A limited-memory multipoint symmetric secant method for bound constrained optimization

Oleg P. Burdakov *, José Mario Martínez † Elvio A. Pilotta ‡

October 30, 2001

Abstract

A new algorithm for solving smooth large-scale minimization problems with bound constraints is introduced. The way of dealing with active constraints is similar to the one used in some recently introduced quadratic solvers. A limited-memory version of the multipoint symmetric secant method is presented. This version is used for approximating the Hessian, which does not need not to be positive definite. Positive definiteness of the approximation is not enforced. Instead, negative curvature information is accumulated in the approximate Hessian. A combination of trust-region and conjugate-gradient approaches is used to explore useful information. Global convergence is proved for a general model algorithm. Results of numerical experiments are presented.

Keywords: large-scale optimization, box constraints, active set methods, gradient projection, trust region, conjugate gradients, multipoint symmetric secant methods, global convergence.

*Division of Optimization, Department of Mathematics, Linköping University, S - 581 83 Linköping, Sweden. Part of the work of this author was done while he was visiting the University of Campinas under support of FAPESP (Grant 1197-11730-5). E-Mail: olbr@mai.liu.se

†Department of Applied Mathematics, IMECC-UNICAMP, University of Campinas, CP 6065, 13081-970 Campinas SP, Brazil. This author was supported by PRONEX-Optimization, FAPESP (Grant 99-372-4-6), CNPq and FAEP-UNICAMP. E-Mail: martinez@ime.unicamp.br

‡Facultad de Matemática, Astronomía y Física, FaMAF-CIEM, Universidad Nacional de Córdoba, Ciudad Universitaria (5000), Argentina. This author was supported by PRONEX-Optimization, FAPESP (Grant 90-372-4-6), CNPq, FAEP-UNICAMP and SECyT-UNC (Grant 194-2000). E-Mail: pilotta@mate.uncor.edu
1 Introduction

We consider the following bound (or box-) constrained optimization problem

\[
\text{Minimize } f(x) \text{ subject to } x \in \Omega, \quad (1)
\]

where the function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is continuously differentiable and the box-type feasible region \( \Omega \) is given by the lower and upper bounds \( \ell, u \in \mathbb{R}^n, \ell < u \), as follows

\[
\Omega = \{ x \in \mathbb{R}^n \mid \ell \leq x \leq u \}.
\]

This problem is very important in practical optimization. On one hand, a lot of applied problems admit mathematical models of type (1). On the other hand, one of the most effective approaches for solving general constrained optimization problems, based on augmented Lagrangians, relies on effective algorithms for solving (1) (see [16, 17, 19, 38]). Finally, in recent works on complementarity and variational inequalities, these problems are reduced to bound constrained minimization problems in an efficient way (see [1, 2, 3, 4] and references therein).

All practical methods for solving (1) are iterative. Given \( x^k \in \Omega \), many methods construct a quadratic model of \( f \), whose gradient at \( x^k \) coincides with the gradient of \( f \), and whose Hessian is an approximation of the Hessian of \( f \). Many different (but related) ways of using this approximation were considered in recent publications. See [13, 16, 30, 34, 37].

In the algorithms introduced in this paper we also use quadratic models, but the way of treating constraints differs from the ones described in [13, 16, 30, 34, 37, 41]. Roughly speaking, our proposal is to treat constraints in the same way the quadratic solvers [5, 23, 33] do. This means that an algorithm for unconstrained minimization on the current face is used, until a separate indicator says that this is not worthwhile anymore. In this case, the face is abandoned along a direction defined in [31, 32, 33] for convex minimization. For this direction, interesting physical interpretations are given in [23]. See also [24, 25, 26, 27, 28, 29]. Moreover, when in the unconstrained search process within a face, the algorithm hits a bound, several new constraints are added to those to be satisfied by the trial point. This allows to avoid the costly process of adding just one constraint per iteration.

Existing algorithms use different ways of constructing Hessians for the quadratic models. The true Hessian of \( f \) as well as limited-memory BFGS and SR1 quasi-Newton approximations are the best known alternatives (see [13, 16]). An interesting Gauss-Newton-type approximation of the Hessian of augmented Lagrangians was considered in [38]. The cases in which the true Hessian is very
costly or difficult to compute and its finite-difference approximation is also time-consuming are not rare in practice. In these cases, it is possible to use the truncated-Newton approach [6, 21], where each “Hessian \times vector” product is replaced by an incremental quotient, but since each of these products involves an additional gradient evaluation, this alternative can also be inefficient. Moreover, in this approach, information about the Hessian matrix obtained at the current iteration is not used on the next one.

On the other hand, the quasi-Newton approximations of the Hessian (see, e.g., [22]) are able to accumulate such kind of information. These approximations involve only one gradient evaluation per iteration, but the true function does not fit with the resulting quadratic model as well as with the true-Hessian models do. In the large-scale case, the conventional quasi-Newton methods, which do not preserve sparsity, generate dense Hessian approximations that cannot be stored explicitly or, if they can be stored, algebraic manipulations with them are prohibitively expensive. To overcome these difficulties, limited-memory alternatives have been developed (see [13, 14, 35] and references therein).

Our limited-memory approach will be based on the multipoint symmetric secant approximations of the Hessian matrix proposed in [9]. These approximations are an extension of the classical multipoint secant scheme (see [39, 42] and references therein) with the advantage that they exploit the symmetry of the Hessian matrix in a natural way. They differ from those introduced in [44]. The idea is that the Hessian approximation should be such that the gradient of the quadratic model would interpolate the gradient of \( f \) at some previous points. However, since this objective conflicts with the symmetry, the most “fresh” information carried by the gradient values is privileged in releasing partially the interpolation requirement. The tendency to instability of the sequential secant methods is overcome with the approach developed in [10, 11, 12]. Since the multipoint symmetric secant schemes are able to accumulate negative curvature information and to generate better Hessian approximations than (say) BFGS, they fit well conservative procedures of treating active constraints. Moreover, a combination of the box-trust-region (BTR) and conjugate gradient (CG) approaches incorporated in our algorithm allows us to take advantage of negative curvature information.

The organization of this paper is as follows. An algorithmic outline of the main algorithm is described in Section 2, where basic global convergence theorems are also proved. In Section 3 we present a sub-algorithm that can be used by the main algorithm for minimization within the current face. The sub-algorithm is based on the BTR and CG approaches, and it assumes that a Hessian approximation is given. A limited-memory multipoint symmetric secant approximation is introduced in Section 4. In Section 5 we discuss some implementation details.
The numerical results are presented in Section 6. Finally, the conclusions are given in Section 7.

2 Main algorithmic model and global convergence

The main algorithm presented in this paper is an active-set method with a special procedure for dropping constraints. It calls a Sub-algorithm for the minimization on the current face. The algorithm visits the different faces of the box using a strategy that will be described below. First we need some general definitions.

As in [33], let us divide the feasible set $\Omega$ into disjoint faces, as follows. For all $I \subset \{1,2,\ldots,n,n+1,n+2,\ldots,2n\}$, we define

$$F_I = \{x \in \Omega \mid x_i = \ell_i \text{ if } i \in I, \ x_i = u_i \text{ if } n+i \in I, \ \ell_i < x_i < u_i \text{ otherwise}\}.$$  

The closure of $F_I$ is denoted by $\bar{F}_I$. Let $[F_I]$ denote the smallest linear manifold that contains $F_I$, and $S_I$ denote the subspace obtained by the parallel translation of $[F_I]$. For brevity, $-\nabla f(x)$ will be called \textit{antigradient}.

Assuming that $x \in F_I$, the orthogonal projection of $-\nabla f(x)$ on $S_I$ will be called \textit{internal antigradient} and denoted by $g_I(x)$. The \textit{chopped antigradient} (see [23, 33]) $g_C(x)$ is defined for $x \in \Omega$ as follows

$$[g_C(x)]_i = \begin{cases} -\frac{\partial f_I}{\partial x_i}(x), & \text{if } x_i = \ell_i \text{ and } \frac{\partial f_I}{\partial x_i}(x) < 0, \\ -\frac{\partial f_I}{\partial x_i}(x), & \text{if } x_i = u_i \text{ and } \frac{\partial f_I}{\partial x_i}(x) > 0, \\ 0, & \text{otherwise}, \end{cases}$$

where $i = 1,\ldots,n$. Here “otherwise” includes, in particular, the case, when $\ell_i < x_i < u_i$.

Since $g_C(x) \perp S_I$, we observe that

$$g_I(x) \perp g_C(x).$$

Denote $g_P(x) = g_I(x) + g_C(x)$. We call the vector $g_P(x)$ \textit{projected antigradient}. Note that $x \in \Omega$ is a stationary point of problem (1) if and only if $g_P(x) = 0$.

In general, the mapping $g_P(x)$ is not continuous, nevertheless, if $x^k \to x$ and $g_P(x^k) \to 0$, this implies that $g_P(x) = 0$ (see [15]).

Given $x^k$, the Sub-algorithm returns $x^{k+1}$, and it is assumed to have the following properties.

\begin{itemize}
\item \textbf{P1.} $f(x^{k+1}) < f(x^k)$.
\item \textbf{P2.} If $x^k \in F_I$, then $x^{k+1} \in F_I$.
\item \textbf{P3.} If $\{x^k, x^{k+1}, x^{k+2}, \ldots\} \subset F_I$ is a set of infinitely many iterates generated all by the Sub-algorithm, then $g_I(x^k) \to 0$.
\end{itemize}
Below we present our main model algorithm. The symbol $\| \cdot \|$ will denote the Euclidean vector norm throughout the paper, although in many cases, any other norm can be used instead.

**Algorithm 2.1.** Assume that $x^0 \in \Omega$ is an arbitrary initial point, $\eta \in (0, 1)$, $0 < \tau_{\min} \leq \tau_{\max} < \infty$, $0 < \beta_{\min} \leq \beta_{\max} < 1$ and $\theta \in (0, 1)$. Let $F_i$ be the face that contains the current iterate $x^k$. The new iterate $x^{k+1}$ is computed as follows.

**Step 1.** Stop if $\|g_P(x^k)\| = 0$ ($x^k$ is a stationary point). If

$$\frac{\|g_C(x^k)\|}{\|g_P(x^k)\|} \geq \eta,$$  \hspace{1cm} (2)

then compute $x^{k+1}$ at Step 2, else compute $x^{k+1}$ using the Sub-algorithm.

**Step 2.** Choose $\tau_k \in [\tau_{\min}, \tau_{\max}]$. Let $\alpha_{\max}$ be the maximum value of $\alpha$, for which $x^k + \alpha g_C(x^k) \in \Omega$. Set $\alpha = \min\{\tau_k, \alpha_{\max}\}$. If

$$f(x^k + \alpha g_C(x^k)) \leq f(x^k) - \theta \alpha \|g_C(x^k)\|^2$$  \hspace{1cm} (3)

set $\alpha_k = \alpha$, $x^{k+1} = x^k + \alpha k g_C(x^k)$ and finish the $k$-th iteration. Else, choose a new value for $\alpha$ from the interval $[\beta_{\min} \alpha, \beta_{\max} \alpha]$ and repeat test (3).

The global convergence theory for this algorithm generalizes the one given in [5] for quadratic minimization.

We assume that $\nabla f(x)$ satisfies the Lipschitz condition: there exists $L > 0$ such that

$$\|\nabla f(y) - \nabla f(x)\| \leq L\|y - x\|, \hspace{1cm} \forall x, y \in \Omega.$$  

This implies that

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2, \hspace{1cm} \forall x, y \in \Omega.$$  \hspace{1cm} (4)

Here and below, $\langle a, b \rangle$ denotes the scalar product $a^T b$ in $\mathbb{R}^n$.

Let us prove global convergence for Algorithm 2.1.

**Theorem 2.1.** Algorithm 2.1 is well defined, and every limit point of its iterates is a stationary point for problem (1).

**Proof.** Denote $K = \{k \in \mathbb{N} \mid \|g_C(x^k)\|/\|g_P(x^k)\| \geq \eta\}$.

To prove that the algorithm is well defined, it is sufficient to show that, for all $k \in K$, condition (3) is satisfied after a finite number of reductions of $\alpha$. Indeed,
for all $\alpha \geq 0$, from (4) we have

$$f(x^k + \alpha g_C(x^k)) \leq f(x^k) - \alpha \|g_C(x^k)\|^2 + \frac{\alpha^2 L}{2} \|g_C(x^k)\|^2.$$ 

This implies that (3) holds for $\alpha \leq \frac{2(1-\theta)}{L}$. Therefore, the new iterate is well defined.

Moreover, the value of $\alpha$ accepted at Step 2 of Algorithm 2.1 is bounded below by

$$\alpha = \min \{r_{\min}, \frac{2(1-\theta)}{L} \beta_{\min} \} > 0.$$ 

Hence, at Step 2 we have

$$f(x^k) - f(x^{k+1}) \geq \theta \alpha \|g_P(x^k)\|^2.$$ 

(5)

Since $f(x^{k+1}) \leq f(x^k)$ for all $k \in \mathbb{N}$ and since $f(x)$ is bounded below on $\Omega$, (5) implies that either $K$ is finite or

$$\sum_{k \in K} \|g_P(x^k)\|^2 < \infty.$$ 

(6)

In the infinite case, (6) implies that $g_P(x^k) \to 0$ for $k \in K$. Consequently, every limit point of $\{x^k\}_{k \in K}$ is a stationary point.

If $K$ is finite, there exists $k_0 \in \mathbb{N}$ and a face $F_I$ such that $x^k \in F_I$ for all $k \geq k_0$. Therefore, $x^{k+1}$ is computed by the Sub-algorithm for all $k \geq k_0$. Then, by the property P3, $\lim_{k \to \infty} \|g_I(x^k)\| = 0$. But, for all $k \geq k_0$, inequality (2) does not hold. Hence $\lim_{k \to \infty} \|g_P(x^k)\| = 0$. As before, this means that every limit point of $\{x^k\}$ is stationary.

Recall that the stationary points of our problem are characterized by $g_P(x) = 0$. If $x$ is a stationary point, such that $x_i = \ell_i$ (or $x_i = u_i$) and $\frac{\partial f}{\partial x_i} = 0$, we call this point degenerate. In the following theorem we prove that, if degenerate points do not exist, the algorithm identifies the active constraints at the limit points in a finite number of iterations.

**Theorem 2.2.** Assume that all the stationary points of (1) are nondegenerate. Then, there exists $I \subset \{1, 2, \ldots, 2n\}$ and there exists $k_0 \in \mathbb{N}$ such that $x^k \in F_I$ for all $k \geq k_0$. Moreover, all the limit points of the sequence $\{x^k\}$ belong to $F_I$.

**Proof.** We exclude from our consideration the trivial case, when Algorithm 2.1 terminates in a finite number of iterations. Let us prove first that Step 2 cannot be executed infinitely many times. Assume, by contradiction, that the iterate $x^{k+1}$ is computed at Step 2 at infinitely many iterations. Then there exists a constraint
which is abandoned infinitely many times. Without loss of generality, assume that this constraint is \( x_i = \ell_i \), i.e., there exists an infinite set \( K \) of the iteration numbers \( k \) such that

\[
x_i^k = \ell_i, \quad x_i^{k+1} > \ell_i,
\]

and \( x^{k+1} \) is computed by Step 2. Let \( x^* \) be a limit point of \( \{ x^k \}_{k \in K} \). By Theorem 2.1, \( x^* \) is a stationary point. From (7) and (8), we have \( x_i^* = \ell_i \) and \( \frac{\partial f}{\partial x_i}(x^*) \leq 0 \). But since \( x^* \) is stationary, \( \frac{\partial f}{\partial x_i}(x^*) \geq 0 \). Hence \( x^* \) is degenerate, which contradicts the theorem assumption. Thus, we proved that there exists \( k_0 \in \mathbb{N} \) and \( I \subset \{1, 2, \ldots, 2n\} \) such that \( x^k \in F_I \) for all \( k \geq k_0 \). This implies that \( x^{k+1} \) is computed by the Sub-algorithm for all \( k \geq k_0 \). Then, according to the property P3, \( g_I(x^k) \to 0 \). By continuity, this gives

\[
\frac{\partial f}{\partial x_i}(x^*) = 0
\]

for all \( i \) such that \( i \notin I \) and \( n+i \notin I \). Since \( x^* \) is nondegenerate, this implies that \( \ell_i < x_i^* < u_i \). Therefore, \( x^* \in F_I \).

3 Minimization within a given face

Algorithm 3.1, which is presented below, is one of the possible implementations of the Sub-algorithm, which is used at Step 1 of Algorithm 2.1 for the minimization within a given face \( F_I \). Given \( x^k \in F_I \) (that violates (2)), a symmetric Hessian approximation \( B^k \in \mathbb{R}^{n \times n} \) and a trust region radius \( \delta_k \), Algorithm 3.1 generates \( x^{k+1} \in F_I \). Recall that the face \( F_I \) changes in Algorithm 2.1, when \( x^{k+1} \in F_I - F_I \).

To simplify the notation, suppose that the face \( F_I \) is the interior of \( \Omega \). The extension to a general \( F_I \) is straightforward. In Algorithm 3.1, the CG method is applied to the quadratic subproblem

\[
\text{Minimize } Q(p) \equiv \frac{1}{2} \langle p, B^k p \rangle + \langle \nabla f(x^k), p \rangle.
\]

Like in [45], the regular CG iterations are interrupted, when the constraints

\[
\|p\|_\infty \leq \delta_k, \quad \ell \leq x^k + p \leq u.
\]

are violated or when a direction of negative curvature is encountered in \( B^k \). A TR approach is used to decide whether the generated trial point is good enough. We also follow the classical approach to modify the trust-region radius.

(7)
Our Sub-algorithm can be stated formally as follows.

**Algorithm 3.1.**

**Step 1.** Starting with $p^0 = 0$, apply to (9) the CG method which is supposed to generate at its $j$th iteration ($j = 0, 1, \ldots$) a search direction $d^j$ and a new iterate $p^{j+1}$. Interrupt this process in any of the following three cases:

Case 1: $\nabla Q(p^j) = 0$. In this case, set $p_{\text{trial}} = p^j$.

Case 2: $Q(p^j)$ tends to $-\infty$ along $d^j$. In this case, proceed as in Case 3, with $p^{j+1}$ replaced by $p^j + M d^j$, where $M$ is a large positive number.

Case 3: $p^{j+1}$ violates at least one of the constraints (10). In this case, define $p'$ as the projection of $p^{j+1}$ on the region given by (10), and define $p''$ as the farthest point from $p^j$ among those belonging to the segment $[p^j, p^{j+1}]$ satisfying (10). If $Q(p') \leq Q(p'')$, then set $p_{\text{trial}} = p'$, else set $p_{\text{trial}} = p''$.

**Step 2.** Set $x_{\text{trial}} = x^k + p_{\text{trial}}$, and compute the following predicted and actual reductions in function value:

$$\Delta_{\text{pred}} = -Q(p_{\text{trial}}), \quad \Delta_{\text{act}} = f(x^k) - f(x_{\text{trial}}).$$

If $\Delta_{\text{act}} < 0.1 \Delta_{\text{pred}}$, set $\delta_k = 0.5 \|p_{\text{trial}}\|_\infty$ and go to Step 1.

If $0.1 \Delta_{\text{pred}} \leq \Delta_{\text{act}} \leq \Delta_{\text{pred}}$, set $x^{k+1} = x_{\text{trial}}$ and $\delta_{k+1} = \delta_k$.

If $\Delta_{\text{act}} > \Delta_{\text{pred}}$, set $x^{k+1} = x_{\text{trial}}$ and $\delta_{k+1} = 3 \delta_k$.

Note that the factors 0.1 and 0.5 used in Step 2 are actually specific values of the two parameters that can have any values on the interval (0, 1). The third parameter presented by the factor 3 can have any value greater than one. Our specific choice of the parameter values was justified by numerical experience.

The existing theoretical results concerning the trust-region methods (see e.g. [16, 19]) can be applied to show that, under the boundedness assumptions on $\|B^k\|$, Algorithm 3.1 is well defined and enjoys the properties P1-P3.

The Hessian approximation $B^k$, that will be introduced in the next section, is a limited-memory approximation. Since this approximation is a low-rank modification of a multiple of the identity matrix, the matrix $B^k$ has a small number of distinct eigenvalues. This results in a small number of iterations required by the CG method to solve the quadratic problem (9) or to identify a negative curvature direction in $Q(p)$. The negative curvature information is accumulated in the Hessian approximations $B^k$ due to the multipoint symmetric secant approach applied for generating $B^k$. 

8
4 Limited-memory multipoint symmetric secant approximations

Let us describe now how to employ the basic idea of the multipoint symmetric secant method for generating the matrices $B_k$.

Denote $s^k = x^{k+1} - x^k$, $y^k = g^{k+1} - g^k$. Consider a special case assuming that the sequence of $n$ vectors $s^0, \ldots, s^{n-1}$ have been generated somehow, and that they are linearly independent. The ideal aim would be to construct a Hessian approximation $B^n \in \mathbb{R}^{n \times n}$ such that

\[(B^n)^T = B^n, \quad \text{(11)}\]
\[B^n s^i = y^i, \quad i = 0, \ldots, n - 1. \quad \text{(12)}\]

However, in general, this is impossible, because the system of $n(n-1)/2 + n^2$ equations (11) and (12) in $n^2$ unknown elements of $B^n$ is overdetermined. The information about the symmetry of the Hessian matrix conflicts here with the information carried by the pairs \{s^i, y^i\}. The idea of the sequential symmetric secant methods introduced in [9] is to release partially relations (12) in order to have $B^n$ well defined. This can be accomplished in various ways. The uniqueness can be achieved by ranging the pairs \{s^i, y^i\} in accordance with the reliability of the information that they carry. For example, for $i > j$, one can consider \{s^j, y^i\} as more reliable for the Hessian approximation than \{s^i, y^j\}, because the first pair was computed at a more recent iteration, and therefore it carries more fresh information. Then, in the process of constructing the Hessian approximation $B^n$, it is natural to use the pairs \{s^i, y^i\} sequentially for $i = n - 1, n - 2, \ldots, 0$. For any current \{s^j, y^j\}, we can ignore the part of its information, which conflicts with the more fresh information \{s^j, y^j\}, $j = n - 1, \ldots, i - 1$, that have already been used.

To clarify this idea, suppose for the moment, that each vector $s^{n-i}$, $i = 1, \ldots, n$, is parallel to the coordinate axis $e^i$. Then the first column and the first row of the Hessian matrix can be approximated by the standard finite-difference formula as $y^{n-1}/\|s^{n-1}\|$. The second column and row, in their parts outside the first column and row, are approximated by $y^{n-2}/\|s^{n-2}\|$, and so on. To fill in the nonfilled part of the $i$th row and column, the components $y^j_i/\|s^{n-i}\|$, $j = i, \ldots, n$, are used (see Fig. 1).

In the general case of arbitrary vectors $s^{n-i}$, the space can be linearly transformed so that, in the new space, the vectors $s^{n-i}$ are parallel to the new coordinate axes $e^i$. Then the described approach can be used to approximate the Hessian matrix in the new space. After returning back to the original space, we
get the approximation

\[ B^n = S^{-T} \text{sym}(S^T Y) S^{-1}, \tag{13} \]

where \( S = [s^{n-1}, s^{n-2}, \ldots, s^0] \), \( Y = [y^{n-1}, y^{n-2}, \ldots, y^0] \in \mathbb{R}^{n \times n} \), and for any matrix \( A \), the symmetrization operation is defined as

\[
(sym A)_{ij} = \begin{cases} 
A_{ij}, & i \geq j, \\
A_{ji}, & \text{otherwise}. 
\end{cases}
\]

Note that \( B^n = f'' \), if \( f(x) \) is quadratic. If not, multipoint symmetric secant formula (13) gives a good approximation to \( f''(x^n) \), provided that the matrix \( S \) is “safely” nonsingular (see [9]).

Let us compare the approximation (13) with the one \( B^n = Y S^{-1} \) given by the classic multipoint secant method [42]. In the new subspace, where \( s^{k+1} \) is parallel to \( \tilde{e}^i \), \( i = 1, \ldots, n \), it is easy to see for each element of the approximations, how “fresh” is the information involved in its computation. Comparing these two approximations, say, row by row (see Fig. 2), one can see that the symmetric one uses more “fresh” information comparing to the classic one, which uses in each row all spectrum of information, from the most “fresh” to the “oldest”. Such comparison of (13) with the symmetric versions of the secant method proposed in [44] demonstrates the same advantages of our approach. This is the reason why, in the symmetric case \( (f''^R = f'') \), (13) generates better approximations, and why this approach is applied here in the limited-memory framework. An important property of (13) is that \( B^n \) can be obtained, for any initial \( B^0 \in \mathbb{R}^{n \times n} \), as the result of \( n \) sequential updatings by the rank-two formula

\[
B^{k+1} = B^k + \frac{(y^k - B^k s^k) (c^k)^T + c^k (y^k - B^k s^k)^T}{\langle s^k, c^k \rangle} - \frac{\langle y^k - B^k s^k, s^k \rangle c^k (c^k)^T}{\langle s^k, c^k \rangle^2},
\tag{14}
\]

where \( c^k \) is any vector in \( \mathbb{R}^n \), such that

\[
\langle c^k, s^i \rangle = 0, \quad 0 \leq i < k, \tag{15}
\]

\[
\langle c^k, s^k \rangle \neq 0. \tag{16}
\]
The sequence \( \{B^k\}_{k=0}^n \) is well defined by (14)-(16) in the sense that there is no break-down for all \( k = 0, \ldots, n - 1 \). Though we assume from now on that \( B^0 \) is symmetric, some of the further assertions do not require this assumption.

It can be easily shown by analogy with [12] that formulas (14)-(16) generate symmetric Hessian approximations that satisfy for all \( k = 0, \ldots, n - 1 \) the following equations

\[
\begin{align*}
(S^k)^T B^{k+1} s^k &= \text{sym}((S^k)^T Y^k), \\
s^T B^{k+1} s^k &= s^T Y^k, \quad \forall s \perp S^k, \\
(S^k)^T B^{k+1} s &= (Y^k)^T s, \quad \forall s \perp S^k,
\end{align*}
\]

where \( S^k = [s^k, s^{k-1}, \ldots, s^0] \), \( Y^k = [y^k, y^{k-1}, \ldots, y^0] \in \mathbb{R}^{n \times (k+1)} \). These equations imply the quasi-Newton equation

\[
B^{k+1} s^k = y^k. \tag{20}
\]

Note that the vector \( c^k \), as well as \( B^{k+1} \), are not uniquely defined by (15) and (16). The uniqueness can be obtained, if we assume that \( B^{k+1} \) is the solution to the following least-change problem

\[
\begin{align*}
\text{Minimize} & \quad \| B - B^k \|_F, \\
\text{subject to} & \quad (S^k)^T B s^k = \text{sym}((S^k)^T Y^k), \\
& \quad s^T B s^k = s^T Y^k, \quad \forall s \perp S^k, \\
& \quad (S^k)^T B s = (Y^k)^T s, \quad \forall s \perp S^k,
\end{align*}
\]

where \( \| \cdot \|_F \) is the Frobenius matrix norm, and \( B^k \) is supposed to satisfy equations similar to (17) and (18). The solution to this problem is unique, and it is given by formula (14) with

\[
c^k = [I - S^{k-1}((S^{k-1})^T S^{k-1})^{-1}(S^{k-1})^T] s^k.
\]

This means that the sequence \( \{c^k\}_{k=0}^{n-1} \) can be obtained, e.g., by the Gram-Schmidt orthogonalization process applied to the sequence \( \{s^k\}_{k=0}^{n-1} \). Denoting

\[
C^k = \left[ \frac{c_0}{\|c_0\|}, \ldots, \frac{c_k}{\|c_k\|} \right] \in \mathbb{R}^{n \times (k+1)},
\]

Figure 2: The symmetric (left) and the classic (right) secant approximations with indication, for each element, the iteration at which its pair \( (s, y) \) was computed.
we see that $ (C^k)^T C^k = I $ and
\[
c^k = [I - C^{k-1}(C^{k-1})^T]s^k. \tag{22}
\]

This choice of $ c^k $ assures that the equation
\[
s^T B^k s = s^T B^0 s, \quad \forall s \perp S^{k-1} \tag{23}
\]
holds for all $ k = 1, \ldots, n $. Note that the sequence of approximations $ B^k $ is uniquely defined by (17)-(19) and (23). Our limited-memory approach will be essentially based on this property.

In the limited-memory methods, the Hessian matrix is approximated by a low-rank modification of a simple matrix $ B^0 $. In the next theorem, we present the multipoint symmetric secant approximations in the form that will be useful for implementation in the framework of the limited-memory approach. For simplicity, the upper indices of $ S^k $ and $ Y^k $ will be omitted.

**Theorem 4.1.** Let $ S = [s^k, s^{k-1}, \ldots, s^0] \in \mathbb{R}^{n \times (k+1)} $ be a full-rank matrix. Suppose that the matrices $ B^1, \ldots, B^{k+1} $ are generated by formulas (14) and (22). Then for any $ B^0 \in \mathbb{R}^{n \times n} $,
\[
B^{k+1} = (I - S(S^T S)^{-1} S^T) B^0 (I - S(S^T S)^{-1} S^T) + \begin{bmatrix} S & Y \end{bmatrix} \begin{bmatrix} -(S^T S)^{-1} \text{sym}(Y^T S)(S^T S)^{-1} & (S^T S)^{-1} \\ (S^T S)^{-1} & 0 \end{bmatrix} \begin{bmatrix} S^T \\ Y^T \end{bmatrix}, \tag{24}
\]

where $ Y = [y^k, y^{k-1}, \ldots, y^0] \in \mathbb{R}^{n \times (k+1)} $.

**Proof.** Let $ S_{\perp} \in \mathbb{R}^{n \times (n-k)} $ be any matrix such that
\[
S^T_{\perp} S_{\perp} = I \quad \text{and} \quad S^T_{\perp} S = 0.
\]

Then equations (17)-(19) and (23) can be written as
\[
\begin{bmatrix} S^T \\ S^T_{\perp} \end{bmatrix} B^{k+1} \begin{bmatrix} S & S_{\perp} \end{bmatrix} = \begin{bmatrix} \text{sym}(S^T Y) & Y^T S_{\perp} \\ S^T_{\perp} Y & S^T_{\perp} S_{\perp} B^0 S_{\perp} \end{bmatrix}. \tag{25}
\]

By the assumption, the matrix $ [S \ S_{\perp}] \in \mathbb{R}^{n \times n} $ is nonsingular. For its inverse matrix, we have
\[
[S \ S_{\perp}]^{-1} = \begin{bmatrix} (S^T S)^{-1} S \\ S^T_{\perp} \end{bmatrix}.
\]

Therefore, with the use of the evident relations
\[
S_{\perp} S^T_{\perp} = I - S(S^T S)^{-1} S^T,
\]
\[ \text{sym}(Y^T S) - S^T Y - Y^T S = -\text{sym}(S^T Y), \]

formula (24) can be easily derived from (25).

In the limited-memory methods, the initial Hessian approximation is usually chosen as \( B^0 = \gamma I \), where the positive scalar \( \gamma \) may vary from iteration to iteration of the main algorithm that calls our Hessian approximation. It may be called, e.g., by Algorithm 2.1 via Algorithm 3.1. Note that the origin of the vector pairs \( \{ s^i, y^i \} \) mentioned in Theorem 4.1 is not essential for using formula (24) in the sense that, given the matrices \( S \) and \( Y \), a Hessian approximation \( B \) can be computed by formula (24). In our approximation, we choose a small collection of vector pairs \( \{ s^i, y^i \} \) among those recently generated by the main algorithm. The number of these pairs, denoted by \( m \), may vary from iteration to iteration. Then we compose accordingly the matrices \( S, Y \in \mathbb{R}^{n \times m} \) and apply formula (24) with \( B^0 = \gamma I \). This gives

\[
B = \gamma I + \begin{bmatrix} S & Y \end{bmatrix} \begin{bmatrix} -W \text{sym}(Y^T S)W - \gamma W & W \\ W & 0 \end{bmatrix} \begin{bmatrix} S^T \\ Y^T \end{bmatrix},
\]

where \( W = (S^T S)^{-1} \in \mathbb{R}^{m \times m} \). The middle matrix is of the size \( 2m \times 2m \). For \( m \ll n \), this means that the matrix \( B \) is a low-rank correction of \( \gamma I \). It is the most essential property of the limited-memory methods. The CG method applied in Algorithm 3.1 benefits from the fact that the number of distinct eigenvalues of the matrix \( B \) does not exceed \( 2m + 1 \).

Note that our limited-memory formulas (24) and (26) differ from the conventional ones, because they are based on different quasi-Newton methods. By analogy to the other limited-memory approaches, the matrix \( B \in \mathbb{R}^{n \times n} \) in our version is not stored explicitly, but instead, the smaller matrices \( S, Y \in \mathbb{R}^{n \times m} \), \( S^T S, Y^T S \in \mathbb{R}^{m \times m} \) are stored and updated. If all we need from the Hessian approximation is just the products of the form \( Bv \) and \( u^T Bv \), which is the case in Algorithm 3.1, these products can be computed by formula (26) with the use of the stored matrices.

If the vectors \( s^i \) to be used in (26) are linearly dependent, the matrix \( S^T S \) is not invertible. Even if it is invertible, but the vectors are almost linearly dependent, the Hessian approximation may be poor. This is typical for the sequential multipoint secant approximations. To avoid such instability in approximation, we use in (26) not all of the available vectors \( s^i \), but a subset. By analogy with [9, 10, 11, 12], the vectors are specially selected for this subset to be safely linearly independent in some sense. The stable methods in [11, 12] enjoy the superlinear convergence property due to the fact that they use only safely linearly independent vectors \( s^i \) for their multipoint symmetric secant approximations, which are
basic for our limited-memory approximations. In the next section, we introduce a measure of linear independence. Any set of vectors, with the value of this measure above a fixed threshold value \( \sigma \in (0, 1] \), will be regarded as safely linearly independent.

Suppose the Hessian approximation \( B^k \) is called at the \( k \)th iteration of the main algorithm. We limit the maximal number of the vector pairs \( \{s^i, y^i\} \) used in constructing \( B^k \) by a parameter \( m_1 \ll n \). Another parameter, \( m_2 \geq m_1 \), protects from using the pairs \( \{s^i, y^i\} \) with \( i < k - m_2 \) (we call such pairs old, in contrast to the other pairs that we call recent). A general scheme of our limited-memory algorithm can be presented as follows.

**Algorithm 4.1.** Given \( m_1 \ll n, \ m_2 \geq m_1, \ \nu \in (0, 1), \ \gamma_k > 0 \) and a set of recent vector pairs \( \{s^i, y^i\} \).

**Step 1.** From the given set of recently computed vectors \( s^i \) choose a subset of at most \( m_1 \) vectors such that, first, \( s^{k-1} \) is included in the subset, second, the vectors in the subset are safely linearly independent. Use all \( s^i \) from the subset as columns (preferably in decreasing order of \( i \)) for composing the matrix \( S \). Compose the matrix \( Y \) accordingly.

**Step 2.** Construct \( B^k \) by formula (26) with \( \gamma = \gamma_k \).

Note that Step 1 can be implemented in various ways. The simplest choice that could meet all the requirements of Step 1 would be to compose \( S \) of just one column \( s^{k-1} \). In this case, formula (26) is equivalent to the Powell-symmetric-Broyden update [22] of the matrix \( B^0 = \gamma_k I \), but unfortunately, this would not take advantage of using information about the Hessian carried by the other recently computed vector pairs \( \{s^i, y^i\} \).

Following the approach aimed on complete use of the available information [9, 10, 11, 12], and thereby on the better Hessian approximation, we involve, first of all, \( \{s^{k-1}, y^{k-1}\} \) in constructing the Hessian approximation, and then we try to involve sequentially less and less recent vector pairs. Specifically, we set initially \( S = [s^{k-1}] \), and then we check, in decreasing order of the iteration number \( i \), whether the safe linear independence of the columns of \( S \) would be lost after adding \( s^i \) as a new column. If this is not the case, \( S \) is enlarged by adding \( s^i \) as its new last column. We stop after checking in this way all \( s^i \) from the given set of recent vectors. Our implementation of this approach is discussed in the next section.
5 Implementation features

We address here to some implementation issues concerning the algorithms presented in the previous sections. The results of numerical experiments are discussed in the next section.

5.1 Updating of $S$ and $B$ at the $k$-th iteration

After the new point $x^{k+1}$ has been generated, we have the new vector pair $\{s^k, y^k\}$ available. Then the matrix $S$ can be updated as described below to have the new one, $S^k$, available for constructing $B^{k+1}$ by formula (26).

Having available $S^{k-1}$, we check whether the vector $s^k$ and the columns of $S^{k-1}$ are safely linearly independent. If so, we simply add $s^k$ to $S^{k-1}$ as a new column so that $S^k = [s^k S^{k-1}]$. Otherwise, we compose $S^k$ of $s^k$ and of some columns of $S^{k-1}$ in such a way that the columns of $S^k$ are safely linearly independent.

To maintain the safe linear independence of the columns of $S$, we use and update the $QR$ decomposition of the matrix $\bar{S}$, such that $\bar{S}P = S$, where $P$ is a permutation matrix, which ensures that the columns $s^i$ in $S$ are ordered in decreasing order of the iteration number $i$. Thus, we assume that the following matrices are available from the previous iteration: the orthogonal matrix $Q^{k-1} \in \mathbb{R}^{n \times m}$, the right-triangular matrix $R^{k-1} \in \mathbb{R}^{m \times m}$ and the permutation matrix $P^{k-1} \in \mathbb{R}^{n \times n}$, such that the matrix $S^{k-1} = Q^{k-1} R^{k-1} P^{k-1}$ has the desired ordering of columns. For $k = 0$, our initialization of $S^{k-1}$ corresponds to the choice $m = 0$. To simplify the linear algebra involved, we do the same at the subsequent iterations, whenever the working face changes. We call this resetting.

Our criterion of safe linear independence is based on the following definition. Given $\sigma \in (0, 1]$, a matrix $A = [a^1, \ldots, a^m] \in \mathbb{R}^{n \times m}$ with $m \leq n$ is said to be \emph{\(\sigma\)-regular}, if for all $i = 1, \ldots, m$,

$$|\sin \varphi_i| \geq \sigma,$$

where $\varphi_i$ is the angle between the column $a^i$ and the subspace generated by the preceding columns $a^1, \ldots, a^{i-1}$. Note that the column lengths $\|a^i\|$ are not essential in this definition.

An additional point to emphasize is that, if the matrix $R$ in the $QR$ decomposition of $A$ is available, the left-hand side of inequality (27) can be easily computed by the formula

$$|\sin \varphi_i| = R_{ii}/\|a^i\|.$$ 

The outlined updating of $\bar{S}$ and of its $QR$ decomposition is presented below by Algorithm 5.1. For simplicity, we use the notations $s_c = s^k$, $\bar{S}_c = \bar{S}^{k-1}$,
\( Q_c = Q^{k-1}, \ R_c = R^{k-1} \) and \( P_c = P^{k-1} \) for the input variables with the subscript \( c \) standing for “current”, and the notations \( \bar{S} = \bar{S}^k, \ Q = Q^k, \ R = R^k \) and \( P = P^k \) for the output variables. The dimension \( m \) is an input-output variable.

**Algorithm 5.1.** Given \( m_1 \ll n, \ m_2 \geq m_1, \ m \leq m_1, \ \sigma \in (0, 1), \ s_c \in \mathbb{R}^n \) and a \( \sigma \)-regular matrix \( \bar{S}_c \in \mathbb{R}^n \times m \) composed of some recent vectors \( s^i \) and at most one old vector. Moreover, given its QR-factors \( Q_c \in \mathbb{R}^n \times m \) and \( R_c \in \mathbb{R}^m \times m \), and also a permutation matrix \( P_c \in \mathbb{R}^m \times m \), such that the columns \( s^i \) of \( S_c = \bar{S}_c P_c \) are ordered in decreasing order of the iteration number \( i \).

**Step 1.** If \( \bar{S}_c \) has an old vector in the last column then set \( m = m - 1 \), exclude the last columns in \( \bar{S}_c \) and \( Q_c \), exclude both the last columns and the last rows in \( R_c \) and \( P_c \).

**Step 2.** If \( (m < m_1) \) and (there is no old vector in \( \bar{S}_c \)) then:
Set \( \bar{r} = Q_c^T s_c, \ q = s_c - Q_c \bar{r}, \ r = \|q\| \) and \( q = q/r \).
If \( r > \sigma \|s_c\| \) then
set \( m = m + 1, \bar{S} = [\bar{S}_c \ s_c], \ Q = [Q_c \ q] \),
\[
R = \begin{bmatrix} R_c & \bar{r} \\ 0 & r \end{bmatrix}, \ P = \begin{bmatrix} 0 & P_c \\ 1 & 0 \end{bmatrix}
\]
and stop.
end if
end if

**Step 3.** Set \( m = 1, \bar{S} = [s_c], \ Q = [s_c/\|s_c\|], \ R = [\|s_c\|], \ P = [1] \).

**Step 4.** Checking one by one the vectors \( s^i \) that compose the columns of \( S_c \) in decreasing order of \( i \), while \( i > k - m_2 \) and \( m < m_1 \), do:
Set \( \bar{r} = Q^T s^i, \ q = s^i - Q \bar{r}, \ r = \|q\| \) and \( q = q/r \).
If \( r > \sigma \|s^i\| \) then
set \( m = m + 1, \bar{S} = [\bar{S} \ s^i], \ Q = [Q \ q] \),
\[
R = \begin{bmatrix} R & \bar{r} \\ 0 & r \end{bmatrix}, \ P = \begin{bmatrix} P & 0 \\ 0 & 1 \end{bmatrix}
\]
end if
and do

One can see that the output matrix \( \bar{S} \) is \( \sigma \)-regular, and that its column vectors \( s^i \) are not old. Moreover, the perturbation matrix of simple structure
\[
P = \begin{bmatrix}
  0 & 1 & \cdots & 0 \\
  1 & 0 & \cdots & 0 \\
 \vdots & \vdots & \ddots & \vdots \\
  0 & 1 & \cdots & 0
\end{bmatrix}
\]

ensures the desired ordering of columns in the matrix \( S = \tilde{S}P \).

Our process of updating the QR decomposition can be viewed as a sort of the Gram-Schmidt orthogonalization process. Though this process in its classical form may loose, in some cases, the orthogonality of \( Q \), its numerical stability can be significantly improved with the use of the approaches discussed, e.g., in [7, 20, 36].

Having available the matrix \( S \), we compose the corresponding matrix \( Y \) and, if required, construct the Hessian approximation by formula (26), in which we set \( W = P^T R^{-1} R^{-T} P \).

Note that the Q-factor is not involved in this Hessian approximation. Therefore, some savings, both in the computational costs and in the memory requirements, can be obtained if, by analogy with [35], we abandon the computation of the matrix \( Q \) in Algorithm 5.1. This means that all the appearances of products of the form \( Q^T v \) and \( Q^T v \) should be substituted by \( R^{-T} S_c^T v \) and \( R^{-T} S_c^T v \), respectively, where \( v \in \mathbb{R}^n \). Since in Algorithm 5.1 the vector \( v \) is either \( s_c \) or a column of \( S_c \), the major computations would involve the small matrices \( S_c^T S_c, R_c \in \mathbb{R}^{m \times m} \) and the small vector \( S_c^T s_c \in \mathbb{R}^m \) only. Since for \( m \ll n \) the major cost of the implicit Hessian approximation \( B^{k+1} \) by formula (26) is determined by the computation of the two products \( S^{k-1} s^k \) and \( S^{k-1} y^k \), the overall computational cost of this approximation can be estimated as \( 2mn \) flops. The outlined approach is expected to improve in the future our implementation of Algorithm 4.1.

5.2 Computation of \( \alpha \) at Step 2 of Algorithm 2.1

Recall that in Algorithm 2.1 we define the step length \( \alpha \) in the chopped direction \( g_C(x_k) \) as the minimum between \( \tau_k \) and \( \alpha_{\text{max}} \), where \( \tau_k \in [\tau_{\text{min}}, \tau_{\text{max}}] \). To take into account the approximated second order information at this step we adopted the spectral choice (see [40],[43]):

\[
\tau_k = \max \left( \tau_{\text{min}}, \min \left( \tau_{\text{max}}, \frac{s_k^T s_k}{s_k^T y^k} \right) \right)
\]  

(28)
5.3 Initial Hessian approximation

For constructing the matrix $B^k$, it is necessary to specify in (26) the value of $\gamma$, which is associated with the initial Hessian approximation $B^0 = \gamma I$. For $k = 0$, we set $\gamma = |f(x^0)|$, like in one of the most common choices for the initial Hessian approximation in the quasi-Newton methods (see [22]). If the resulting $\gamma$ is less than a tolerance $\varepsilon$, we set $\gamma = 1.0$, which is equivalent to the choice $B^0 = I$. For $k > 0$, we follow the spectral choice (28) by setting $\gamma = 1/\tau_k$.

6 Numerical experiments

Algorithm 2.1, along with the Sub-algorithm 3.1 and the multipoint symmetric secant approximations $B^k$, define an implementable algorithm for box-constrained minimization of differentiable functions. Each particular implementation is determined by the choice of values of the free parameters. We adjusted them, using a small set of test problems, and obtained for our implementation the following default options:

- $\varepsilon = 10^{-5}$, tolerance for 2-norm of projected gradient $g_P$ (Algorithm 2.1).
- $\eta = 0.9$, tolerance in the test to quit the face or not (Algorithm 2.1).
- $\tau_{\text{min}} = 10^{-3}$, $\tau_{\text{max}} = 10^3$, bounds for the spectral parameter $\tau_k$ (Algorithm 2.1).
- $\theta = 10^{-4}$, line search parameter (Algorithm 2.1).
- $\beta_{\text{min}} = 0.1$, $\beta_{\text{max}} = 0.9$, parameters specifying the allowed decrease interval for $\alpha$ (Algorithm 2.1).
- $M = 10$, multiplier increasing search direction length (Case 2 in Algorithm 3.1).
- $\delta_0 = 1.0$, initial trust region radius (Algorithm 3.1).
- $\sigma = 0.01$, threshold in the $\sigma$-regularity test (Algorithm 5.1).
- $m_1 = 5$, maximal number of columns in the matrix $S$ (Algorithm 5.1).
- $m_2 = 7$, parameter protecting from using too old vectors $s^j$ (Algorithm 5.1).
The resulting code, named BSS, was compared with the code LANCELOT [16, 18] on a set of 20 bound constrained problems from the CUTE collection [8]. BSS and LANCELOT were implemented in Fortran 77 and compiled with f77 compiler. All the experiments were done with the -0 optimization compiler option on a SUN UltraSPARC1 station.

LANCELOT was used with the following default options:

- bandsolver-preconditioned-cg-solver-used 5
- exact-Cauchy-point-required
- solve-bqp-accurately
- gradient-tolerance 1.0D-05
- constraints-tolerance 1.0D-06
- maximum-number-of-iterations 5000

The numerical results are presented by Table 1. We list the name of the problem, the type of the objective function ("q", "ssq" and "o" stand for quadratic, sum of squares and other, respectively), the number of variables, the number of iterations (It) and the CPU time (in seconds) for BSS and LANCELOT. For LANCELOT, the results are presented for the three options:

- exact-second-derivatives-used (LAN(1))
- bfgs-approximation-used (LAN(2))
- sri-approximation-used (LAN(3))

BSS and LANCELOT returned close solutions, except the problem S368, marked off by (*), for which maximum number of iterations was reached by LAN(2).

The reported results were obtained for $\eta = 0.9$. This is a rather conservative strategy that worked better than the “greedy” ones, which correspond to small values of $\eta$. This means that, in general, it is worthwhile to stay in the current face, exploiting the quadratic model, instead of trying to change easily the active set of constraints.

We performed testing for different value of $m_1$ in the range from 3 to 20 with $m_2 = 1.5 * m_1$. The results did not change too much with $m_1$. This behavior can be explained by the mentioned strategy of resetting the matrix $S$. 

19
<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>n</th>
<th>BSS</th>
<th>LAN(1)</th>
<th>LAN(2)</th>
<th>LAN(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BQP/GAG1M</td>
<td>q</td>
<td>50</td>
<td>34</td>
<td>0.03</td>
<td>4</td>
<td>0.64</td>
</tr>
<tr>
<td>BQP/GAG1M</td>
<td>q</td>
<td>50</td>
<td>22</td>
<td>0.02</td>
<td>3</td>
<td>0.04</td>
</tr>
<tr>
<td>DECONV</td>
<td>sq</td>
<td>61</td>
<td>724</td>
<td>3.00</td>
<td>14</td>
<td>0.36</td>
</tr>
<tr>
<td>HARKER</td>
<td>p2</td>
<td>100</td>
<td>113</td>
<td>0.37</td>
<td>8</td>
<td>0.94</td>
</tr>
<tr>
<td>HS110</td>
<td>sq</td>
<td>100</td>
<td>2</td>
<td>0.01</td>
<td>1</td>
<td>0.04</td>
</tr>
<tr>
<td>S365</td>
<td>o</td>
<td>100</td>
<td>26</td>
<td>3.44</td>
<td>8</td>
<td>3.14</td>
</tr>
<tr>
<td>EXPLIN</td>
<td></td>
<td>500</td>
<td>77</td>
<td>0.09</td>
<td>11</td>
<td>0.45</td>
</tr>
<tr>
<td>EXPLIN2</td>
<td></td>
<td>500</td>
<td>236</td>
<td>0.66</td>
<td>13</td>
<td>0.48</td>
</tr>
<tr>
<td>EXPQUAD</td>
<td></td>
<td>500</td>
<td>4284</td>
<td>64.63</td>
<td>792</td>
<td>105.74</td>
</tr>
<tr>
<td>QRTQUAD</td>
<td></td>
<td>500</td>
<td>2378</td>
<td>19.72</td>
<td>859</td>
<td>239.15</td>
</tr>
<tr>
<td>CVXBP1</td>
<td>q</td>
<td>10000</td>
<td>15</td>
<td>1.23</td>
<td>1</td>
<td>4.76</td>
</tr>
<tr>
<td>HAIFLDC</td>
<td>sq</td>
<td>10000</td>
<td>396</td>
<td>184.74</td>
<td>4</td>
<td>4.63</td>
</tr>
<tr>
<td>MCCORMCK</td>
<td>o</td>
<td>10000</td>
<td>17</td>
<td>5.87</td>
<td>4</td>
<td>4.73</td>
</tr>
<tr>
<td>NONSOMP</td>
<td>sq</td>
<td>10000</td>
<td>23</td>
<td>3.05</td>
<td>8</td>
<td>6.63</td>
</tr>
<tr>
<td>NCXVRBP1</td>
<td>q</td>
<td>10000</td>
<td>5</td>
<td>0.66</td>
<td>4</td>
<td>8.31</td>
</tr>
<tr>
<td>NCXVRBP2</td>
<td>q</td>
<td>10000</td>
<td>60</td>
<td>8.96</td>
<td>5</td>
<td>11.94</td>
</tr>
<tr>
<td>FENTDI</td>
<td>q</td>
<td>10000</td>
<td>3</td>
<td>0.37</td>
<td>1</td>
<td>3.05</td>
</tr>
<tr>
<td>PROFBNL</td>
<td>o</td>
<td>10000</td>
<td>3</td>
<td>0.61</td>
<td>1</td>
<td>27.63</td>
</tr>
<tr>
<td>QUADIN</td>
<td>q</td>
<td>10001</td>
<td>53</td>
<td>2.51</td>
<td>4</td>
<td>8.30</td>
</tr>
<tr>
<td>TOHSION6</td>
<td>q</td>
<td>14884</td>
<td>292</td>
<td>123.73</td>
<td>8</td>
<td>17.29</td>
</tr>
</tbody>
</table>

Table 1. Performance of BSS versus LANCELOT.

Note that too large values of the parameter $\sigma$, say above 0.1, would be very restrictive in the sense that too many vectors $s^i$ should be rejected in the process of Hessian approximation. The numerical results were not too different for the values $\sigma = 0.1, 0.01$ and $0.001$, mainly because of the resetting strategy. Since $\sigma = 0.01$ produced slightly better results, we adopted this parameter for our implementation.

The performed testing of LANCELOT and of our limited memory algorithm BSS was not aimed on establishing a superiority of any of them over the other. We used LANCELOT simply as a benchmark. For this reason, we ran it only with its default settings and did not experiment with its various options to find the one that would give the best results on the test problems. Moreover, we tested just the mentioned three ways of LANCELOT to compute or approximate the Hessian. However, the obtained numerical results were sufficient to draw some conclusions.

Observe that, in general, the number of iterations of BSS is larger than the one of LANCELOT with second derivatives, but this does not reflect on the computer time. The reason is that our subproblems are very cheap due to the low-rank character of the Hessian approximations, and so, the number of the CG-iterations in the trust-region subproblems is very small.

20
7 Conclusions

Active set methods are among the most traditional tools of constrained optimization. Their appeal comes from the fact that they allow the algorithmic designer to take full advantage of previously developed unconstrained optimization techniques. As far as new ideas in unconstrained minimization continue to be introduced, the implementation of active set methods based on those ideas is a natural task.

The unconstrained optimization technique exploited in this paper is the memoryless multipoint symmetric secant scheme, which is related to quasi-Newton methods. The fulfillment of several secant equations within a given face (or subspace) usually ensures Newton-like properties of the search directions generated on that face. On the other hand, since the approximate Hessians so far generated are not necessarily positive definite, a trust-region strategy for global convergence is in order. In this paper we adopted the memoryless approach, thanks to which large problems can be solved. Moreover, a small number of low-rank corrections guarantees that the Hessian approximations possess a small number of distinct eigenvalues and, so, the CG method is efficient for solving the resulting quadratic subproblems.

The comparison with LANCELOT reveals that the method introduced here is reliable, and moreover, that it is able to compete successfully with the well-elaborated optimization solvers, like LANCELOT. It is interesting to observe that the new method worked very well in problems where the performance of LANCELOT was rather poor (NCVXBQP1, PENTDI, PROBPENL, QUDLIN), whereas LANCELOT was more efficient in others (HATFLDC, TORSION6). This fact indicates that the trust-region strategy of LANCELOT and of other box-constrained solvers is complementary to the active-set strategy in the sense that difficult problems for one of them are relatively easy for the other.

As it was mentioned in Introduction, one of the main purposes of box-constrained solvers rely on their efficient application as sub-algorithms for more general nonlinear programming algorithms (see [17, 38]). This will be one of our research objectives for the near future. Moreover, as it was mentioned in Subsection 5.1, we are going also to employ some ideas of the paper [35] to improve the computational efficiency of our limited-memory multipoint symmetric secant approximation of the Hessian. Note that the current implementation is based on resetting of the matrix $S$, which is applied whenever the working face changes. With an affordable complication of the linear algebra involved, we can avoid this resetting. This could contribute to the improvement of the efficiency.
References


