Optimization Methods and Software
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/goms20

A trust region method based on a new affine scaling technique for simple bounded optimization

Xiao Wang [a] & Ya-Xiang Yuan [a]
[a] State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing , The Academy of Mathematics and Systems Sciences, Chinese Academy of Sciences , PO Box 2719, Beijing , 100190 , P.R. China
Published online: 10 Oct 2011.

To cite this article: Xiao Wang & Ya-Xiang Yuan (2013) A trust region method based on a new affine scaling technique for simple bounded optimization, Optimization Methods and Software, 28:4, 871-888, DOI: 10.1080/10556788.2011.622378

To link to this article: http://dx.doi.org/10.1080/10556788.2011.622378

Optimization Methods & Software, 2013
Vol. 28, No. 4, 871–888, http://dx.doi.org/10.1080/10556788.2011.622378

Taylor & Francis
Taylor & Francis Group

A trust region method based on a new affine scaling technique for simple bounded optimization†

Xiao Wang* and Ya-Xiang Yuan

State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, The Academy of Mathematics and Systems Sciences, Chinese Academy of Sciences, PO Box 2719, Beijing, 100190, P.R. China

(*Received 16 December 2010; final version received 6 September 2011*)

In this paper, we propose a new trust region affine scaling method for nonlinear programming with simple bounds. Our new method is an interior-point trust region method with a new scaling technique. The scaling matrix depends on the distances of the current iterate to the boundaries, the gradient of the objective function and the trust region radius. This scaling technique is different from the existing ones. It is motivated by our analysis of the linear programming case. The trial step is obtained by minimizing the quadratic approximation to the objective function in the scaled trust region. It is proved that our algorithm guarantees that at least one accumulation point of the iterates is a stationary point. Preliminary numerical experience on problems with simple bounds from the CUTEr collection is also reported. The numerical performance reveals that our method is effective and competitive with the famous algorithm LANCELOT. It also indicates that the new scaling technique is very effective and might be a good alternative to that used in the subroutine *fmincon* from Matlab optimization toolbox.

Keywords: bound constrained optimization; trust region; interior point; affine scaling

1. Introduction

In this paper, we study the general bound constrained optimization problems:

$$\min_{x\in\mathbb{R}^n} \quad f(x) \tag{1a}$$

$$\text{s.t.} \quad l \le x \le u, \tag{1b}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a twice continuously differentiable function, $l = (l^{(1)}, \ldots, l^{(n)})^{\mathrm{T}}$ and $u = (u^{(1)}, \ldots, u^{(n)})^{\mathrm{T}}$. Likewise, it is possible to include one-sided constraints by setting $l^{(i)}$ to $-\infty$ or $u^{(i)}$ to ∞, depending on which bound is required. Without loss of generality, we assume that $l^{(i)} < u^{(i)}$ for all i.

Though the bound constrained optimization problem (1) is a very simple constrained optimization problem, it appears often in practice and has been studied extensively. Many

*Corresponding author. Email: wangxiao@lsec.cc.ac.cn
†This paper has been presented in ICOTA8.

numerical methods for (1) have been proposed [2,3,5–8,11,12,16–18,20–25,28–31,33,37]. Some of these papers studied the special case where the objective function $f(x)$ is quadratic, whereas others considered the general nonlinear $f(x)$. Most of these numerical methods use quadratic models to compute the search direction or the trust region trial step, though some also use projected gradients. Either an active set technique or an interior-point technique is used in these algorithms. The algorithm that we propose in this paper is a trust region interior-point method in which we use a new affine scaling technique.

The affine scaling algorithm was first introduced by Dikin [13] for linear programming. The method was later re-discovered independently by Barnes [1] and by Vanderbei *et al*. [35]. A second-order affine scaling algorithm for convex quadratic programming (QP) problems was proposed by Dikin and Zorkaltsev [14] in 1980. In that paper, an ellipsoid that centres at the current point and whose radius is a fixed fraction $\beta \in (0, 1)$ of the largest scaled ellipsoid inscribed in the non-negative orthant was constructed. The next iterate was generated by minimizing the objective function over the intersection of the ellipsoid with the feasible region. Many papers have been published on affine scaling algorithms for QP problems [4,26,33]. Affine scaling algorithms for solving linearly constrained convex programming problems have also been studied [27,32]. All the aforementioned affine scaling methods require that the ellipsoid defined be contained in the feasible region. For general nonlinear programming problems, affine scaling algorithms normally use trust region techniques to ensure convergence. Trust region methods are a large class of numerical methods for nonlinear programming [10,36].

In [7], a famous affine scaling algorithm for solving (1) was proposed. Based on an equivalent scaled Karush-Kuhn-Tucker (KKT) system, the authors minimized a quadratic function subject to an ellipsoidal constraint. In that algorithm, the scaling matrix D_k is dependent on the distance of x_k to the bounds and $\nabla f(x_k)$. However, to maintain feasibility, after obtaining the solution of the subproblem, truncation needs to be done. A similar idea was also adopted by others, such as Bellavia *et al*. [2] and Coleman and Li [6]. Since truncation may lead to short steps, the authors took a projection step instead of performing truncation in [22,23]. When the bound of some variable is very far away from the current iterate point, the scaling technique of Coleman and Li would give a very large diagonal element of D_k. In this case, the ellipsoid $\|D_k^{-1}d\| \leq \Delta_k$ can be very oblong, and consequently the trust region step would likely be very large along one particular coordinate direction, which means that all the other variables would remain nearly unchanged. Different from previous works, in our algorithm, the trust region technique is dedicated to the design of the scaling matrix.

In this paper, we study a trust region method which generates the trial step s_k by minimizing a second-order approximation of $f(x)$ over an ellipsoidal trust region defined by a new affine scaling technique. Motivated by linear programming, we find an interesting relationship among the scaling matrix D_k, the trust region radius Δ_k and the gradient g_k. We minimize a quadratic function in a scaled trust region and require that every iterate be a strictly feasible point. Without any non-degeneracy assumption, we obtain the global convergence of the iterates to stationary points. Lastly, we test the new algorithm on the majority of the bound constrained optimization problems from CUTEr and compare it with LANCELOT and the subroutine *fmincon* in the Matlab optimization toolbox based on the method proposed in [6]. The numerical results show that our new method is very effective.

This paper is organized as follows. In Section 2, we state the algorithm in detail, whose global convergence to first-order critical points is shown in Section 3. Computational results are presented and discussed in Section 4. In Section 5, we give some concluding remarks.

Notation In all the following expressions, the norm symbol without a subscript, $\|\cdot\|$, refers to the 2-norm. Superscript '(i)' refers to the elements of the vector and superscript 'ii' refers to the diagonal elements of the matrix. Subscript 'k' refers to the iteration indices, and f_k is taken as $f(x_k)$, while g_k means $\nabla f(x_k)$.

2. The algorithm

The interior-point technique requires that all iteration points x_k be in the interior of the feasible set. Define the following sets:

$$\mathcal{F} := \{x \in \mathbb{R}^n : l \le x \le u\}, \quad \mathcal{F}^\circ := \{x \in \mathbb{R}^n : l < x < u\}. \tag{2}$$

A trust region affine scaling method for the bound constrained optimization (1) normally solves the subproblem at the current iterate point $x_k \in \mathcal{F}^\circ$ stated by

$$\min_{d \in \mathbb{R}^n} \quad g_k^{\mathrm{T}} d + \frac{1}{2} d^{\mathrm{T}} B_k d \tag{3a}$$

$$\text{s.t.} \quad ||D_k^{-1} d|| \le \Delta_k, \tag{3b}$$

where $\Delta_k > 0$ is the trust region radius and D_k is a positive definite diagonal matrix. The scaling matrix D_k is often chosen so that every point in the ellipsoid $\Omega_k = \{d| \ ||D_k^{-1} d|| \le \Delta_k\}$ gives a feasible step. Namely, $x_k + d$ is a feasible point of the original bound constrained problem (1) for all d in Ω_k. For scaling matrices D_k which do not have this property, either the step computed by (3) may need to be truncated or an additional constraint $l \le x_k + d \le u$ has to be added to (3).

The affine scaling matrix D_k given by Coleman and Li [6,7] is defined by

$$(D_k)_{ii} = \sqrt{|v^{(i)}(x_k)|}, \quad i = 1, \ldots, n, \tag{4}$$

where $v^{(i)}(x)$ is defined by

$$v^{(i)}(x) = \begin{cases} x^{(i)} - u^{(i)} & \text{if } g^{(i)}(x) < 0 \text{ and } u^{(i)} < \infty, \\ x^{(i)} - l^{(i)} & \text{if } g^{(i)}(x) \ge 0 \text{ and } l^{(i)} > -\infty, \\ -1 & \text{if } g^{(i)}(x) < 0 \text{ and } u^{(i)} = \infty, \\ 1 & \text{if } g^{(i)}(x) \ge 0 \text{ and } l^{(i)} = -\infty. \end{cases} \tag{5}$$

The derivation of the scaling matrix (4) is based on Newton's method for the diagonal system

$$D(x)^2 g(x) = 0. \tag{6}$$

We now derive our new scaling technique. Consider the very simple case where we need to solve the following linear programming:

$$\min_{x \in \mathbb{R}^n} \quad c^{\mathrm{T}} x \tag{7a}$$

$$\text{s.t.} \quad x \ge 0, \tag{7b}$$

with $c > 0$. It is easy to see that the optimal solution is $x_* = 0$, and all the constraints are active. Hence, for any current point x_k, it is desirable to have $s_k = -x_k$. Now, suppose we have a trust

region subproblem of the following form:

$$\min_{d\in\mathbb{R}^n} \quad c^{\mathrm{T}}d \tag{8a}$$

$$\text{s.t.} \quad \|D_k^{-1}d\| \le \Delta_k. \tag{8b}$$

We would like to choose D_k in such a way that the solution of (8) will yield $s_k = -x_k$. The KKT system of (8) is

$$\begin{aligned} c + \lambda_k D_k^{-2} s_k &= 0, \\ \lambda_k(\Delta_k - \|D_k^{-1}s_k\|) &= 0, \\ \lambda_k &\ge 0. \end{aligned} \tag{9}$$

From (9), we have

$$s_k = -\frac{1}{\lambda_k} D_k^2 c \tag{10}$$

with

$$\lambda_k = \frac{c^{\mathrm{T}}x_k}{\Delta_k^2}. \tag{11}$$

Remembering that we require $s_k = -x_k$, we can set

$$(D_k)_{ii} = \sqrt{\lambda_k}\sqrt{\frac{x_k^{(i)}}{c^{(i)}}}, \quad i = 1,\dots,n. \tag{12}$$

For general nonlinear problems, we use the scaling technique (11)–(12) for those variables that seem to be active likely and set $(D_k)_{ii} = 1$ for inactive variables.

In order to make our descriptions clear, we need to introduce two vectors a_k and b_k, which are defined componentwise by

$$a_k^{(i)} = x_k^{(i)} - l^{(i)}, \quad b_k^{(i)} = u^{(i)} - x_k^{(i)}, \quad i = 1,\dots,n.$$

Then, we define two index sets as follows:

$$\mathcal{S}_k^1 = \{i : a_k^{(i)} \le \Delta_k, g_k^{(i)} \ge \epsilon a_k^{(i)}\}, \quad \mathcal{S}_k^2 = \{i : b_k^{(i)} \le \Delta_k, -g_k^{(i)} \ge \epsilon b_k^{(i)}\}, \tag{13}$$

where $\epsilon \ll 1$ is a positive constant. We consider these two sets as some prediction of active indices. Motivated by (11), we define a parameter t_k as

$$t_k = \frac{\sqrt{\sum_{i\in\mathcal{S}_k^1} a_k^{(i)} g_k^{(i)} + \sum_{i\in\mathcal{S}_k^2} b_k^{(i)} |g_k^{(i)}|}}{\Delta_k}. \tag{14}$$

And, our new scaling matrix D_k is defined by

$$(D_k)_{ii} = \begin{cases} t_k \cdot \sqrt{\dfrac{a_k^{(i)}}{g_k^{(i)}}}, & i \in \mathcal{S}_k^1, \\ t_k \cdot \sqrt{\dfrac{b_k^{(i)}}{|g_k^{(i)}|}}, & i \in \mathcal{S}_k^2, \\ 1, & i \notin \mathcal{S}_k \triangleq \mathcal{S}_k^1 \cup \mathcal{S}_k^2. \end{cases} \tag{15}$$

Because $x_k \in \mathcal{F}^\circ$, (15) is well defined and $(D_k)_{ii} > 0$ for all i. Moreover, $(D_k)_{ii}$ is always finite no matter the bound corresponding to $x^{(i)}$ is finite or not. In addition, it is easy to see that

$$\frac{a_k^{(i)}}{g_k^{(i)}} \le \frac{1}{\epsilon}, \quad i \in \mathcal{S}_k^1, \quad \frac{b_k^{(i)}}{|g_k^{(i)}|} \le \frac{1}{\epsilon},\ i \in \mathcal{S}_k^2,$$

and

$$t_k \le \frac{\sqrt{\sum_{i\in\mathcal{S}_k^1} a_k^{(i)} g_k^{(i)} + \sum_{i\in\mathcal{S}_k^2} b_k^i |g_k^i|}}{\Delta_k} \le \sqrt{\frac{n\|g_k\|}{\Delta_k}}.$$

Hence,

$$\|D_k\| \le \max\left\{\sqrt{\frac{n\|g_k\|}{\epsilon\Delta_k}}, 1\right\}. \tag{16}$$

The trust region subproblem of our affine scaling algorithm can now be given as follows:

$$\min_{d\in\mathbb{R}^n} \quad q_k(d) = g_k^{\mathrm{T}} d + \frac{1}{2} d^{\mathrm{T}} B_k d \tag{17a}$$

$$\text{s.t.} \quad \|D_k^{-1} d\| \le \Delta_k, \tag{17b}$$

$$x_k + d \in \mathcal{F}^\circ. \tag{17c}$$

Since problem (17) may not admit a solution, as the feasible set may not be a closed one, and an exact solution of (17) may be hard to find even if it exists, an approximate solution s_k is computed. Similar to classical analysis in trust region methods, we require s_k to satisfy the feasible conditions (17b)–(17c) and the following inequality:

$$q_k(s_k) \le \beta q_k(d_k^c) \tag{18}$$

for some constant $\beta \in (0, 1)$, where d_k^c is the *Cauchy point* of (17) defined by

$$d_k^c = \arg\min_d \left\{ q_k(d) : d = -\tau \frac{D_k^2 g_k}{\|D_k g_k\|}, \tau \ge 0,\ \|D_k^{-1} d\| \le \Delta_k,\ x_k + d \in \mathcal{F} \right\}. \tag{19}$$

Once the trust region trial step s_k is obtained, we need to compute the *actual reduction* in f

$$\mathrm{Ared}_k = f(x_k) - f(x_k + s_k)$$

and the *predicted reduction*

$$\mathrm{Pred}_k = q_k(0) - q_k(s_k).$$

The ratio

$$\rho_k = \frac{\mathrm{Ared}_k}{\mathrm{Pred}_k} \tag{20}$$

plays an important role in deciding whether the trial step can be accepted and in adjusting the trust region radius. To be more precise, the next iterate is defined by

$$x_{k+1} = \begin{cases} x_k + s_k & \text{if } \rho_k \ge \eta, \\ x_k & \text{if } \rho_k < \eta, \end{cases} \tag{21}$$

where η is a positive constant in $(0, 1)$. The trust region radius in the next iteration is given by

$$\Delta_{k+1} \in [\Delta_k, \min(\gamma_2 \Delta_k, \bar{\Delta})] \quad \text{if } \rho_k > \eta_2, \tag{22a}$$

$$\Delta_{k+1} = \Delta_k \quad \text{if } \eta_1 \leq \rho_k \leq \eta_2, \tag{22b}$$

$$\Delta_{k+1} \in (0, \gamma_1 \Delta_k] \quad \text{if } \rho_k < \eta_1, \tag{22c}$$

where $\eta_1 \leq \eta_2$ are two positive constants in $(\eta, 1)$.

The termination conditions for our algorithm are standard. Optimality conditions for problem (1) are well established [36]. Assume that the feasible point x_* is a *stationary point*, then the first-order necessary conditions hold at x_*:

$$\begin{aligned} &g^{(i)}(x_*) = 0 \quad \text{if } l^{(i)} < x_*^{(i)} < u^{(i)}, \\ &g^{(i)}(x_*) \geq 0 \quad \text{if } x_*^{(i)} = l^{(i)}, \\ &g^{(i)}(x_*) \leq 0 \quad \text{if } x_*^{(i)} = u^{(i)}. \end{aligned} \tag{23}$$

The *projected gradient* of the objective function f at point x, $\hat{g}(x)$, is defined componentwise by

$$\hat{g}^{(i)}(x) = \begin{cases} g^{(i)}(x) & \text{if } l^{(i)} < x^i < u^{(i)}, \\ \min\{0, g^{(i)}(x)\} & \text{if } x^{(i)} = l^{(i)}, \\ \max\{0, g^{(i)}(x)\} & \text{if } x^{(i)} = u^{(i)}. \end{cases} \tag{24}$$

It can be seen that (23) holds if and only if

$$\hat{g}(x_*) = 0. \tag{25}$$

In our algorithm, we will use $\|\hat{g}(x)\|$ as the first-order criticality measure [8,10]. We also stop our algorithm if either the trust region radius or the predicted reduction Pred_k is sufficiently small, or the trial step is too short.

Now, we are ready to give the descriptions of our complete algorithm for solving (1).

ALGORITHM 2.1 *Trust region affine scaling algorithm*

Step 0: Initialization.
Given $x_0 \in \mathcal{F}^\circ$, $\Delta_0 > 0$ and the constants $\bar{\Delta}$, ε, η, η_1 and η_2 satisfying

$$\bar{\Delta} > 1,\ \varepsilon > 0,\ \bar{\varepsilon} > 0,\ 0 < \gamma_1 < 1 < \gamma_2,\ 0 < \eta < \eta_1 \leq \eta_2 < 1.$$

Compute $f(x_0)$ and $\hat{g}(x_0)$, set $k = 0$.

Step 1: Termination test.
If $\|\hat{g}_k\| < \varepsilon$ or $\Delta_k < \bar{\varepsilon}$, then stop (returning x_k as a solution).

Step 2: Determine a trial step.
Compute s_k satisfying (18).
If $\text{Pred}_k < \bar{\varepsilon}$, or $\|s_k\| < \bar{\varepsilon}$, then stop (returning $x_k + s_k$ as a solution).

Step 3: Test to accept the trial step
Compute $f(x_k + s_k)$ and ρ_k through (20).
Set x_{k+1} by (21).

Step 4: Update the trust region radius.
Choose Δ_{k+1} by (22).
$k := k + 1$, go to Step 1.

In Section 4, we will discuss the details of the implementation of the above algorithm to practical computation.

3. Global convergence properties

In this section, we prove that the iterates x_k generated by Algorithm 2.1 are not bounded from stationary points. Before we give the convergence results, some assumptions need to be made.

(AS.1) $f \in \mathcal{C}^2$.
(AS.2) Given an initial point $x_0 \in \mathcal{F}^\circ$, the level set $\mathcal{L} = \{x \in \mathcal{F} : f(x) \leq f(x_0)\}$ is compact.
(AS.3) There exists a constant $\chi_B > 1$ such that $\|B_k\| \leq \chi_B$ for all k.

From (AS.1) and (AS.2), we know that there exist positive scalars M_1 and χ_g such that

$$\|\nabla^2 f(x)\| \leq M_1 \quad \text{and} \quad \|g_k\| \leq \chi_g$$

for all k and all $x \in \mathcal{L}$. Without loss of generality, we assume that $M_1 = \chi_B$, then for all k and all $x \in \mathcal{L}$,

$$\|B_k\| \leq \chi_B, \quad \|\nabla^2 f(x)\| \leq \chi_B.$$

The following lemma is a natural consequence of the construction of Algorithm 2.1.

LEMMA 3.1 *Suppose that* $\{x_k\}$ *is a sequence generated by Algorithm* 2.1, *then*

$$x_k \in \mathcal{F}^\circ$$

for all k.

Proof For each $x_k \in \mathcal{F}^\circ$, from the definition of s_k, we know that $x_k + s_k \in \mathcal{F}^\circ$. Thus, no matter $x_{k+1} = x_k + s_k$ or $x_{k+1} = x_k$, we always have $x_{k+1} \in \mathcal{F}^\circ$. Consequently, by induction, $x_k \in \mathcal{F}^\circ$ for all k. ■

Following the standard techniques for analysing trust region algorithms, we first give a lower bound for the predicted reduction Pred_k. In order to do so, we need to introduce a new vector v_k, which is defined componentwise by

$$v_k^{(i)} = \begin{cases} a_k^{(i)}, & i \in \mathcal{S}_k^1, \\ b_k^{(i)}, & i \in \mathcal{S}_k^2, \\ 1, & i \notin \mathcal{S}_k. \end{cases} \tag{26}$$

LEMMA 3.2 *Assume that* (AS.1)–(AS.3) *hold. If* s_k *satisfies* (18), *then*

$$\text{Pred}_k \geq \beta \frac{\|v_k \circ g_k\|}{2\bar{\Delta}} \min\left\{ \frac{\epsilon \|v_k \circ g_k\| \Delta_k}{n\chi_g \chi_B \bar{\Delta}}, \frac{\|v_k \circ g_k\|}{\chi_B \bar{\Delta}}, \Delta_k \right\}, \tag{27}$$

where '$\circ$*' represents the Hadamard product of two vectors, namely* $(v \circ g)^{(i)} = v^{(i)} g^{(i)}$ *for all i.*

Proof Denote $d(\tau) = -\tau(D_k^2 g_k / \|D_k g_k\|)(\tau \geq 0)$. First, we need to determine the bound on τ as the Cauchy step must be in the intersection of the trust region and the feasible region.

(1) $d(\tau)$ should satisfy the trust region constraint, namely $\|D_k^{-1} d_k^c(\tau)\| \leq \Delta_k$, which is equivalent to $\tau \in [0,\ \Delta_k]$.
(2) $d(\tau)$ should satisfy the feasibility constraint $x_k + d(\tau) \in \mathcal{F}$. First, we observe that

$$\|D_k g_k\|^2 = t_k^2 \left[\sum_{i \in \mathcal{S}_k^1} a_k^{(i)} g_k^{(i)} + \sum_{i \in \mathcal{S}_k^2} b_k^{(i)} |g_k^{(i)}| \right] + \sum_{i \notin \mathcal{S}_k} g_k^{(i)2} \geq t_k^4 \Delta_k^2. \tag{28}$$

Next, let us consider the following subcases:

Case 2.1 $i \in \mathcal{S}_k^1$. The feasibility condition $x_k^{(i)} + d^{(i)}(\tau) \geq l^{(i)}$ is equivalent to

$$\tau \leq \frac{\|D_k g_k\|}{t_k^2}. \tag{29}$$

Inequality (28) implies that $\|D_k g_k\|/t_k^2 \geq \Delta_k$. Hence, $x_k^{(i)} + d^{(i)}(\tau) \geq l^{(i)}$ is satisfied if $\tau \in [0, \Delta_k]$.

Case 2.2 $i \in \mathcal{S}_k^2$. Similar to Case 2.1, we can show that $x_k^{(i)} + d^{(i)}(\tau) \leq u^{(i)}$ always holds if $\tau \in [0,\ \Delta_k]$.

Case 2.3 $i \notin \mathcal{S}_k$. If $g_k^{(i)} = 0$, τ can be any value in the interval $[0, +\infty)$. It remains for us to consider the case where $g_k^{(i)} \neq 0$.

If $g_k^{(i)} \neq 0$, without loss of generality, we only need to consider the case $g_k^{(i)} > 0$. The feasibility condition $x_k^{(i)} + d^{(i)}(\tau) \geq l^{(i)}$ is equivalent to

$$\tau \leq a_k^{(i)} \frac{\|D_k g_k\|}{g_k^{(i)}}. \tag{30}$$

If $a_k^{(i)} > \Delta_k$, it is easy to see that $a_k^{(i)}(\|D_k g_k\|/g_k^{(i)}) \geq \Delta_k$. Thus, the feasibility condition $x_k^{(i)} + d^{(i)}(\tau) \geq l^{(i)}$ is guaranteed if $\tau \in [0, \Delta_k]$.

If $a_k^{(i)} \leq \Delta_k$, the fact that $i \notin \mathcal{S}_k$ implies that $0 < g_k^{(i)} < \epsilon a_k^{(i)}$. In this case, $a_k^{(i)}(\|D_k g_k\|/g_k^{(i)}) \geq ||D_k g_k||/\epsilon$. Hence, $x_k^{(i)} + d^{(i)}(\tau) \geq l^{(i)}$ holds as long as $\tau \leq ||D_k g_k||/\epsilon$.

To sum up, we have proved that

$$x_k + d(\tau) \in \mathcal{F} \quad \text{for all} \quad \tau \in \left[0, \min\left\{\frac{\|D_k g_k\|}{\epsilon}, \Delta_k\right\}\right]. \tag{31}$$

Define $\varphi : \mathbb{R} \to \mathbb{R}$ by setting $\varphi(\tau) = q_k(d(\tau))$. Let τ^* be the minimizer of φ on $[0, \min\{\|D_k g_k\|/\epsilon, \Delta_k\}]$. The definition of $\varphi(\tau)$ gives that

$$\varphi(\tau) = -\tau \|D_k g_k\| + \frac{\tau^2}{2} \frac{g_k^{\mathrm{T}} D_k^2 B_k D_k^2 g_k}{\|D_k g_k\|^2}.$$

Direct calculation indicates that

$$-\varphi(\tau^*) \geq \frac{\|D_k g_k\|}{2} \min\left\{\frac{\|D_k g_k\|}{\|D_k B_k D_k\|}, \frac{1}{\epsilon}\|D_k g_k\|, \Delta_k\right\}. \tag{32}$$

Under our assumptions, it follows from (16) that

$$\|D_k\| \leq \max\left\{\sqrt{\frac{n\chi_g}{\epsilon\Delta_k}}, 1\right\} \quad \Longrightarrow \quad \frac{1}{\|D_k\|^2} \geq \min\left\{\frac{\epsilon\Delta_k}{n\chi_g}, 1\right\}.$$

As a result, (32) implies that

$$\begin{aligned} -\varphi(\tau^*) &\geq \frac{\|D_k g_k\|}{2} \min\left\{\frac{\epsilon\|D_k g_k\|\Delta_k}{n\chi_g\chi_B}, \frac{1}{\chi_B}\|D_k g_k\|, \frac{1}{\epsilon}\|D_k g_k\|, \Delta_k\right\} \\ &\geq \frac{\|D_k g_k\|}{2} \min\left\{\frac{\epsilon\|D_k g_k\|\Delta_k}{n\chi_g\chi_B}, \frac{1}{\chi_B}\|D_k g_k\|, \Delta_k\right\} \end{aligned} \tag{33}$$

as $\epsilon \ll 1$. Moreover,

$$\begin{aligned}\|D_k g_k\|^2 &= \frac{[\sum_{i\in\mathcal{S}_k^1} a_k^{(i)} g_k^{(i)} + \sum_{i\in\mathcal{S}_k^2} b_k^{(i)} |g_k^{(i)}|]^2}{\Delta_k^2} + \sum_{i\notin\mathcal{S}_k} g_k^{(i)2} \\ &\geq \frac{[\sum_{i\in\mathcal{S}_k^1} a_k^{(i)} g_k^{(i)} + \sum_{i\in\mathcal{S}_k^2} b_k^{(i)} |g_k^{(i)}|]^2 + \sum_{i\notin\mathcal{S}_k} g_k^{(i)2}}{\max\{\bar{\Delta}^2, 1\}} \\ &\geq \frac{\|v_k \circ g_k\|^2}{\max\{\bar{\Delta}^2, 1\}} \\ &\geq \frac{\|v_k \circ g_k\|^2}{\bar{\Delta}^2} \end{aligned} \tag{34}$$

since $\bar{\Delta} > 1$. Combining (33)–(34) with (18)–(19), we obtain (27). ■

For simplicity, we denote

$$\zeta_1 = \frac{\beta}{2\bar{\Delta}}, \quad \zeta_2 = \frac{\epsilon}{n\chi_g\chi_B\bar{\Delta}}, \quad \zeta_3 = \frac{1}{\chi_B\bar{\Delta}}, \tag{35}$$

and then (27) can be rewritten as

$$\text{Pred}_k \geq \zeta_1 \|v_k \circ g_k\| \min\{\zeta_2 \|v_k \circ g_k\| \Delta_k, \zeta_3 \|v_k \circ g_k\|, \Delta_k\}. \tag{36}$$

Before analysis, we make another assumption.

(AS.4) If $\lim_{k\to\infty} \Delta_k = 0$, the following relation holds:

$$\|s_k\| = O(\Delta_k). \tag{37}$$

This assumption seems to be reasonable. To illustrate this, let us consider the scaled ball $\|D_k^{-1} d\| \leq \Delta_k$. For $i \in \mathcal{S}_k$, the corresponding axis length of this ball is Δ_k. By the definition of $\mathcal{S}_k^1$, we know that the corresponding variable is very close to the boundary (the distance is less than Δ_k), and also since $g_k^{(i)}$ is positive, intuitively it will approach the boundary if we search by some descent algorithm. So, $\|s_k\|$ should be $O(\Delta_k)$ along these coordinate directions. A similar consideration can be made on $\mathcal{S}_k^2$.

LEMMA 3.3 *Assume that* (AS.1)–(AS.4) *hold. Then, the sequence* $\{x_k\}$ *generated by Algorithm* 2.1 *satisfies*

$$\liminf_{k\to\infty} \|v_k \circ g_k\| = 0. \tag{38}$$

Proof In order to obtain a contradiction, we assume that there exist k_0 and $\varepsilon > 0$ such that

$$\|v_k \circ g_k\| \geq \varepsilon \quad \forall\, k \geq k_0.$$

Then, from Lemma 3.2, we deduce that for all $k \geq k_0$,

$$\text{Pred}_k \geq \zeta_1 \varepsilon \min\{\zeta_2 \varepsilon \Delta_k, \zeta_3 \varepsilon, \Delta_k\}.$$

Denote $\mathcal{S}$ as the set of all successful iterations

$$\mathcal{S} = \{k \in \mathbb{Z}_+ : \rho_k \geq \eta\}.$$

If there are only finitely many successful iterations, that is, $\mathcal{S}$ is a finite set, then for all k sufficiently large, from the framework of Algorithm 2.1, we have $\Delta_{k+1} \leq \gamma_1 \Delta_k$, which implies that

$$\sum_{k=0}^{\infty} \Delta_k < \infty.$$

If $\mathcal{S}$ is an infinite set, from (21), we know that $\text{Ared}_k \geq \eta \, \text{Pred}_k$ for all $k \in \mathcal{S}$. Since $\{f_k\}$ is a non-increasing sequence, it follows that

$$\sum_{k \in \mathcal{S}} \Delta_k < \infty.$$

The updating rules of the trust region radius in our algorithm imply that

$$\sum_{k=0}^{\infty} \Delta_k \leq \left(1 + \frac{\gamma_2}{1-\gamma_1}\right)\left(\Delta_1 + \sum_{k \in \mathcal{S}} \Delta_k\right) < \infty.$$

Hence,

$$\lim_{k \to \infty} \Delta_k = 0. \tag{39}$$

Consequently, for all k sufficiently large,

$$\text{Pred}_k \geq \zeta_1 \varepsilon \min\{\zeta_2 \varepsilon, 1\} \Delta_k. \tag{40}$$

Besides, (AS.4) implies that $s_k \to 0$. Thus, for all k sufficiently large,

$$|\text{Ared}_k - \text{Pred}_k| \leq \tfrac{1}{2} |s_k^{\mathrm{T}} (\nabla^2 f(\theta_k) - B_k) s_k| \leq \chi_B \|s_k\|^2, \tag{41}$$

where $\theta_k \in [x_k, x_{k+1}]$. By (AS.4), it can be concluded that

$$\lim_{k \to \infty} \rho_k = 1,$$

and hence for all k sufficiently large, $\Delta_{k+1} \geq \Delta_k$, which contradicts (39). Therefore, (38) is true. ■

Based on the above analysis, we obtain the main convergence result in this paper as follows.

THEOREM 3.1 *Under* (AS.1)–(AS.4), *the sequence* $\{x_k\}$ *generated by Algorithm* 2.1 *is not bounded from the stationary points.*

Proof From Lemma 3.3, assume that there exist a subsequence $\mathcal{K}$ and a point x_* such that

$$\lim_{\substack{k \to \infty \\ k \in \mathcal{K}}} x_k = x_*, \quad \lim_{\substack{k \to \infty \\ k \in \mathcal{K}}} \|v_k \circ g_k\| = 0. \tag{42}$$

In order to obtain convergence, we need to consider the following cases.

Case 1 $l^{(i)} < x_*^{(i)} < u^{(i)}$. By the definition of v_k in (26), we have that for all large $k \in \mathcal{K}$, no matter which is the set that encompasses the index i,

$$v_k^i \geq \min\left\{\frac{u^{(i)} - x_*^{(i)}}{2}, \frac{x_*^{(i)} - l^{(i)}}{2}, 1\right\} > 0,$$

which concludes that $g_*^{(i)} = 0$.

Case 2 $x_*^{(i)} = l^{(i)}$. If there are infinitely many indices $k \in \mathcal{K}$ such that $i \in \mathcal{S}_k^1$, since

$$g_k^{(i)} \geq \varepsilon a_k^{(i)},$$

it implies that $g_*^{(i)} \geq 0$. If there are only a finite number of indices $k \in \mathcal{K}$ such that $i \in \mathcal{S}_k^1$, for all large k, we have

$$v_k^{(i)} \geq \min\left\{\frac{u^{(i)} - l^{(i)}}{2}, 1\right\},$$

which indicates that $g_*^{(i)} = 0$.

Case 3 $x_*^{(i)} = u^{(i)}$. Similar to Case 2, we have that $g_*^{(i)} \leq 0$ in this case.

The above analysis shows that x_* is a KKT point. ■

4. Preliminary numerical experience

In this section, we examine the practical behaviour of Algorithm 2.1 given in Section 2. Our implementation is written in Matlab 7.6.0 (R2008a). The numerical results have been obtained on a PC with a 1.86 GHz Pentium Dual-Core microprocessor and 1 GB of memory running Fedora 8.0. We test all the 106 simple bound constrained optimization problems from CUTEr collection [19]. Due to memory limitation, 103 problems could be decoded by SifDec in our machine except problems CHENHARK, QRTQUAD and QUDLIN. Before we give the computational results, some issues need to be considered.

First, in our algorithm, since CUTEr can provide the exact Hessian of the functions, we set $B_k = \nabla^2 f(x_k)$ in the trust region subproblem (17). We also relax $x_k + d \in \mathcal{F}^\circ$ to $x_k + d \in \mathcal{F}$. Thus, by replacing $D_k^{-1} d$ with $\tilde{d}$, we obtain the following trust region subproblem:

$$\min_{\tilde{d} \in \mathbb{R}^n} \quad \tilde{q}_k(\tilde{d}) = (D_k g_k)^{\mathrm{T}} \tilde{d} + \frac{1}{2} \tilde{d}^{\mathrm{T}} (D_k B_k D_k) \tilde{d} \tag{43a}$$

$$\text{s.t.} \quad \|\tilde{d}\| \leq \Delta_k, \tag{43b}$$

$$D_k^{-1}(l - x_k) \leq \tilde{d} \leq D_k^{-1}(u - x_k). \tag{43c}$$

Since the exact solution of (43) is not easy to obtain, an obvious inexact solution can be obtained by truncating the exact solution of (43a)–(43b) to the feasible region of (43c). However, numerical results show that this approach is not a good idea, as the step obtained in this way is normally very small and does not satisfy the sufficient descent condition (18). Therefore, we apply Powell's subroutine [30] to obtain an inexact solution of the QP subproblem (43). The step $\tilde{d}_k$ obtained by Powell's method may lie on the boundary of the feasible region. Thus, we let

$$s_k = \beta D_k \tilde{d}_k \tag{44}$$

for $\beta \in (0, 1)$. Consequently, we always have $x_k + s_k \in \mathcal{F}^\circ$.

Second, the CUTEr problems provide the initial points x_{start}. However, the starting points may not be in $\mathcal{F}^\circ$. We modify the initial points as follows:

$$x_0^{(i)} = \begin{cases} l^{(i)} + 0.5\min\{1, u^{(i)} - l^{(i)}\} & \text{if } x_{\text{start}}^{(i)} < l^{(i)} + 10^{-12}, \\ u^{(i)} - 0.5\min\{1, u^{(i)} - l^{(i)}\} & \text{if } x_{\text{start}}^{(i)} > u^{(i)} - 10^{-12}, \\ x_{\text{start}}^{(i)} & \text{otherwise.} \end{cases}$$

Next, in some CUTEr problems, some of the variables may be fixed. Then, we deal with this case as follows. Let $\mathcal{P}$ be the set of indices of the variables which are to be fixed. Let e_i be the ith column of the n by n identity matrix and let P be the matrix made up of columns e_i, $i \notin \mathcal{P}$. Then, consider (17) as a function of the free variables $\bar{d} = P^{\mathrm{T}} d$.

The parameters used in our implementation are set as follows:

$$\Delta_0 = 1,\ \bar{\Delta} = 100,\ \eta = 10^{-8},\ \beta = 0.9999,\ \varepsilon = 10^{-5},\ \bar{\varepsilon} = 10^{-15},\ \epsilon = 10^{-8}.$$

For updating the trust region radius, we use the following strategy:

$$\Delta_{k+1} = \begin{cases} \max\{\Delta_k, 1.5\|D_k^{-1} s_k\|\} & \text{if } \rho_k > 0.9, \\ \Delta_k & \text{if } \rho_k \in [0.1, 0.9], \\ \max\{0.5\Delta_k, 0.75\|D_k^{-1} s_k\|\} & \text{if } \rho_k \in [10^{-8}, 0.1), \\ 0.5\Delta_k & \text{if } \rho_k \in (-\infty, 10^{-8}). \end{cases} \tag{45}$$

We compare Algorithm 2.1 with the famous code LANCELOT [9] and the subroutine fmincon from the Matlab optimization toolbox in Figures 1 and 2, respectively. Since some parameters in LANCELOT and fmincon are optional to meet different users' demand, in our experiments, we adopt the following options to make the comparison as fair as possible:

For LANCELOT,

BEGIN
gradient-accuracy-required 1e-5
exact-second-derivatives-used
trust-region-radius 1.0
maximum-number-of-iterations 1000
two-norm-trust-region-used
END

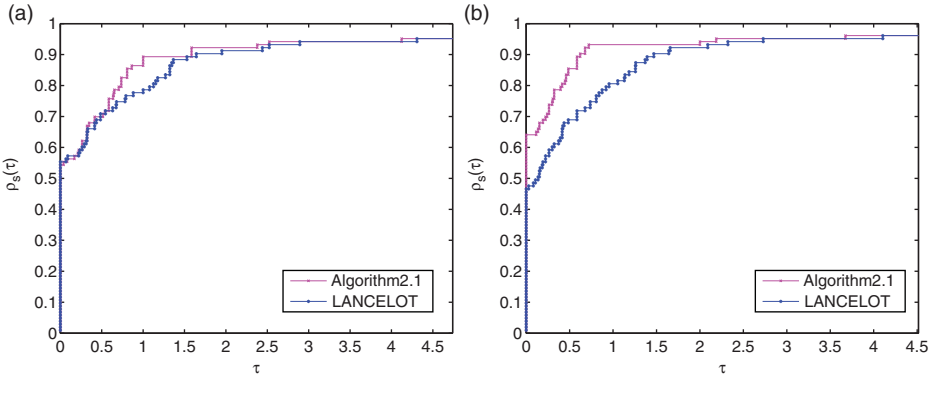

Figure 1. (a) Performance profile on function evaluations between Algorithm 2.1 and LANCELOT. (b) Performance profile on gradient evaluations between Algorithm 2.1 and LANCELOT.

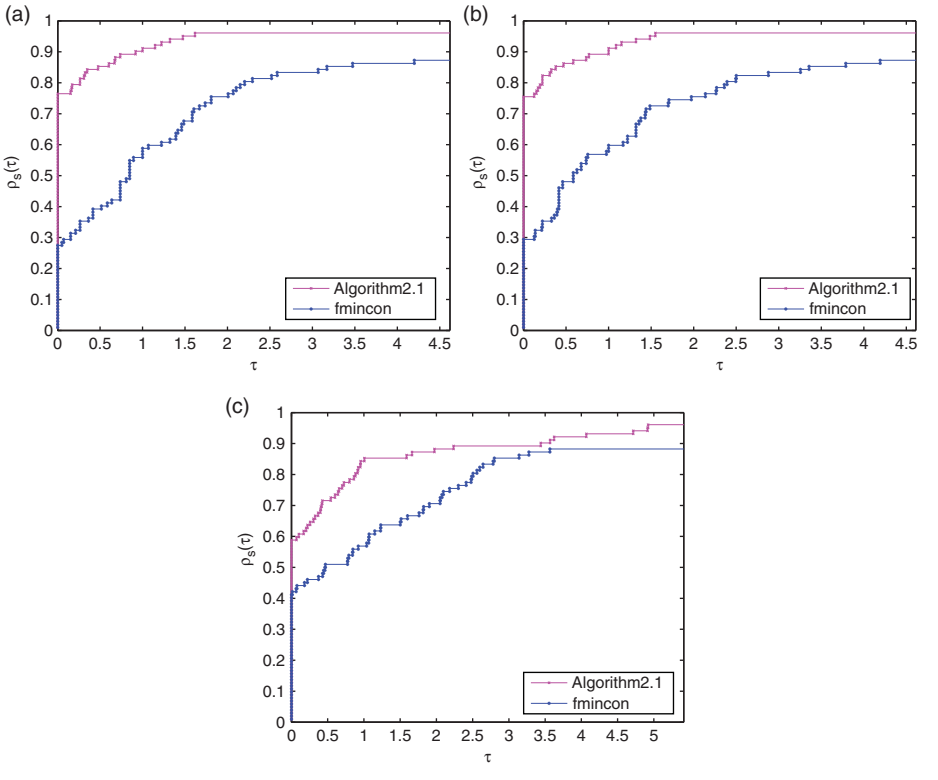

Figure 2. (a) Performance profile on function evaluations between Algorithm 2.1 and fmincon. (b) Performance profile on gradient evaluations between Algorithm 2.1 and fmincon. (c) Performance profile on CPU time between Algorithm 2.1 and fmincon.

For *fmincon*,
options = optimset('Algorithm', 'Trust-Region-Reflective Algorithm', 'Hessian', 'on', 'InitTrustRegionRadius', '1', 'MaxFunEvals', '1000', 'TolCon', 1e-12, 'TolFun', '1e-6', 'TolPCG', 1e-5, 'TolX', 1e-15).

In Table 1, we report the experiment results performing all the aforementioned algorithms. For each problem, the number of function evaluations '*nf*' and the number of gradient evaluations '*ng*' are given. And '*n*' denotes the dimension of the problems. The entry 'F' means that the algorithm stops when either the number of iterations or the number of function evaluations exceeds the maximum number. In addition, we report the CPU time of Algorithm 2.1 and fmincon. Moreover, we display all the problems which achieve different final objective function values by Algorithm 2.1 and fmincon in Table 2. Combining with Table 1, it can be seen that on these problems, Algorithm 2.1 obtains a better solution at a lower computational cost.

To give a vivid description of the algorithms, efficiency comparisons are shown in Figures 1–2, using the performance profiles introduced by Dolan and Moré [15]. The performance profiles are generated by executing solvers $\mathcal{S}$ on the test set $\mathcal{P}$. Considering the measure of interest, for example, number of function evaluations, for each problem $p \in \mathcal{P}$ and each solver $s \in \mathcal{S}$, define

$$n_{p,s} = \text{number of function evaluations to solve problem } p \text{ by solver } s.$$

To compare the performance on problem p by solver s with the best performance on p by any solver, define the performance ratio

$$r_{p,s} = \frac{n_{p,s}}{\min\{n_{p,s} : 1 \le s \le n_s\}}, \tag{46}$$

Table 1. Numerical results on simple bounded optimization problems from CUTEr.

		LANCELOT		fmincon			Algorithm 2.1		
Problem	n	nf	ng	nf	ng	CPU time	nf	ng	CPU time
3PK	30	16	17	25	24	0.1201	28	28	0.1875
ALLINIT	4	16	17	9	8	0.0314	10	9	0.0227
BDEXP	100	12	10	F	F	F	16	16	0.1555
BIGGSB1	100	12	13	13	12	0.1906	7	8	0.383
BLEACHNG	9	52	53	7	6	35.9055	7	7	35.6095
BQP1VAR	1	1	2	5	4	0.0117	3	3	0.0022
BQPGABIM	46	3	4	6	5	0.0574	3	3	0.0549
BQPGASIM	50	3	4	5	4	0.047	3	3	0.0628
CAMEL6	2	5	6	8	7	0.0278	6	6	0.0082
CHEBYQAD	100	140	110	28	27	3.429	62	46	6.4551
CVXBQP1	1000	4	5	7	6	10.913	3	3	4.6465
DECONVB	61	18	15	25	24	0.3352	F	F	F
EG1	3	4	5	9	8	0.0354	7	6	0.0125
EXPLIN	1200	21	22	30	29	81.478	21	22	109.4536
EXPLIN2	1200	17	18	18	17	48.4334	18	17	79.9722
EXPQUAD	120	22	20	23	22	0.2553	32	26	0.268
GRIDGENA	1226	99	86	4	3	8.2426	5	5	138.3152
HADAMALS	400	69	60	12	11	2.2947	12	12	2.744
HART6	6	8	8	8	7	0.0286	9	8	0.0167
HATFLDA	4	24	25	63	62	0.1445	11	11	0.0122
HATFLDB	4	20	21	21	20	0.0668	8	8	0.0096
HATFLDC	25	4	5	6	5	0.0294	5	5	0.0227
HIMMELP1	2	11	11	7	6	0.0227	14	15	0.08911
HS1	2	36	30	30	29	0.0667	29	25	0.0325
HS110	50	3	4	5	4	0.0222	3	3	0.0063
HS2	2	6	7	12	11	0.0351	9	8	0.0123
HS25	3	0	1	1	0	0.0118	0	1	0.0026
HS3	2	4	5	5	4	0.0144	8	8	0.0076
HS38	4	50	42	189	188	0.4567	47	39	0.0926
HS3MOD	2	4	5	12	11	0.037	8	8	0.0099
HS4	2	1	2	5	4	0.0142	3	3	0.0025
HS45	6	3	4	6	5	0.0234	5	5	0.0042
HS5	2	5	6	8	7	0.0252	6	6	0.0045
JNLBRNG1	1024	7	8	23	22	35.4785	5	5	68.5448
JNLBRNG2	1024	6	7	16	15	23.8144	6	6	34.6967
JNLBRNGA	1024	7	8	20	19	30.2861	6	7	142.6648
JNLBRNGB	1024	7	8	18	17	26.7303	10	11	48.599
LINVERSE	199	22	19	126	125	4.3002	21	17	6.5245
LOGROS	2	101	82	22	21	0.0619	35	26	0.0279
MAXLIKA	8	8	9	15	14	0.0912	46	41	0.1682
MCCORMCK	1000	8	9	28	27	43.6524	10	10	20.8506
MDHOLE	2	53	48	50	49	0.154	45	39	0.0435
NCVXBQP1	1000	9	10	10	9	15.5187	9	9	17.8724
NCVXBQP2	1000	8	9	49	48	76.3267	32	13	94.1669
NCVXBQP3	1000	11	12	36	35	56.0918	20	13	88.4003
NOBNDTOR	1024	7	8	15	14	3.2927	3	4	5.7361
NONSCOMP	484	12	13	21	20	5.6525	20	20	7.4561
OBSTCLAE	484	7	8	202	201	65.9593	11	11	10.5567
OBSTCLAL	1024	8	9	13	12	22.0112	F	F	F
OBSTCLBL	1024	8	9	10	9	14.6636	5	6	385.0913
OBSTCLBM	1024	4	5	9	8	13.3187	4	4	144.5011
OBSTCLBU	1024	8	9	9	8	13.9918	F	F	F
OSLBQP	8	2	3	10	9	0.0289	4	4	0.0042
PALMER1	4	39	25	25	24	0.087	31	22	0.0415
PALMER1A	6	71	63	27	26	0.0976	41	36	0.0838
PALMER1B	4	36	35	155	154	0.367	37	30	0.0623
PALMER1E	8	205	183	F	F	F	53	43	0.1409
PALMER2	4	25	23	9	8	0.0355	25	17	0.0261

(*Continued*)

Table 1. Continued

		LANCELOT		fmincon			Algorithm 2.1		
Problem	n	*nf*	*ng*	*nf*	*ng*	CPU time	*nf*	*ng*	CPU time
PALMER2A	6	178	156	513	512	1.1716	57	50	0.1067
PALMER2B	4	19	19	65	64	0.1744	31	25	0.0421
PALMER2E	8	108	101	F	F	F	125	99	0.3433
PALMER3	4	16	16	10	9	0.0397	25	20	0.0526
PALMER3A	6	176	153	612	611	1.3692	73	64	0.1553
PALMER3B	4	40	35	28	27	0.1047	18	16	0.0345
PALMER3E	8	58	55	386	385	1.1896	87	68	0.1965
PALMER4	4	23	21	9	8	0.0361	17	16	0.0268
PALMER4A	6	66	57	232	231	0.5319	54	44	0.1261
PALMER4B	4	35	31	22	21	0.0621	19	16	0.0548
PALMER4E	8	122	108	F	F	F	48	39	0.1129
PALMER5A	8	F	F	F	F	F	1000	828	2.1048
PALMER5B	9	F	F	F	F	F	150	144	0.3704
PALMER5D	8	9	10	6	5	0.031	14	14	0.0225
PALMER5E	8	F	F	F	F	F	298	295	0.6184
PALMER6A	6	255	218	F	F	F	102	91	0.2009
PALMER6E	8	75	70	522	511	1.3207	47	37	0.1364
PALMER7A	6	F	F	F	F	F	730	714	1.4255
PALMER7E	8	12	13	F	F	F	209	166	0.4624
PALMER8A	6	10	10	158	157	0.3751	52	40	0.2084
PALMER8E	8	67	62	F	F	F	26	24	0.1138
PENTDI	1000	1	2	12	11	19.043	F	F	F
PROBPENL	500	4	5	5	4	1.7571	6	6	2.2039
PSPDOC	4	8	8	21	20	0.0718	8	8	0.0173
QR3DLS	155	89	79	194	193	5.3834	61	59	3.0024
S368	100	12	11	11	10	0.6419	14	13	0.8284
SCOND1LS	500	217	186	84	83	0.3504	140	104	0.5689
SIM2BQP	1	1	2	6	5	0.0201	3	3	0.0047
SIMBQP	2	4	5	7	6	0.0229	7	7	0.0109
SINEALI	100	15	14	10	9	0.0883	12	10	0.0948
SPECAN	9	10	11	12	11	1.2789	10	10	1.2137
TORSION1	1024	10	11	11	10	16.1714	4	5	487.1617
TORSION2	1024	5	6	11	10	16.2599	4	4	492.338
TORSION3	1024	5	6	10	9	14.4309	6	7	45.7184
TORSION4	1024	4	5	10	9	14.4825	6	6	43.5582
TORSION5	1024	3	4	15	14	21.3003	5	6	12.4611
TORSION6	1024	5	6	15	14	21.3937	5	5	12.3605
TORSIONA	1024	10	11	9	8	13.339	5	6	158.266
TORSIONB	1024	4	5	9	8	13.1222	5	5	161.7327
TORSIONC	1024	5	6	9	8	12.9654	5	6	25.1166
TORSIOND	1024	4	5	9	8	13.0406	5	5	24.681
TORSIONE	1024	3	4	14	13	19.8415	5	6	23.1277
TORSIONF	1024	5	6	14	13	20.591	5	5	23.1345
WEEDS	3	55	49	56	55	0.163	32	28	0.0695
YFIT	3	95	81	F	F	F	45	39	0.0645

where n_s is the number of solvers. Whenever the solver s does not solve problem p, set $r_{p,s} = r_M$. Here, r_M is a very large preset positive constant. To obtain the overall performance of solver s on the test set $\mathcal{P}$, define

$$\rho_s(\tau) = \frac{1}{n_p}\text{size}\{p \in \mathcal{P} : \log_2 r_{p,s} \leq \tau\}. \tag{47}$$

Equivalently, $\rho_s(\tau)$ represents the probability that the performance ratio $r_{p,s}$ is within the factor 2^τ. It is easy to see that $\rho_s(0)$ is the probability that the solver s wins over the rest of the solvers.

From Figure 1, we can see that Algorithm 2.1 performs slightly better than LANCELOT on gradient evaluations and about the same on function evaluations. It shows that the probability

Table 2. Problems with different objective function values.

Problem	fmincon fval	Algorithm 2.1 fval
CHEBYQAD	0.0095	0.0087
LINVERSE	68.0015	68
PALMER1A	59.7543	0.0899
PALMER2A	0.0204	0.0171
PALMER3A	0.0379	0.0204
PALMER3E	0.049	5.1e−5
PALMER4	2.285e+3	2.424e+3
PALMER4A	0.0432	0.0406
PALMER6E	0.079	2.24e−4

that Algorithm 2.1 is faster on the function evaluations is near 55% (obtained from $\rho_s(0)$) and on the gradient evaluations is 62%. If we choose being within a factor of $1(\tau = 1)$ of the best solver as the scope of our interest, Algorithm 2.1 will be a winner for about 90% of the reported problems on gradient evaluations. So, our algorithm is effective and comparable with the famous solver LANCELOT. A similar analysis can be made from Figure 2. We can see that the new method performs better than fmincon not only on the function evaluations but also on the gradient evaluations, which shows that the new scaling technique that we proposed is more effective than that of Coleman and Li on the tested problems when considering the function and gradient evaluations. Figure 2(c) indicates that the overall performance on CPU time of Algorithm 2.1 is better than fmincon. However, we have to admit that on some large-scale problems given in Table 1, Algorithm 2.1 does not perform as well as fmincon. As pointed by a referee, the reason for this may be that the inexact solution of (43) obtained by Powell's subroutine is not good enough. We believe that our algorithm can be improved if we can solve (43) more efficiently, particularly for large-scale problems.

5. Conclusion

In this paper, we proposed a new affine scaling method for solving bound constrained optimization problems. It combines affine scaling strategy with trust region technique. By studying a simple linear programming problem, we derived a new scaling technique for the possible active variables. Our scaling matrix depends not only on the gradient of the current iterate and its distances to the bounds, but also on the relationship between these distances and the trust region radii. Different from traditional affine scaling methods, the trust region radius also plays an important role in the design of the scaling matrix. Through analysis, without any degeneracy, we proved that the iterates generated by our algorithm are not bounded away from the stationary points. Lastly, we tested our algorithm on the majority of the bound constrained optimization problems from CUTEr. The preliminary numerical experience verified the theoretical results obtained and also demonstrated that the new algorithm is comparable with LANCELOT. We also compared our new algorithm with the Matlab subroutine fmincon. The numerical results showed that our new scaling technique is competitive and a viable alternative to that of Coleman and Li.

Acknowledgements

This work was partially supported by NSFC grants 10831006 and 11021101 and CAS grant kjcx-yw-s7.

References

[1] E.R. Barnes, *A variation on Karmarkar's algorithm for solving linear programming problems*, Math. Program. 36 (1986), pp. 174–182.

[2] S. Bellavia, M. Macconi, and B. Morini, *An affine scaling trust-region approach to bound-constrained nonlinear systems* Appl. Numer. Math. 44 (2003), PP. 257–280.

[3] E.G. Birgin and J.M. Martinez, *Large-scale active-set box-constrained optimization method with spectral projected gradients*, Comput. Optim. Appl. 23 (2002), pp. 101–125.

[4] J. Bonnans and M. Bouhtou, *The trust region affine interior point algorithm for convex and nonconvex quadratic programming*, RAIRO Rech. Opér. 29 (1995), pp. 195–217.

[5] R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu, *A limited memory algorithm for bound constrained optimization*, SIAM J. Sci. Comput. 16 (1995), pp. 1190–1208.

[6] T.F. Coleman and Y. Li, *On the convergence of interior-reflection Newton methods for nonlinear minimization subject to bounds*, Math. Program. 67 (1994), pp. 189–224.

[7] T.F. Coleman and Y. Li, *An interior trust region approach for nonlinear minimization subject to bounds*, SIAM J. Optim. 6 (1996), pp. 418–445.

[8] A.R. Conn, N.I.M. Gould, and Ph.L. Toint, *Global convergence of a class of trust region algorithms for optimization with simple bounds*, SIAM J. Numer. Anal. 25 (1988), pp. 433–460; 26 (1989), pp. 764–767.

[9] A.R. Conn, N.I.M. Gould, and Ph.L. Toint, *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*, Springer, Heidelberg, New York, 1992.

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

[11] Y.H. Dai and R. Fletcher, *Projected Barzilai–Borwein methods for large-scale box-constrained quadratic programming*, Numer. Math. 100 (2005), pp. 21–47.

[12] J.E. Dennis and L.N. Vicente, *Trust-region interior-point algorithms for minimization problems with simple bounds*, in *Applied Mathematics and Paralle Computing*, K. Ritter, H. Fisher, H. Riedmüller and S. Schäffler, eds., Physica-Verlag, Springer, Berlin, 1996, pp. 97–107.

[13] I.I. Dikin, *Iterative solution of problems of linear and quadratic programming*, Dokl. Akad. Nauk SSSR 174 (1967), pp. 747–748. (Transl. Soviet Math. Dokl. 8 (1967), pp. 674–675.)

[14] I.I. Dikin and V.I. Zorkaltsev, *Iterative Solutions of Mathematical Programming Problems*, Nauka, Novosibirsk, 1980.

[15] E.D. Dolan and J.J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program. 91 (2002), pp. 201–213.

[16] F. Facchinei, J. Júdice, and J. Soares, *An active set Newton's algorithm for large-scale nonlinear programs with box constraints*, SIAM J. Optim. 8 (1998), pp. 158–186.

[17] F. Facchinei, S. Lucidi, and L. Palagi, *A truncated Newton algorithm for large scale box constrained optimization*, SIAM J. Optim. 12 (2002), pp. 1100–1125.

[18] A. Friedlander, J.M. Martinez, and S. Santos, *A new trust region algorithm for bound constrained minimization*, Appl. Math. Optim. 30 (1994), pp. 235–266.

[19] N.I.M. Gould, D. Orban, and Ph.L. Toint, *CUTEr, a constrained and unconstrained testing environment (revisited)*, ACM Trans. Math. Softw. 29 (2003), pp. 373–394.

[20] S. Gratton, M. Mouffe, Ph.L. Toint, and M. Weber-Mendonça, *A recursive l_∞-trust-region method for bound-constrained nonlinear optimization*, IMA. J. Numer. Anal. 28 (2008), pp. 827–861.

[21] W.W. Hager and H.C. Zhang, *A new active set algorithm for box constrained optimization*, SIAM J. Optim. 17 (2006), pp. 526–557.

[22] M. Heinkenschloss, M. Ulbrich, and S. Ulbrich, *Superlinear and quadratic convergence of affine-scaling interior-point Newton methods for problems with simple bounds without strict complementarity assumption*, Math. Program. 86 (1999), pp. 615–635.

[23] C. Kanzow and A. Klug, *On affine scaling interior-point Newton methods for nonlinear minimization with bound constraints*, Comput. Optim. Appl. 35 (2006), pp. 177–197.

[24] M. Lescrenier, *Convergence of trust region algorithms for optimization with bounds when strict complementarity does not hold*, SIAM J. Numer. Anal. 28 (1991), pp. 476–495.

[25] C. Lin and J.J. Moré, *Newton's method for large bound-constrained optimization problem*, SIAM J. Optim. 9 (1999), pp. 1100–1127.

[26] R.D.C. Monteiro and T. Tsuchiya, *Global convergence of the affine scaling algorithm for convex quadratic programmming*, SIAM J. Optim. 8 (1998), pp. 26–58.

[27] R.D.C. Monteiro and Y. Wang, *Trust region affine scaling algorithms for linearly constrained convex and concave programs*, Math. Program. 80 (1998), pp. 283–313.

[28] J.J. Moré and G. Toraldo, *On the solution of large quadratic programming problems with bound constraints*, SIAM J. Optim. 1 (1991), pp. 93–113.

[29] Q. Ni and Y. Yuan, *A subspace limited memory quasi-Newton algorithm for large-scale nonlinear bound constrained optimization*, Math. Comput. 66 (1997), pp. 1509–1520.

[30] M.J.D. Powell, *The BOBYQA algorithm for bound constrained optimization without derivatives*, Technical Report, Department of Applied Mathematics and Theoretical Physics, NA2009/06, Cambridge, 2009.

[31] A. Schwartz and E. Polak, *Family of projected descent methods for optimization problems with simple bounds*, J. Optim. Theory Appl. 92 (1997), pp. 1–31.

[32] J. Sun, *A convergence analysis for a convex version of Dikin's algorithm*, Ann. Oper. Res. 62 (1996), pp. 357–374.
[33] P. Tseng, *Convergence properties of Dikin's affine scaling algorithm for nonconvex quadratic minimization*, J. Glob. Optim. 30 (2004), pp. 285–300.
[34] M. Ulbrich, S. Ulbrich, and M. Heinkenschloss, *Global convergence of trust-region interior-point algorithms for infinite-dimensional nonconvex minimization subject to point-wise bounds*, SIAM J. Control Optim. 37 (1999), pp. 731–764.
[35] R.J. Vanderbei, M.S. Meketon, and B.A. Freedman, *A modification of Karmarkar's linear programming algorithm*, Algorithmica 1 (1986), pp. 395–407.
[36] Y. Yuan, *Computational Methods for Nonlinear Optimization (in Chinese)*, Science Press, Beijing, China, 2007.
[37] C. Zhu, R.H. Byrd, P. Lu, and J. Nocedal, *L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization*, ACM Trans. Math. Softw. 23 (1997), pp. 550–560.