Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational and Applied Mathematics 222 (2008) 429–439
JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS
www.elsevier.com/locate/cam

Modified subspace limited memory BFGS algorithm for large-scale bound constrained optimization☆

Yunhai Xiao[a,*], Hongchuan Zhang[b]

[a] *Institute of Applied Mathematics, School of Mathematics and Information Science, Henan University, Kaifeng, Henan, 475004, PR China*
[b] *College of Electric and Information Engineering, Hunan University, Changsha, 410082, PR China*

Received 19 April 2006; received in revised form 7 September 2007

Abstract

In this paper, a subspace limited memory BFGS algorithm for solving large-scale bound constrained optimization problems is developed. It is modifications of the subspace limited memory quasi-Newton method proposed by Ni and Yuan [Q. Ni, Y.X. Yuan, A subspace limited memory quasi-Newton algorithm for large-scale nonlinear bound constrained optimization, Math. Comput. 66 (1997) 1509–1520]. An important property of our proposed method is that more limited memory BFGS update is used. Under appropriate conditions, the global convergence of the method is established. The implementations of the method on CUTE test problems are presented, which indicate the modifications are beneficial to the performance of the algorithm.

Keywords: Bound constrained problem; Limited memory BFGS method; Projected line search; Stationary point; Gradient projection

1. Introduction

The bound constrained optimization problem to be considered is

$$\min f(x) \quad \text{s.t. } l \le x \le u, \tag{1.1}$$

where $f : \Re^n \to \Re$ is continuously differentiable, and $l, u \in \Re^n$ with $l < u$. Here and throughout the vector inequality is element-wise. The gradient of f at x is denoted by $\nabla f = (\nabla f_1(x), \ldots, \nabla f_n(x))$. Let Ω be the feasible region of (1.1), i.e.

$$\Omega = \{x \in R^n : l \le x \le u\}. \tag{1.2}$$

A vector $\overline{x} \in \Omega$ is said to be a stationary point for problem (1.1) if it satisfies

$$\begin{cases} l_i = \overline{x}_i \Rightarrow \nabla f_i(\overline{x}) \ge 0, \\ l_i < \overline{x}_i < u_i \Rightarrow \nabla f_i(\overline{x}) = 0, \\ \overline{x}_i = u_i \Rightarrow \nabla f_i(\overline{x}) \le 0. \end{cases} \tag{1.3}$$

☆ This work was supported by Chinese NSF grant 10471036.
* Corresponding address: School of Mathematics and Information Science, Henan University, 475000 Kaifeng, Henan, PR China.
E-mail addresses: yunhai816@163.com (Y. Xiao), tingboweila@126.com (H. Zhang).

0377-0427/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2007.11.014

Strict complementarity is said to be hold at $\overline{x}$ if the strict inequality hold in the first and the third implications of (1.3). The bound constrained minimization algorithms are usually used as subalgorithms for solving the subproblems in the augmented Lagrangian and penalty methods for general constrained optimization problems (see [1,2,9,10,13,22]).

Many algorithms for solving this type of problems are based on active set strategies (see [4,14,15,21]). The approaches of this class are quite efficient for relatively small dimensional problems, but are typically unattractive for large-scale problems (see [3,12]). The main reason is that at most one constraint can be added to or dropped from the active set at each iteration. Recently, there is a growing interest in the design of active set methods that are capable of making rapid changes to incorrect predictions (e.g. [3,4,14,15]).

The gradient projection methods (e.g. [3]) provide an alternative way for the solution of problem (1.1). They have the advantage that many constraints can be added to or dropped from the active set at each step. The gradient projection technique has been further studied for solving nonlinear optimization problems, both for general linearly constrained case and for the bound constrained case (see [5,8,9,19,20,23]). The bound constrained optimization problems have received much attention in recent decades, we refer to [16,17] for good reviews.

In [24], Ni and Yuan proposed a subspace limited memory quasi-Newton algorithm for solving problem (1.1). At each iteration, the active and inactive variables are determined by a small tolerance ϵ, and the inactive variables are updated by limited memory BFGS method. The reported numerical experiments [24] show that this method is effective for solving large-scale bound constrained optimization problems. The purpose of this paper is to further study the application of the limited memory BFGS method for the solution of problem (1.1). The proposed method in this paper can be considered as the modifications of the subspace limited memory BFGS method by Ni and Yuan. Unlike [24] where only the active variables are updated by the limited memory BFGS method, in this paper, we also adopt the limited memory BFGS method to update parts of the active variables. Specifically, at each iteration, we use a parameter to estimate the active or inactive variables, and the parameter becomes zero automatically as the iteration progresses. Under appropriate, we establish the global convergence of the method. In addition, the proposed method reserves the advantages of the limited memory BFGS method. The numerical experiments of the method on a set of large problems indicated are promising.

The paper is organized as follows. We first discuss the construction of our new algorithm in the next section. In Section 3, we establish the global convergence of the method. In Section 4, we test the performance of the proposed algorithm and compare it with the well known routine L-BFGS-B. Throughout the paper, the symbol $\|\cdot\|$ denotes the Euclidean norm of vectors.

2. Motivation and the new algorithm

In this section, based on the method in [24], we propose a new algorithm for solving (1.1). The method generates a sequence of points $\{x^k\}$ by

$$x^{k+1} = x^k + \alpha_k d^k, \quad k = 0, 1, 2, \ldots,$$

where d^k is a descent direction of f at x^k, and α_k is the steplength that is determined by a projected line search. In what follows, we describe the method in detail.

In order to introduce the procedure that estimate the active bounds, we give a sufficiently small scalar ϵ_b, such that $0 < \epsilon_b < \min_i \frac{1}{3}(u_i - l_i)$. Let

$$\omega_k = \|x^k - P_\Omega[x^k - \nabla f(x^k)]\| \quad \text{and} \quad \epsilon_k = \min\{\epsilon_b, \omega_k\}, \tag{2.1}$$

where $P_\Omega[\cdot]$ is the projection on Ω, that is

$$P_\Omega[x] = \begin{cases} x_i, & \text{if } l_i \le x_i \le u_i, \\ l_i, & \text{if } x_i < l_i, \\ u_i, & \text{if } x_i > u_i. \end{cases} \tag{2.2}$$

Note that while the starting point is far from a stationary point, then ϵ_k will be bounded away from zero, and becomes zero automatically as the iteration progresses.

For $x \in \Omega$, we define index set $A(x)$ and $B(x)$ as follows:

$$A(x) = \{i : l_i \le x_i \le l_i + \epsilon_k \text{ or } u_i - \epsilon_k \le x_i \le u_i\}, \tag{2.3}$$

$$B(x) = \{1, \ldots, n\} \backslash A(x) = \{i : l_i + \epsilon_k \le x_i \le u_i - \epsilon_k\}. \tag{2.4}$$

The variables with indices in $A(x)$ are called active variables, while the variables with indices in $B(x)$ are called inactive variables.

In order to obtain the search direction of the active variables, as in [24], we also partition the active set $A(x)$ into three parts,

$$A_1(x) = \{i : x_i = l_i \text{ or } x_i = u_i, \text{ and } (l_i + u_i - 2x_i)\nabla f_i \ge 0\}, \tag{2.5}$$

$$A_2(x) = \{i : l_i \le x_i \le l_i + \epsilon_k \text{ or } u_i - \epsilon_k \le x_i \le u_i, \text{ and } (l_i + u_i - 2x_i)\nabla f_i < 0\}, \tag{2.6}$$

$$A_3(x) = \{i : l_i < x_i \le l_i + \epsilon_k \text{ or } u_i - \epsilon_k \le x_i < u_i, \text{ and } (l_i + u_i - 2x_i)\nabla f_i \ge 0\}. \tag{2.7}$$

Consider the above implications, we note that $A_1(x)$ is the index set of variables where the corresponding gradient satisfy the first (or third) implication of (1.3), therefore it is reasonable to be fixed. $A_3(x)$ is the set of the variables where the steepest directions move towards the boundary, thus the steepest descent direction in this subspace should be backtracked to ensure feasibility. $A_2(x)$ is the index set of the variables where the steepest descent direction move into the interior of the feasible region, in this situation, unlike [24] where the steepest descent direction is used, we use the Newton direction instead. We assume, from now on, the variables with indices in $B(x) \cup A_2(x)$ are called free variables, although most of the prevenient researchers did not use the assertion.

Let Z be the matrix whose columns are $\{e_i \mid i \in B(x^k) \cup A_2(x^k)\}$, and P be the matrix whose columns are $\{e_i \mid i \in A_3(x^k)\}$, where e_i is the i-th column of the identity matrix I in $R^{n\times n}$. Let $\overline{H}^k = Z^T H^k Z$, where H^k is an approximation of $\nabla^2 f^{-1}(x^k)$. From now on, we abbreviate $\nabla f(x^k)$ as g^k for sake of simplicity.

For simplicity, we let $B^k = B(x^k)$ and $A_i^k = A_i(x^k)(i = 1, 2, 3)$. The search direction d^k is defined by

$$d_i^k = \begin{cases} 0, & \text{if } i \in A_1^k, \\ -(PP^T \Lambda_k g^k)_i, & \text{if } i \in A_3^k, \\ -(Z\overline{H}^k Z^T g^k)_i, & \text{if } i \in B^k \cup A_2^k, \end{cases} \tag{2.8}$$

where $\Lambda_k = diag(\lambda_1^k, \ldots, \lambda_n^k)$ which is given by

$$\lambda_i^k = \begin{cases} 0, & \text{if } i \notin A_3^k, \\ \dfrac{x_i^k - l_i}{g_i^k}, & \text{if } l_i < x_i \le l_i + \epsilon_k \text{ and } x_i^k - g_i^k \le l_i, \\ \dfrac{x_i^k - u_i}{g_i^k}, & \text{if } u_i - \epsilon_k \le x_i < u_i \text{ and } x_i^k - g_i^k \ge u_i, \\ 1, & \text{otherwise.} \end{cases} \tag{2.9}$$

The calculation of the search direction at current point x^k in the subspace of $i \in A_3^k$ can be explained as follows: Let $d_i^k = -g_i^k$, if in the next iteration we have $x_i^k + d_i^k = x_i^{k+1} \in \Omega$, we set $\lambda_i^k = 1$. Otherwise we backtrack the steepest direction with steplength $\lambda_i^k = \frac{x_i^k - l_i}{g_i^k}$, and in the next iteration we also have $x_i^{k+1} = x_i^k + \lambda_i^k d_i^k = l_i$. The case of the third implication in (2.9) is analogous. This is to say, the definition of Λ_k in (2.9) ensures that

$$l_i \le x_i^k + d_i^k \le u_i, \quad \forall i \in A_3^k \tag{2.10}$$

holds.

The following result shows that whenever $d^k \ne 0$, it is at least a descent direction for objective function f at the current point, it is very important to establish our global convergence theorem.

Lemma 2.1. *If H^k is positive definite, then d^k defined by (2.8) satisfies*

$$(d^k)^T g^k \le 0, \tag{2.11}$$

and the equality holds if and only if $d^k = 0$.

Proof. From the definition of search direction d^k in (2.8), we have

$$(d^k)^T g^k = \sum_{i\in A_3^k} -(PP^T \Lambda_k g^k)_i g_i^k + \sum_{i\in B^k\cup A_2^k} -g_i^k (Z\overline{H}^k Z^T g^k)_i \le 0.$$

The above relation comes from the positive definite of H^k and the definition of the active set A_3^k, (2.8) and (2.9). Which also indicates $(d^k)^T g^k = 0$ if and only if $d^k = 0$. □

The limited memory BFGS method is an adaptation of the BFGS method to large-scale problems. In the limited memory BFGS method, matrix H^k is obtained by updating the basic matrix $H^0 m$ times using BFGS formula with the previous m iterations. The standard BFGS correction with H^k have the following form:

$$H^{k+1} = (V^k)^T H^k V^k + \rho^k s^k (s^k)^T, \tag{2.12}$$

where $\rho^k = \frac{1}{(y^k)^T s^k}$, $s^k = x^{k+1} - x^k$, $y^k = g^{k+1} - g^k$, and $V^k = I - \rho^k y^k (s^k)^T$. Therefore, H^{k+1} in the limited BFGS method has the following form:

$$\begin{aligned} H^{k+1} &= (V^k)^T[(V^{k-1})^T H^{k-1} V^{k-1} + \rho^{k-1} s^{k-1}(s^{k-1})^T]V^k + \rho^k s^k (s^k)^T \\ &= [(V^k)^T \cdots (V^{k-m+1})^T] H^{k-m+1} [V^{k-m+1} \cdots V^{k-1}] \\ &\quad + \rho^{k-m+1}[(V^{k-1})^T \cdots (V^{k-m+2})^T] s^{k-m+1}(s^{k-m+1})^T [V^{k-m+2} \cdots V^{k-1}] \\ &\quad + \cdots + \rho^k s^k (s^k)^T. \end{aligned} \tag{2.13}$$

To maintain the positive definiteness of the limited memory BFGS matrix, some researchers suggest to discard a correction pair $\{s^k, y^k\}$ if $(y^k)^T s^k \le 0$ (e.g. [6,7]). Another approach which proposed by Powell can be seen in [25].

The projected line search has been used by several authors for solving quadratic and nonlinear programming problems with bounds on the variables (see [3,19,23,24]). The projected search requires that a steplength, α_k, be chosen such that

$$\phi^k(\alpha) \le \phi^k(0) + \delta \phi^{k'}(0)\alpha \tag{2.14}$$

is satisfied for some constant $\delta \in (0, \frac{1}{2})$. Here ϕ^k is the piecewise twice continuously differentiable function, that is

$$\phi^k(\alpha) = f(P_\Omega[x^k + \alpha d^k]).$$

Now, we state the steps of the modified subspace limited memory BFGS (MSLBFGS) algorithm as follows.

Algorithm 2.1. Step 0. Given starting point $x^0 \in \Omega$, constant $\delta \in (0, \frac{1}{2})$, $m \in (3, 20)$ and $\beta \in (0, 1)$, the "basic matrix" H^0, a small scalar $\epsilon_b > 0$ and $0 < \sigma_1 < \sigma_2 < 1$; compute $f(x^0)$, g^0 and set $k = 0$.
Step 1. Determine B^k, A_i^k $(i = 1, 2, 3)$ according to (2.3)–(2.7).
Step 2. Determine d^k by (2.8).
Step 3. If $d^k = 0$, then stop.
Step 4. If

$$f(P_\Omega[x^k + \alpha_{k,j} d^k]) \le f(x^k) + \delta \nabla f(x^k)^T (P_\Omega[x^k + \alpha_{k,j} d^k] - x^k). \tag{2.15}$$

then define $\alpha_k = \alpha_{k,j}$, else find $\alpha_{k,j} \in [\sigma_1 \alpha_{k,j}, \sigma_2 \alpha_{k,j}]$, go to Step 4.
Step 5. Set $x^{k+1} = P_\Omega[x^k + \alpha_k d^k]$. Update H^k to get H^{k+1} by (2.13).
Step 6. Let $k = k + 1$. Go to Step 1.

3. Convergence analysis

In the section, we show that the sequence $\{x^k\}$ generated by Algorithm 2.1 converges to a stationary point of problem (1.1). The following result shows that the Step 4 is well defined.

Lemma 3.1. *Suppose the sequence* $\{d^k\}$ *and* $\{x^k\}$ *be generated by Algorithm 2.1, and assume that* $d^k \neq 0$. *Then we have*

$$\min\left\{1, \frac{\|u-l\|_\infty}{\|d^k\|_\infty}\right\} \geq \beta^k \geq \min\left\{1, \frac{\epsilon_k}{\|d^k\|_\infty}\right\}, \tag{3.1}$$

where $\beta^k = \sup_{0\leq\gamma\leq 1}\{\gamma \mid l \leq x^k + \gamma d^k \leq u\}$.

Proof. Since $d^k \neq 0$, from (2.8) we have $g^k \neq 0$, so also $\epsilon_k \neq 0$. By the definition of β^k, x^k and $x^k + \beta^k d^k$ are feasible points of (1.1), which gives

$$\|\beta^k d^k\|_\infty \leq \|u-l\|_\infty.$$

Thus the first part of (3.1) is true.

Now we show the second part of (3.1). It is sufficient to prove that

$$x_i^k + \overline{\beta} d_i^k \in [l_i, u_i] \tag{3.2}$$

for all $i = 1, \ldots, n$, where $\overline{\beta} = \min\{1, \frac{\epsilon_k}{\|d^k\|_\infty}\}$. If $i \in B^k$, (3.2) follows from (2.4) and $|\overline{\beta} d_i^k| \leq \epsilon_k$. If $i \in A_1^k$, (3.2) is trivial as $d_i^k = 0$. If $i \in A_3^k$, it follows from definition (2.9) that

$$x_i^k + d_i^k \in [l_i, u_i] \tag{3.3}$$

which implies (3.2). Finally we consider the case when $i \in A_2^k$. We have $d_i^k = -(Z\overline{H}^k Z^T g^k)_i \neq 0$. If $d_i > 0$, then $x_i^k \in [l_i, l_i + \epsilon_k]$ which shows that

$$l_i \leq x_i^k < x_i^k + \overline{\beta} d_i^k \leq x_i^k + \epsilon_k \leq l_i + 2\epsilon_k < u_i. \tag{3.4}$$

Similarly, if $d_i^k < 0$, we have

$$l_i < u_i - 2\epsilon_k \leq x_i^k - \epsilon_k \leq x_i^k + \overline{\beta} d_i^k < x_i^k \leq u_i. \tag{3.5}$$

Hence (3.2) holds for all $i = 1, \ldots, n$. Therefore, we have shown that the relations (3.1) hold. □

The following lemma indicates that the search direction does not vanish if the iteration point is not a stationary point. The proof can be found in [24, Lemma 3.1], and we omit it here.

Lemma 3.2. *Suppose the sequence* $\{d^k\}$ *and* $\{x^k\}$ *be generated by Algorithm 2.1. Then* x^k *is a stationary point of problem (1.1) if and only if* $d^k = 0$.

The following theorem establishes the global convergence of Algorithm 2.1, and the proof is similar with Theorem 3.2 in [24].

Theorem 3.1. *Suppose the sequence* $\{d^k\}$ *and* $\{x^k\}$ *be generated by Algorithm 2.1. Assume that* f *is twice continuously differentiable in* Ω, *and there exists positive constants* γ_1, γ_2 *such that*

$$\gamma_1 \|Z^T g^k\|^2 \leq (g^k)^T Z\overline{H}^k Z^T g^k \tag{3.6}$$

$$\|Z^T \overline{H}^k Z\| \leq \gamma_2 \tag{3.7}$$

for all k. *Then every accumulation point of* $\{x^k\}$ *is a stationary point of the problem (1.1).*

Proof. First we establish an upper bound for $(g^k)^T d^k$,

$$\begin{aligned}(g^k)^T d^k &= -(g^k)^T Z\overline{H}^k Z^T g^k - \|P^T \Lambda_k^{1/2} g^k\|^2 \\ &\leq -\left(\gamma_1 \|Z^T g^k\|^2 + \sum_{i\in A_3^k} \tau_i^k |g_i^k|\right),\end{aligned} \tag{3.8}$$

where $\tau_i^k = \min\{|g_i^k|,\ |x_i^k - l_i|,\ |u_i - x_i^k|\}$. We also have

$$\|d^k\|^2 = \|Z\overline{H}^k Z^T g^k\|^2 + \|P^T \Lambda_k g^k\|^2. \tag{3.9}$$

Because $\lambda_i^k \in [0, 1]$, and $\overline{H}^k$ satisfies (3.7), it follows from (3.8) and (3.9) that

$$\|d^k\|^2 \leq -\max\{1, \gamma_2\}(g^k)^T d^k. \tag{3.10}$$

Furthermore, from (3.7) and (3.9) yield

$$\|d^k\|^2 \leq \gamma_2^2 \|g^k\|^2 + \|g^k\|^2 \leq (\gamma_2^2 + 1)\eta_1, \tag{3.11}$$

where $\eta_1 = \max_{x\in\Omega} \|g^k\|^2$. Thus from (3.1) and (3.11), there exists a constant $\overline{\beta} \in (0, 1)$ such that

$$\beta_k \geq \overline{\beta} \quad \text{for all } k. \tag{3.12}$$

If $\alpha_k < \sigma_1\overline{\beta}$, by the definition of α_k there exists $\alpha_{k,j}$ such that $\alpha_{k,j} \leq \frac{\alpha_k}{\sigma_1}$, and $\alpha_{k,j}$ is an unacceptable steplength, which implies that

$$\begin{aligned} f(x^k) + \delta\alpha_{k,j}(g^k)^T d^k &\leq f(x^k + \alpha_k d^k) \\ &\leq f(x^k) + \alpha_{k,j}(g^k)^T d^k + \frac{1}{2}\eta_2 \alpha_{k,j}^2 \|d^k\|^2, \end{aligned} \tag{3.13}$$

where $\eta_2 = \max_{x\in\Omega} \|\nabla^2 f(x)\|$. The above inequality and (3.10) imply that

$$\alpha_{k,j} \geq \frac{-2(1-\delta)(g^k)^T d^k}{\eta_2 \|d^k\|^2} \geq \frac{2(1-\delta)}{\eta_2 \max\{1, \gamma_2\}}. \tag{3.14}$$

Hence the above inequality and $\alpha_k \geq \sigma_1\alpha_{k,j}$ yield

$$\alpha_k \geq \min\left\{\frac{2\sigma_1(1-\delta)}{\eta_2 \max\{1, \gamma_2\}},\ \sigma_1\overline{\beta}\right\} > 0 \tag{3.15}$$

for all k. Because Ω is a bounded set,

$$\infty > \sum_{k=1}^{\infty}(f(x^k) - f(x^{k+1})) \geq \sum_{k=1}^{\infty} -\delta\alpha_k(g^k)^T d^k. \tag{3.16}$$

(3.15) and (3.16) show that

$$\sum_{k=1}^{\infty} -(g^k)^T d^k < \infty, \tag{3.17}$$

which implies

$$\lim_{k\to\infty} (g^k)^T d^k = 0. \tag{3.18}$$

It follows from (3.18) and (3.8) that

$$\lim_{k\to\infty} \|Z^T g^k\| = 0, \tag{3.19}$$

$$\lim_{k\to\infty} \sum_{i\in A_3^k} \tau_i^k |g_i^k| = 0. \tag{3.20}$$

For the remanent proof we can see [24, Theorem 3.2]. □

Conditions (3.6) and (3.7) are satisfied if the matrix H^k is adjusted by limited memory BFGS inverse m-update (2.13).

Table 4.1
Performance of MSLBFGS

Problem	Dim	iter	fcnt	time	$f(x)$	pginfn	pgtwon
nonscomp	1000	68	244	0.031	0.180671E−10	0.653346E−05	0.173956E−04
	2000	68	244	0.141	0.366668E−10	0.990403E−05	0.253062E−04
	5000	71	253	0.281	0.100082E−10	0.969158E−05	0.176238E−04
explin	12	68	239	0.016	−0.684995E+04	0.565795E−05	0.650469E−05
	120	91	317	0.016	−0.723756E+06	0.904604E−05	0.108707E−04
	1200	418	2000	0.375	−0.719252E+08	0.623492E+01	0.714967E+01
explin2	12	54	192	0.000	−0.709247E+04	0.688739E−05	0.688739E−05
	120	154	529	0.016	−0.724459E+06	0.849201E−05	0.106886E−04
	1200	416	2000	0.453	−0.719988E+08	0.113915E−02	0.113915E−02
mccormck	100	32	61	0.016	−0.917881E+02	0.711845E−05	0.189380E−04
	500	32	61	0.000	−0.457077E+03	0.711845E−05	0.189380E−04
	1000	32	61	0.016	−0.913689E+03	0.711845E−05	0.189380E−04
probpenl	1000	33	298	0.125	0.199800E−06	0.521194E−05	0.164810E−03
	5000	35	351	0.703	0.399921E−07	0.621565E−05	0.439512E−03
qrtquad	12	78	2000	0.016	−0.111826E+04	0.845710E+02	0.154986E+03
	120	400	2000	0.031	−0.364807E+07	0.114270E−02	0.127514E−02
	1200	354	2001	0.500	−0.360944E+10	0.192813E+05	0.345388E+05
bdexp	5000	1001	1002	7.859	0.571006E−01	0.152184E−03	0.792156E−02
hatfldc	25	58	147	0.000	0.651092E−10	0.956225E−05	0.193433E−04
	1000	49	126	0.031	0.483624E−10	0.896835E−05	0.196945E−04
	5000	53	136	0.188	0.551863E−10	0.899976E−05	0.150257E−04
hs110	50	1	2	0.000	−0.999000E+10	0.000000E+00	0.000000E+00
	100	1	2	0.000	−0.998002E+20	0.000000E+00	0.000000E+00
biggsb1	1000	1001	1795	0.531	0.197249E−01	0.493937E−03	0.320872E−02
	5000	1001	1795	2.531	0.197249E−01	0.493937E−03	0.320872E−02
hatflda	1000	2	2000	4.750	0.721579E+01	0.100000E+21	0.100000E+21
edensch	2000	35	95	0.031	0.120033E+05	0.984496E−05	0.327110E−03
edensch	2000	0	1	0.000	0.339990E+05	0.000000E+00	0.000000E+00
edensch	2000	160	2000	0.578	0.137024E+05	0.107196E−04	0.317539E−04
edensch	2000	156	2000	0.609	0.120046E+05	0.190917E−04	0.255474E−04
edensch	2000	52	171	0.109	0.144263E+05	0.574050E−05	0.311840E−04
penalty1	2000	1001	1082	1.062	0.195573E−01	0.258528E−04	0.112000E−02
penalty1	2000	1001	1078	0.984	0.196613E−01	0.163280E−03	0.580891E−02
penalty1	2000	12	79	0.031	0.411068E+02	0.100762E−05	0.212506E−04
penalty1	2000	31	143	0.047	0.950806E+02	0.627337E−05	0.114593E−03
cvbqp1	1000	1	2	0.000	0.225225E+05	0.000000E+00	0.000000E+00
nbcvbqp1	1000	1	2	0.000	−0.198680E+09	0.000000E+00	0.000000E+00
nbcvbqp2	1000	195	2000	0.406	−0.133388E+09	0.142076E−02	0.143546E−02
nbcvbqp3	1000	54	274	0.094	−0.657738E+08	0.942524E−05	0.961508E−05
qudlin	1000	1	2	0.000	−0.500000E+08	0.000000E+00	0.000000E+00
	5000	1	2	0.000	−0.125000E+10	0.000000E+00	0.000000E+00
s368	100	75	2000	0.922	0.105413E+10	0.970297E+00	0.570547E+01
harkerp2	100	58	2000	0.406	−0.957207E−06	0.999582E+00	0.999582E+01
pentdi	1000	21	62	0.000	−0.375000E+00	0.626885E−05	0.626885E−05
	5000	21	62	0.047	−0.375000E+00	0.626885E−05	0.626885E−05
chenhark	1000	820	2000	0.609	−0.996941E+00	0.175296E−02	0.577422E−02
sineali	1000	182	2001	0.500	−0.997205E+05	0.582061E+01	0.204143E+02

We note that our global convergence result based on a small scalar $\epsilon_b > 0$. If we initialize $\epsilon_b > 0$, then the algorithm falls into an exact active set framework. In this case, although more variables are updated by the limited memory BFGS method, we think the direction should be more complicated. For example, suppose that at the optimal solution the i-th component is at its lower bound l_i, and x_i^k is very near to the bound, we set $d_i^k = l_i - x_i^k$, and in the next iteration we have $x_i^{k+1} = l_i$, provided that λ_i^k is defined by the third implication of (2.9). The above strategy is very natural and simple, and it is easier to be performed than use limited memory BFGS update instead, the numerical experiments as follows indicate our claims.

Table 4.2
Performance of NIYUAN

Problem	dim	iter	fcnt	time	$f(x)$	pginfn	pgtwon
nonscomp	1000	60	174	0.031	0.124730E−10	0.811342E−05	0.203497E−04
	2000	233	634	0.281	0.218738E−10	0.975859E−05	0.106368E−04
	5000	233	634	0.703	0.218738E−10	0.975859E−05	0.106368E−04
explin	12	90	253	0.000	−0.684995E+04	0.842454E−05	0.870068E−05
	120	118	326	0.016	−0.723756E+06	0.842447E−05	0.106771E−04
	1200	542	2000	0.516	−0.719244E+08	0.398635E−02	0.474291E−02
explin2	12	65	191	0.000	−0.709247E+04	0.839495E−05	0.839495E−05
	120	742	2000	0.109	−0.724459E+06	0.340264E−04	0.340264E−04
	1200	542	2001	0.547	−0.719988E+08	0.512678E−03	0.514950E−03
mccormck	100	47	90	0.000	−0.917881E+02	0.899848E−05	0.101388E−04
	500	55	106	0.016	−0.457077E+03	0.949250E−05	0.102015E−04
	1000	61	119	0.047	−0.913689E+03	0.863085E−05	0.927314E−05
probpenl	1000	104	729	0.359	0.199800E−06	0.832719E−05	0.263323E−03
	5000	2	15	0.031	0.399920E−07	0.159874E−07	0.284578E−06
qrtquad	12	110	2000	0.016	−0.112839E+04	0.856392E+02	0.157287E+03
	120	525	2000	0.094	−0.364807E+07	0.213147E−02	0.242191E−02
	1200	428	2000	0.484	−0.337453E+10	0.386732E+05	0.750081E+05
bdexp	5000	1001	1002	8.078	0.571006E−01	0.152184E−03	0.792156E−02
hatfldc	25	53	105	0.000	0.319488E−10	0.880218E−05	0.263278E−04
	1000	46	92	0.031	0.265908E−08	0.871123E−05	0.103529E−03
	5000	46	92	0.172	0.133791E−07	0.871123E−05	0.231514E−03
hs110	50	1	2	0.000	−0.999000E+10	0.000000E+00	0.000000E+00
	100	1	2	0.000	−0.998002E+20	0.000000E+00	0.000000E+00
biggsb1	1000	1001	1688	0.609	0.196231E−01	0.468674E−02	0.660522E−02
	5000	1001	1688	2.609	0.196231E−01	0.468674E−02	0.660522E−02
hatflda	1000	2	2000	4.750	0.721579E+01	0.100000E+21	0.100000E+21
edensch	2000	100	273	0.172	0.120033E+05	0.900582E−05	0.182303E−04
edensch	2000	0	1	0.000	0.339990E+05	0.000000E+00	0.000000E+00
edensch	2000	241	2000	0.688	0.137024E+05	0.113263E−03	0.235224E−03
edensch	2000	233	2000	0.641	0.120046E+05	0.122858E−03	0.199621E−03
edensch	2000	479	2000	0.875	0.144263E+05	0.208728E−04	0.408671E−04
penalty1	2000	1001	1059	0.906	0.198206E−01	0.556919E−04	0.128783E−02
penalty1	2000	1001	1101	0.922	0.196674E−01	0.174975E−03	0.619743E−02
penalty1	2000	40	147	0.062	0.411068E+02	0.845885E−05	0.178404E−03
penalty1	2000	32	148	0.062	0.950806E+02	0.832141E−05	0.152028E−03
cvbqp1	1000	1	2	0.000	0.225225E+05	0.000000E+00	0.000000E+00
nbcvbqp1	1000	1	2	0.000	−0.198680E+09	0.000000E+00	0.000000E+00
nbcvbqp2	1000	255	2000	0.500	−0.133388E+09	0.502444E−03	0.531423E−03
nbcvbqp3	1000	208	2000	0.453	−0.657738E+08	0.470544E−03	0.559064E−03
qudlin	1000	1	2	0.000	−0.500000E+08	0.000000E+00	0.000000E+00
	5000	1	2	0.000	−0.125000E+10	0.000000E+00	0.000000E+00
s368	100	106	2000	1.000	0.105413E+10	0.970297E+00	0.570547E+01
harkerp2	100	84	2000	0.547	−0.959748E−06	0.999592E+00	0.999592E+01
pentdi	1000	16	32	0.000	−0.375000E+00	0.768000E−05	0.809543E−05
	5000	16	32	0.063	−0.375000E+00	0.768000E−05	0.809543E−05
chenhark	1000	1001	1846	0.641	−0.998423E+00	0.481583E−03	0.246523E−02
sineali	1000	2	5	0.000	−0.976805E+05	0.000000E+00	0.000000E+00

Problem: name of the problem; dim: dimension of the problem; iter: number of iterations; fcnt: number of function evaluations; time: CPU time in seconds; $f(x)$: final function value; pginfn: infinity norm of the final gradient projection; pgtwon: norm of the final gradient projection.

4. Numerical experiments

In this section we test the numerical behaviour of Algorithm 2.1 (MSLBFGS), and compare with the subspace limited memory quasi-Newton algorithm (called NIYUAN here) in [24]. For all test problems, we stop the iteration if

$$\|P_\Omega(x^k - g^k) - x^k\| \le 10^{-5} \quad \text{or} \quad \|P_\Omega(x^k - g^k) - x^k\|_\infty \le 10^{-5} \tag{4.1}$$

Table 4.3
Performance of L-BFGS-B

problem	dim	iter	fcnt	time	$f(x)$	pginfn
nonscomp	1000	28	32	0.031	0.164332E−08	0.218253E−03
	2000	36	41	0.078	0.108606E−08	0.156895E−03
	5000	34	38	0.188	0.806444E−09	0.472007E−04
mccormck	100	683	777	0.172	−0.500187E+13	0.532978E+02
probpenl	1000	2	4	0.016	0.199800E−06	0.166319E−06
	5000	2	4	0.016	0.399920E−07	0.310914E−06
qrtquad	12	5	87	0.000	−0.119530E+04	0.663373E+02
	120	5	49	0.016	−0.291560E+07	0.448379E+04
	1200	1	21	0.000	0.000000E+00	0.119900E+05
bdexp	5000	16	18	0.234	0.196787E−02	0.552114E−05
hatfldc	25	18	22	0.000	0.550329E−09	0.371552E−04
	1000	20	24	0.016	0.819539E−09	0.528848E−04
	5000	21	26	0.125	0.193954E−09	0.253317E−04
biggsb1	1000	1398	1450	1.438	0.486819E−06	0.915080E−05
biggsb1	5000	3772	3897	20.172	0.126528E−03	0.831051E−05
edensch	2000	13	16	0.031	0.120033E+05	0.386546E−02
edensch	2000	0	1	0.000	0.339990E+05	0.000000E+00
edensch	2000	13	16	0.047	0.120033E+05	0.386546E−02
edensch	2000	13	16	0.016	0.120033E+05	0.386546E−02
edensch	2000	65	90	0.188	0.120033E+05	0.451844E−02
penalty1	2000	61	71	0.141	0.195552E−01	0.278097E−05
penalty1	2000	61	71	0.125	0.195552E−01	0.278097E−05
penalty1	2000	61	71	0.172	0.195552E−01	0.278097E−05
penalty1	2000	61	71	0.172	0.195552E−01	0.278097E−05
cvbqp1	1000	625	640	0.703	0.629610E−06	0.735262E−02
bcvbqp1	1000	1	21	0.016	−0.492469E+06	0.525000E+04
bcvbqp2	1000	1	21	0.000	−0.281250E+06	0.375000E+04
bcvbqp3	1000	1	21	0.016	0.705938E+05	0.375000E+04
qudlin	1000	21	268	0.047	$-\infty$	0.424033+155
s368	100	1	15	0.016	0.105413E+10	0.221993E+02
harkerp2	100	3	42	0.266	0.437139E+11	0.658443E+02
pentdi	1000	17	37	0.016	−0.410122E+02	0.644267E−03
	5000	17	37	0.125	−0.207679E+03	0.982725E−03
sineali	1000	2	50	0.031	−0.998754E+05	0.407471E+03

is satisfied. The iteration is also stopped if the number of iterations exceed 1000, or the number of function evaluations reached 2000. Both the codes are written in Fortran77 and in double precision arithmetic. All runs are perform on a PC with CPU P4 2.6 GHz and 256 M memory. Our experiments are performed on a set of the nonlinear bound constrained problems from CUTE (see [11]) collection that have second derivatives available, all the selected problems both have upper and lower bound. Different bounds of problems `penalty1` and `edensch` are designed in [6].

Both in algorithm MSLBFGS and NIYUAN, we choose $\delta = 10^{-4}$ in projected line search, and choose $\epsilon_b = \frac{1}{1000}\min\{u_i - l_i | i = 1, \ldots, n\}$, this choice is very small and can approximate the active set at initialization. In the limited memory BFGS update, we set the "basic matrix" to be the identity matrix I. The number of corrections pairs used in limited memory BFGS method is $m = 5$.

We note that, if the initial data is far from a local minimizer, the active set can change with each iteration, and the calculations involved H^k must be designed to account for this. One efficient way to do this is to store the sequence $\{\bar{s}^k\}$ and $\{\bar{y}^k\}$ according to the reduced gradient and projected steps at each iteration, and use a recursive algorithm to determine the search direction in free subspace. Let $p \leq m$ be the indices of the correction pairs, initialize $\bar{d} = -g^k$. An updating process that implements these savings in computation is as follows:

Subalgorithm 4.1. **update**(**p**, $\{\bar{s}^k\}$, $\{\bar{y}^k\}$, I, $\bar{d}$, Z)
step 1. $\bar{d} = Z^T\bar{d}$.
step 2. if $p = 0$, $\bar{d} = I\bar{d}$, return.
step 3. $\alpha = \frac{\bar{s}_{p-1}^T\bar{d}}{\bar{y}_{p-1}^T\bar{s}_{p-1}^k}$, $\bar{d} = \bar{d} - \alpha\bar{y}_{p-1}^k$.

step 4. call update($p-1$, $\{\bar{s}^k\}$, $\{\bar{y}^k\}$, I, $\bar{d}$, Z),

step 5. $\bar{d} = \bar{d} + (\alpha - \frac{\bar{s}_{p-1}^T \bar{d}}{\bar{y}_{p-1}^T \bar{s}_{p-1}^k})\bar{s}_{p-1}^k$,

step 6. $\bar{d} = Z^T \bar{d}$.

The above recursive subalgorithm comes from [18], and it is used to compute the direction (2.8) of variables with indices in $B^k \cup A_2^k$. The output of the subalgorithm is an $n \times 1$ vector $\bar{d}$ with $\bar{d}_i = 0$ for all $i \notin A_2^k \cup B^k$. In our codes, we reinitialize p to zero when $(\bar{y}^k)^T \bar{s}^k \leq 0$ or $p = m$ is met, that is to say, a steepest descent direction is used in this situation.

The numerical results of the algorithms MSLBFGS and NIYUAN are listed in Tables 4.1 and 4.2, respectively. The columns have the following meanings:

From the above tables, we observe that MSLBFGS method performed a little better than NIYUAN method. In most cases, its behaviour in the problems where it works quite efficiently, which require fewer iterations, fewer function evaluations and less time consuming. Preliminary numerical results also indicate the modifications are beneficial to the performance of the algorithm.

In order to asses the reliability algorithm MSLBFGS, we tested this method against a well-known routine L-BFGS-B (see [6,26]). The fortran77 codes of L-BFGS-B are contributed by J. Nocedal, which are available at
http://www.ece.northwestern.edu/~nocedal/software.html.

In running the codes, default values are used for all parameters. Table 4.3 reports the numerical results for this method.

We see from Table 4.3 that L-BFGS-B fails to solve the problems `explin, explin2, hs110, hatflda`, and `chenark`, for the reason can be found in [26]. We also note an interesting fact that L-BFGS-B is sometimes stopped before the projected gradient sufficient to satisfy the stopping condition, and the reason should be that the codes could make no further progress in reducing the value of f (see [26]). In addition, L-BFGS-B method seemly perform better than MSLBFGS and NIYUAN did, especially on problems `nonscomp, probpenl, hatfldc, edensch` and `penalty1`. Moreover, preliminary experimental comparisons also indicate algorithm MSLBFGS is competitive with some well-known methods.

Acknowledgements

We thank Professor Dong-Hui Li for his careful reading of the manuscript and for catching several typos. We are also very grateful to two anonymous referees for their useful suggestions and comments on the previous version of this paper. This work was supported by Chinese NSF grant 10471036.

References

[1] R. Andreani, E.G. Birgin, J.M. Martínez, M.L. Schuverdt, Augmented Lagrangian methods under the constant positive linear dependence constraint qualification, Math. Program. 111 (2008) 5–32.
[2] R. Andreani, E.G. Birgin, J.M. Martínez, M.L. Schuverdt, On augmented Lagrangian methods with general lower-level constraints, SIAM J. Optim. 18 (2007) 1286–1309.
[3] D.P. Bertsekas, Projected Newton methods for optimization problems with simple constrains, SIAM J. Control Optim. 20 (1982) 221–246.
[4] E.G. Birgin, J.M. Martínez, Large-scale active-set box-constrained optimization method with spectral projected gradients, Comput. Optim. Appl. 23 (2002) 101–125.
[5] J.V. Burke, J.J. Moré, On the identification of active constraints, SIAM J. Numer. Anal. 25 (1988) 1197–1211.
[6] R.H. Byrd, P.H. Lu, J. Nocedal, A limited memory algorithm for bound constrained optimization, SIAM J. Sci. Stat. Comput. 16 (1995) 1190–1208.
[7] R.H. Byrd, J. Nocedal, R.B. Schnabel, Representations of quasi-Newton matrices and their use in limited memory methods, Math. Program. 63 (1994) 129–156.
[8] P. Calamai, J.J. Moré, Projected gradient for linearly constrained programs, Math. Program. 39 (1987) 93–116.
[9] A.R. Conn, N.I.M. Gould, Ph.L. Toint, Global convergence of a class of trust region algorithm for optimization with simple bounds, SIAM J. Numer. Anal. 25 (1988) 433–460.
[10] A.R. Conn, N.I.M. Gould, Ph.L. Toint, A globally convergent augmented Lagrangean algorithm for optimization with general constraints and simple bounds, SIAM. J. Numer. Anal. 28 (1991) 545–472.
[11] A.R. Conn, N.I.M. Gould, Ph.L. Toint, CUTE: Constrained and unconstrained testing environment, ACM Trans. Math. Software 21 (1995) 123–160.
[12] Y.H. Dai, R. Fletcher, Projected Barzilai–Borwein methods for large-scale box constrained quadratic programming, Numer. Math. 100 (2005) 21–47.

[13] M.A. Diniz-Ehrhardt, M.A. Gomes-Ruggiero, J.M. Martínez, S.A. Santos, Augmented Lagrangian algorithms based on the spectral projected gradient for solving nonlinear programming problems, J. Optim. Theory Appl. 123 (2004) 497–517.
[14] F. Facchinei, J. Júdice, J. Soares, An active set Newton slgorithm for large-scale nonlinear programs with box canstranits, SIAM J. Optim. 8 (1998) 158–186.
[15] F. Facchinei, S. Lucidi, L. Palagi, A truncated Newton algorithm for large scale box constrained optimization, SIAM J. Optim. 12 (2002) 1100–1125.
[16] N.I.M. Gould, D. Orban, Ph.L. Toint, Numerical methods for large-scale nonlinear optimization, Acta Numer. 14 (2005) 299–361.
[17] W.W. Hager, H. Zhang, A new active set algorithm for box constrained optimizaiton, SIAM J. Optim. 17 (2006) 526–557.
[18] C.T. Kelley, Iterative Methods for Optimization, Philadelphia, Pa., 1999.
[19] C.J. Lin, J.J. Moré, Nowton's method for large bound-constrained optimization problems, SIAM J. Optim. 9 (1999) 1100–1127.
[20] M. Lescrenier, Convergence of trust region algorithm for optimization with bounds when strict complementarity does not hold, SIAM J. Numer. Anal. 28 (1991) 467–695.
[21] D.G. Luenberger, Introduction to Linear and Nonlinear Programming, Addison-Wesley, Reading, MA, 1973 (Chapter 11).
[22] J.M. Martínez, BOX-QUACAN and the implementation of Augmented Lagrangian algorithms for minimization with inequality constraints, Comput. Appl. Math. 19 (2000) 31–56.
[23] Q. Ni, A subspace projected conjuagte gradient algorithm for large bound constrained quadratic programming, Numer. Math. (A Journal of Chinese Universities) 7 (1998) 51–60.
[24] Q. Ni, Y.X. Yuan, A subspace limited memory quasi-Newton algorithm for large-scale nonlinear bound constrained optimization, Math. Comput. 66 (1997) 1509–1520.
[25] M.J.D. Powell, A fast algorithm for nonlinearly constrained optimization calculations, Numer. Anal. (1978) 155–157.
[26] C.Y. Zhu, R.H. Byrd, P.H. Lu, J. Nocedal, L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization, ACM Trans. Math. Software 23 (1997) 550–560.