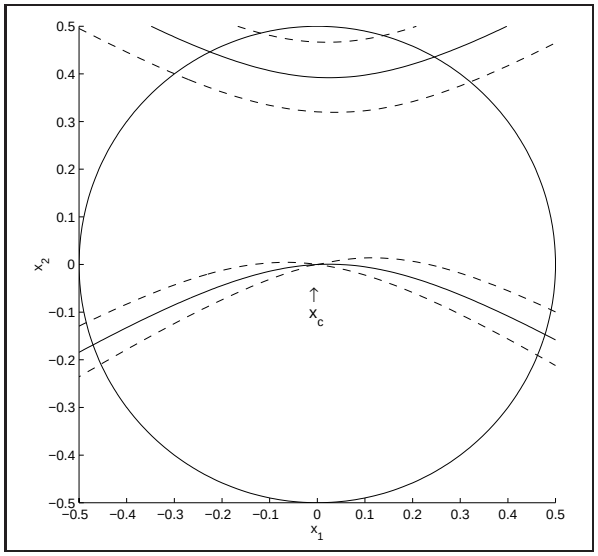

Figure 3: Example for $n = 2$ of the feasible region of subproblem (4.12). The circle centered at x_c represents the trust region. The solid curves depict the taboo region (4.10) (in this example B_c is indefinite), and the boundary of the region defined by the wedge constraint (4.12c) is plotted using dashed lines.

ourselves with finding an approximate solution of (4.12), and we will do so using a procedure that is analogous to that employed in the linear case. While solving the subproblem we will also determine if the value of the wedge parameter γ needs to be decreased.

We first use the technique described by Moré and Sorensen [9] to compute a solution $s_{\rm TR}$ of the trust region problem (4.12a)–(4.12b), ignoring the wedge constraint. If $s_{\rm TR}$ satisfies (4.12c), then the solution of the subproblem (4.12) will be given by $s_c = s_{\rm TR}$. Otherwise, starting from $s_{\rm TR}$, we compute a sequence of trial steps $s^i, i = 1, 2, \ldots$, with $\|s^i\| = \|s_{\rm TR}\|$, that move away from the taboo region in a direction along which the violation of the wedge constraint (4.12c) initially decreases. We generate trial steps until the wedge constraint is satisfied or until the value of the quadratic model (4.12a) has increased too much with respect to its value at $s_{\rm TR}$, in which case γ is decreased.

More precisely, let us define the left hand side in (4.12c) by $\phi(s)$, i.e.,

$$\phi(s) = |b_c^T s + \tfrac{1}{2} s^T B_c s|.$$

Its gradient is given by

$$\nabla\phi(s) = \mathrm{sign}(b_c^T s + \tfrac{1}{2} s^T B_c s)(B_c s + b_c),$$

provided $\phi(s) \neq 0$. We compute the trial steps $s^i,\ i = 1, 2, \ldots$, by incrementally rotating $s_{\rm TR}$ on the plane $\mathrm{span}\{s_{\rm TR}, B_c s_{\rm TR} + b_c\}$, and in the direction of $\nabla\phi(s_{\rm TR})$. Note that $B_c s_{\rm TR} + b_c$ is the normal to the taboo region at $s_{\rm TR}$, and by moving along this normal, we move away from the taboo region. (The gradient of ϕ indicates whether we should move along the normal or the negative of it.)

By means of this rotation, the violation of the wedge constraint initially decreases, as ϕ initially decreases and the right hand side of (4.12c) remains constant due to the equality $\|s^i\| = \|s_{\rm TR}\|$ for all i.

As we generate the trial steps s^i, we monitor the value of the quadratic model (4.12a); we continue the rotation until a trial step s^i satisfies (4.12c) or until the value of m_c has increased too much, in the sense that

$$\mathrm{pred}(s^i) \geq \tfrac{1}{2}\,\mathrm{pred}(s_{\rm TR}) \tag{4.13}$$

where $\mathrm{pred}(s) = m_c(x_c) - m_c(x_c + s)$, with $s \in I\!R^n$. We take this as an indication that the wedge parameter γ is too large. We then "close the wedge," i.e., reduce γ, so that the most current trial step s^i satisfies (4.12c) as an equality,

$$\gamma = |b_c^T s_c + \tfrac{1}{2} s_c^T B_c s_c| \,/\, \left(\|s_c\| \sqrt{1 + \tfrac{1}{2}\|s_c\|^2} \right). \tag{4.14}$$

This new choice of γ will be passed onto the next iteration, so that the values of γ form a non-increasing sequence.

This step computation procedure is described below. Here $s(\theta)$ denotes a rotation of $s_{\rm TR}$, by an angle θ, in the $\mathrm{span}\{s_{\rm TR}, B_c s_{\rm TR} + b_c\}$ and in the direction of $\nabla\phi(s_{\rm TR})$.

Procedure QuadStep
The input parameters are $g_c, G_c, b_c, B_c, \gamma,\ \Delta_c$ and $\delta\theta$.

Solve the trust region subproblem (4.12a)–(4.12b) to obtain $s_{\rm TR}$.
If $s_{\rm TR}$ satisfies (4.12c), return $s_c = s_{\rm TR}$, and γ; **Stop**
Else
  Set $\theta = \delta\theta$
  Repeat until $s(\theta)$ satisfies (4.12c) or $\mathrm{pred}(s(\theta)) \geq \frac{1}{2}\,\mathrm{pred}(s_{\rm TR})$
    $\theta \leftarrow \theta + \delta\theta$
  End Repeat
  Set $s_c = s(\theta)$
  If s_c violates (4.12c), define γ by (4.14)
End if
Return s_c and γ; **Stop**

5 Numerical Results

To assess the robustness and efficiency of the wedge algorithm, we will compare it with three other methods for derivative-free optimization on a selection of problems from the CUTE collection [1]. In this section we denote the kth iterate by x_k; the subscript k will also be used in all the quantities associated with x_k ($f_k = f(x_k)$, Δ_k, etc.)

All the experiments were performed on a Sun Ultra 5 with 384 MB of memory. Double precision IEEE arithmetic was used, except for COBYLA, which is written in single precision. The wedge algorithm was implemented in Matlab.

The trust region parameters in Step 5 of Algorithm 1 were set to $\alpha = 0$ and $\beta = 1/2$, and the trust region update strategy was as follows:

if $\mathrm{ared}(s_k) \leq 0$
  $\Delta_{k+1} = \frac{1}{2}\|s_k\|$
else
  if $\|s_k\| = \Delta_k$, $\Delta_{k+1} = 2\Delta_k$
  else, $\Delta_{k+1} = \Delta_k$
end if

Other strategies are possible, but the one described here appears to work well in practice because it permits the trust region radius Δ to increase fast, allowing larger steps.

We first tested the linear version of the wedge algorithm (WEDlin) and COBYLA [11], a trust region method that uses linear interpolation models. The starting point x_0 in this, and all the results reported below, was supplied by CUTE. The $n+1$ initial satellites required by the first iteration of WEDlin were defined as

$$y^i = x_0 \pm \Delta_0\, e_i, \qquad i = 1, \ldots, n, \tag{5.1}$$

where the initial trust region radius was set to $\Delta_0 = 1$, e_i denotes the ith canonical vector, and the sign in (5.1) was chosen randomly. To account for the randomness introduced in the selection of the initial sample points in both the linear and quadratic versions of the wedge algorithm, each test problem was run five times. The median results, in terms of function evaluations, is reported.

The wedge parameter in WEDlin was chosen as $\gamma = 0.4$, and was kept constant throughout the iteration (only the algorithm that uses quadratic models reduces the value of γ). The parameter `rhoend` (size of the simplex at termination) in COBYLA was set to 10×macheps, where macheps denotes double precision unit roundoff. Similarly, WEDlin stopped if

$$\Delta_k \leq 10 \times \text{macheps}, \tag{5.2}$$

which is taken as an indication that no further progress can be made.

The stopping tests for WEDlin and COBYLA were as follows. We first solved each problem using the NITRO software package [2], which for unconstrained problems amounts to a Newton method using exact second derivatives. NITRO was stopped when $\|\nabla f(x_k)\| < 10^{-6}$ and its final function value f_* was recorded. WEDlin and COBYLA were stopped when

$$\frac{f_k - f_*}{f_0 - f_*} \leq \epsilon, \qquad \text{or equivalently,} \qquad f_0 - f_k \geq (1-\epsilon)(f_0 - f_*), \tag{5.3}$$

where $\epsilon = 10^{-6}$ and f_0 is the objective value at the initial point. Therefore we require that the decrease $f_0 - f_k$ obtained by the algorithms is at least $1-\epsilon$ times the decrease obtained by NITRO. A stopping test of the form (5.3) is also used in [6]. A limit of 8000 function evaluations was imposed in all runs. The results are given in Table 1, where we report the number of function evaluations and the final value of the objective function obtained by each algorithm. We also report (% wed act) the percentage of iterations in WEDlin where the wedge constraint was active.

Though COBYLA was mainly designed for constrained optimization, these results suggest that WEDlin is competitive with COBYLA on unconstrained problems. It is interesting to note also that the wedge constraint is active in a significant fraction of the iterations, showing that this constraint does play an important role in the method.

We now comment on the abnormal terminations of COBYLA. **(1)** As mentioned before, COBYLA is written in single precision. In problem AKIVA, the single precision version of CUTE gives an initial function value of 2.70488, which is lower than the optimal objective value obtained by NITRO, $f_* = 6.16604$. Therefore COBYLA satisfies (5.3), and hence terminates, at the starting point. However, when we run WEDlin in double precision, we obtain the initial function value of 14.5561. **(2)** COBYLA crashes when it tries to evaluate f at a point where f is not defined. (**3**) COBYLA stopped because the size of the simplex (`rhoend`) is less than or equal to 10×macheps; this is a built-in stopping test.

Next we compare the quadratic version of the wedge algorithm (WEDquad) with the quasi-Newton code L-BFGS-B [15] using finite differences to approximate the gradient (QNfd), and all its default settings.

The initial value of the wedge parameter in WEDquad was chosen as $\gamma = 0.4$, and as explained in section 4, it is allowed to change over the iterations. The value $\delta\theta = \pi/600$ was used in procedure QuadStep. As suggested in [12], the $\frac{1}{2}(n+1)(n+2)$ sample points required to define the initial quadratic interpolation model can be chosen as the vertices and the mid-points of a simplex. This was done in WEDquad, using the simplex defined by x_0 and the points (5.1).

		# of evaluations		final f		% wed
	n	COB	WEDlin	COB	WEDlin	act
AKIVA	2	(1)	3184		6.1660e+00	25
BEALE	2	427	286	1.4070e-05	1.2190e-05	25
BIGGS6	6	244	347	2.4254e-01	2.4257e-01	33
BOX3	3	375	2276	7.2034e-06	1.8831e-06	33
BRKMCC	2	148	115	1.6905e-01	1.6904e-01	20
BROWNAL	10	11	629	0.0000e+00	2.6477e-04	38
BROWNDEN	4	392	310	8.5830e+04	8.5830e+04	31
BRYBND	10	8000	8000	8.1488e-03	3.2411e-02	31
CLIFF	2	4	3	1.0008e+00	1.0009e+00	0
CRAGGLVY	10	1085	1065	1.8899e+00	1.8898e+00	32
CUBE	2	8000	6231	2.9539e-03	7.4876e-04	25
DENSCHNA	2	30	28	5.9490e-06	2.1462e-06	19
DIXMAANK	15	1244	907	1.0004e+00	1.0003e+00	29
DQDRTIC	10	3272	2704	1.4387e-02	1.4435e-02	28
EDENSCH	6	102	96	3.9292e+01	3.9303e+01	22
EIGENALS	6	993	7	9.9986e-07	0.0000e+00	0
ENGVAL1	2	21	28	3.4571e-06	1.2317e-05	8
ENGVAL2	3	8000	8000	8.5447e-03	2.5925e-03	34
FMINSURF	16	439	506	1.0000e+00	1.0000e+00	31
FREUROTH	2	6470	4106	4.9040e+01	4.8984e+01	25
GROWTHLS	3	5089(3)	8000	1.2695e+01	1.2425e+01	36
GULF	3	8000	8000	6.0838e+00	5.5864e+00	34
HAIRY	2	3195	98	2.0000e+01	2.0000e+01	25
HATFLDE	3	8000	8000	3.5948e-04	1.0446e-04	38
HEART6LS	6	8000	8000	4.3838e+00	2.8673e+00	33
JENSMP	2	2425	918	1.2437e+02	1.2436e+02	25
KOWOSB	4	3756(3)	8000	3.6637e-04	3.1584e-04	36
MANCINO	10	140	131	1.2200e-01	9.5097e-02	24
MOREBV	10	8000	8000	2.1123e-06	1.7340e-06	30
OSBORNEA	5	(2)	8000		1.5742e-03	38
PFIT1LS	3	8000	8000	3.3536e-02	3.8083e-03	34
POWER	10	88	141	2.9705e-03	2.8584e-03	26
ROSENBR	2	8000	5049	8.1733e-04	2.4151e-05	25
SCHMVETT	3	245	68	-3.0000e+00	-3.0000e+00	18
SISSER	2	14	11	1.1430e-06	1.7838e-06	11
SNAIL	2	8000	1915	7.8217e+00	1.0883e-05	25
VARDIM	10	16	103	8.0418e-01	2.1317e+00	37
WATSON	12	8000	8000	8.0928e-04	1.6597e-04	37
WOODS	4	175	130	1.0903e-02	1.0758e-02	22

Table 1: COBYLA vs WEDGElin: Number of function evaluations, final objective function value and percentage of iterations in which the wedge constraint was active in WEDquad. The boxes indicate that an algorithm required at least 50 fewer function evaluations than the other; if both algorithms reached the limit of 8000 function evaluations, the boxes indicate the algorithm that obtained a lower function value.

The results are reported in Table 2. In addition to the number of function evaluations, the final function value and the percentage of iterations (% wed act) in which the wedge constraint was active, we also report the final value of the wedge parameter γ. As mentioned earlier, WEDquad was run five times and the median of the results are reported. The stopping test was (5.3).

The results show that WEDquad is sometimes, but not always, more efficient than the finite-difference quasi-Newton algorithm. However, WEDquad appears to be more reliable, with one failure, compared to five failures of QNfd.

In problem AKIVA, QNdf returned the function value $-\infty$ at the second iteration and terminated. Most of the failures of QNfd are attributed to lack of progress in the line search due to errors in the finite-difference approximations to the gradients (the code uses forward differences).

Finally, we compare WEDquad with the code DFO [3] that implements a trust region method using quadratic interpolation models. At the time of writing, DFO was not available to the public, and the comparisons reported below are based on the results reported in [5], plus results on additional problems supplied by Katya Scheinberg [13]. For this reason, WEDquad was stopped when

$$f_k \leq f_{\text{DFO}}, \tag{5.4}$$

where f_{DFO} denotes the final objective values obtained by DFO. Both algorithms were run five times for each problem and the median of the results is reported. The initial satellites and all other parameters of WEDquad were chosen as in the previous experiment. The results are given in Table 3.

It is difficult to design an adequate stopping test for comparing derivative free methods, and this last comparison is an example of that. The code DFO may require a significant number of evaluations to recognize convergence to its best value of f [3, 13] (which occurs by default when $\Delta_k < 10^{-4}$). As a result, using the stopping test (5.4) may be adverse to DFO. On the other hand, WEDquad does not employ any special termination mechanism. Its default stop test is (5.2), and in this respect it is completely analogous to gradient-based trust region methods.

Note from Tables 2–3 that, on most problems, the wedge parameter γ is reduced significantly from its initial value of 0.4, and that at the same time, the wedge constraint is active fairly often. Therefore the algorithm appears to have achieved a good balance between the geometric condition requirement and the desire to allow full trust region steps when possible.

We conclude by noting that choosing the initial satellites in WEDquad as the vertices and midpoints of a simplex is not an efficient strategy since it requires $O(n^2)$ function evaluations to start the algorithm. It would be more efficient to use, for example, linear (or underdetermined quadratic) models during the early iterations, but we have not yet experimented with this option. Other refinements that are likely to improve performance include the option of increasing the wedge parameter in certain iterations. This can be done either directly (e.g., when s_{TR} in procedure QuadStep satisfies the wedge constraint (4.12c)); or indirectly, by relaxing the (quite demanding) requirement that the approximate solution

		# of evaluations		final f		% wed	final
	n	QNfd	WEDquad	QNfd	WEDquad	act	γ
AKIVA	2	5(*)	38	1.45561e+01	6.1660e+00	39	1.7e-06
BEALE	2	44	31	6.6203e-06	5.7477e-07	27	1.1e-03
BIGGS6	6	100	63	2.2235e-01	2.2763e-01	11	1.3e-03
BOX3	3	42	49	6.7426e-07	2.9466e-07	28	7.3e-05
BRKMCC	2	20	11	1.6904e-01	1.6904e-01	67	5.1e-04
BROWNAL	10	90	143	7.5175e-05	2.2297e-04	10	2.7e-04
BROWNDEN	4	122	70	8.5827e+04	8.5827e+04	5	5.2e-04
BRYBND	10	442	537	1.4670e-04	9.2160e-05	4	2.1e-06
CLIFF	2	53	6	4.4956e+02	1.0009e+00	0	0.4e+00
CRAGGLVY	10	387	530	1.8896e+00	1.8897e+00	3	1.5e-04
CUBE	2	119	128	1.9725e-04	7.3466e-04	13	5.1e-08
DENSCHNA	2	29	22	6.7846e-08	3.4250e-06	12	8.3e-04
DIXMAANK	15	338	682	1.0001e+00	1.0003e+00	3	9.9e-05
DQDRTIC	10	145	71	6.8361e-04	1.1992e-18	50	2.0e-03
EDENSCH	6	114	153	3.9291e+01	3.9303e+01	5	1.7e-03
EIGENALS	6	86	28	1.9489e-07	0.0000e+00	0	0.4e+00
ENGVAL1	2	38	31	1.5099e-06	3.7017e-06	15	2.3e-03
ENGVAL2	3	118	67	5.0904e-04	4.0162e-04	27	6.1e-07
FMINSURF	16	376	764	1.0000e+00	1.0000e+00	2	1.4e-04
FREUROTH	2	65	28	4.8984e+01	4.8984e+01	43	1.3e-04
GROWTHLS	3	45(*)	1445	3.5421e+03	1.0895e+00	4	2.9e-09
GULF	3	274	234	1.2715e-06	1.1460e-05	8	4.1e-06
HAIRY	2	146	40	2.0000e+01	2.0000e+01	11	1.3e-03
HATFLDE	3	58	70	2.0464e-05	3.7258e-05	8	5.8e-05
HEART6LS	6	4558(*)	8000(#)	4.9343e-02	2.2899e-01	1	1.0e-08
JENSMP	2	145(*)	44	2.5958e+02	1.2436e+02	23	1.3e-04
KOWOSB	4	167	101	3.0780e-04	3.0780e-04	11	4.1e-06
MANCINO	10	35	133	6.8355e-03	2.6684e-02	16	3.3e-05
MOREBV	10	585	115	6.2161e-09	1.3512e-08	22	5.2e-07
OSBORNEA	5	13(*)	1128	8.7903e-01	5.5500e-05	10	1.2e-10
PFIT1LS	3	210	93	4.5981e-04	7.9554e-04	14	4.8e-06
POWER	10	178	312	2.0331e-03	2.0220e-03	3	4.1e-05
ROSENBR	2	140	86	1.2816e-06	1.9314e-05	7	6.8e-05
SCHMVETT	3	46	35	-3.0000e+00	-3.0000e+00	31	3.8e-04
SISSER	2	29	35	1.8357e-06	2.2536e-06	20	4.9e-04
SNAIL	2	29	297	4.0694e-06	1.6856e-06	5	2.0e-04
VARDIM	10	200	692	1.8900e+00	2.1709e+00	1	5.3e-05
WATSON	12	873	460	2.9748e-05	2.9340e-05	3	7.8e-06
WOODS	4	97	114	3.3756e-03	1.0954e-02	13	1.4e-04

Table 2: QNfiniteDiff vs WEDGEquad: Number of function evaluations, final objective function value, percentage of iterations in which the wedge constraint was active in WEDquad, and final value of the wedge parameter γ. The boxes indicate that an algorithm required at least 50 fewer function evaluations than the other. A (*) indicates that the quasi-Newton code could not make further progress and terminated. The symbol # indicates that WEDquad reached the maximum number of function evaluations allowed.

		# of evaluations		final f		% wed	final
	n	DFO	WEDquad	DFO	WEDquad	act	γ
AKIVA	2	68	38	6.1660e+00	6.1660e+00	39	1.7e-06
BEALE	2	55	35	1.1011e-08	1.3360e-10	36	4.0e-05
BIGGS6	6	1364	362	1.7195e-05	1.5875e-05	6	9.9e-05
BOX3	3	81	51	2.2859e-07	1.1230e-07	36	7.3e-05
BRKMCC	2	24	11	1.6904e-01	1.6904e-01	67	5.1e-04
BROWNAL	10	837	234	9.2867e-07	8.7100e-07	9	1.0e-06
BROWNDEN	4	110	71	8.5822e+04	8.5822e+04	5	1.1e-03
BRYBND	10	528	618	9.9818e-08	8.0361e-08	6	4.1e-09
CLIFF	2	75	57	1.9979e-01	1.9979e-01	35	4.7e-07
CRAGGLVY	10	1026	553	1.8866e+00	1.8866e+00	3	5.7e-05
CUBE	2	107	138	2.6504e-07	2.2024e-07	13	5.1e-08
DENSCHNA	2	24	24	3.7212e-08	1.5670e-09	23	1.4e-04
DIXMAANK	15	1118	691	1.0000e+00	1.0000e+00	3	6.4e-05
DQDRTIC	10	403	75	1.6263e-20	3.1892e-21	60	5.3e-04
EDENSCH	6	177	167	3.9287e+01	3.9287e+01	6	1.6e-03
EIGENALS	6	211	28	8.9164e-07	0.0000e+00	0	0.4e+00
ENGVAL1	2	29	34	1.0550e-07	8.4608e-09	29	2.3e-05
ENGVAL2	3	149	80	2.8479e-07	7.3229e-08	20	4.1e-06
FMINSURF	16	1210	764	1.0000e+00	1.0000e+00	2	1.4e-04
FREUROTH	2	75	28	4.8984e+01	4.8984e+01	43	1.3e-04
GROWTHLS	3	243	123	1.2396e+01	1.2394e+01	23	8.5e-09
GULF	3	411	167	1.4075e-03	1.2626e-03	10	2.4e-05
HAIRY	2	51	40	2.0000e+01	2.0000e+01	11	1.3e-03
HATFLDE	3	95	85	3.8660e-06	2.9474e-06	17	4.4e-06
HEART6LS	6	1350	912	4.3167e-01	4.3161e-01	4	2.5e-07
JENSMP	2	97	44	1.2436e+02	1.2436e+02	23	1.3e-04
KOWOSB	4	117	62	3.6359e-04	3.5956e-04	11	1.7e-04
MANCINO	10	276	140	1.5268e-07	6.6249e-08	12	2.2e-05
MOREBV	10	476	104	6.0560e-07	6.5589e-08	21	1.2e-05
OSBORNEA	5	329	137	9.3144e-03	3.1947e-03	8	1.1e-05
PFIT1LS	3	180	94	4.2637e-04	4.0025e-04	14	4.8e-06
POWER	10	206	500	2.6064e-06	2.3942e-06	2	4.1e-05
ROSENBR	2	81	88	1.9716e-07	1.2083e-08	9	1.6e-05
SCHMVETT	3	53	35	-3.0000e+00	-3.0000e+00	31	3.8e-04
SISSER	2	27	38	1.2473e-06	9.2610e-07	18	4.9e-04
SNAIL	2	246	300	1.2661e-08	1.6242e-09	5	2.0e-04
VARDIM	10	2061	1156	2.6730e-07	2.0213e-08	1	2.4e-06
WATSON	12	1919	441	4.3239e-05	4.3197e-05	3	7.8e-06
WOODS	4	122	143	3.3079e-07	3.0107e-07	15	7.3e-04

Table 3: DFO vs WEDGEquad: Number of function evaluations, final objective function value, percentage of iterations in which the wedge constraint was active in WEDquad, and final value of the wedge parameter γ. The boxes indicate that an algorithm required at least 50 fewer function evaluations than the other.

of the quadratic subproblem satisfy the test (4.13). An alternative test could be $\text{pred}(s^i) \geq \frac{1}{2}\,\text{pred}(s_{\text{CA}})$, where s_{CA} is the *Cauchy point* [10] for subproblem (4.12a)–(4.12b).

Acknowledgements. We are grateful to Katya Scheinberg for some useful observations on testing derivative-free methods and for providing results with DFO, and to Mike Powell for supplying the code COBYLA.

References

[1] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[2] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[3] A. R. Conn, K. Scheinberg, and Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In M. D. Buhmann and A. Iserles, editors, *Approximation theory and optimization*, pages 83–108. Cambridge University Press, Cambridge, 1997.

[4] A. R. Conn, K. Scheinberg, and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79:397–414, 1997.

[5] A. R. Conn, K. Scheinberg, and Ph. L. Toint. A derivative free optimization algorithm in practice. Technical Report 98/11, July 1998. Accepted for the American Institute of Aeronautics and Astronautics Conference, St Louis, September 1998.

[6] C. Elster and A. Neumaier. A grid algorithm for bound constrained optimization of noisy functions. *IMA Journal of Numerical Analysis*, pages 585–608, 1995.

[7] M. Marazzi. *Nonlinear Optimization With and Without Derivatives*. PhD thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, 2145 Sheridan Road, Evanston, Illinois 60208-3119, USA. Expected to appear in June 2001.

[8] M. Marazzi and J. Nocedal. Global convergence of wedge trust region methods for derivative free optimization. In preparation.

[9] J. Moré and D.C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4:553–572, 1983.

[10] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.

[11] 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*, pages 51–67, Printed in the Netherlands, 1994. Kluwer Academic Publishers.

[12] M. J. D. Powell. Direct search algorithms for optimization calculations. In *Acta Numerica*, pages 288–336. Cambridge University Press, Cambridge, 1998.

[13] K. Scheinberg. Private communication.

[14] M. H. Wright. Direct search methods: Once scorned, now respectable. In *Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis)*, pages 191–208. Addison Wesley Longman, 1996.

[15] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. L-BFGS-B, fortran subroutines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.