Numer. Math. (2006) 104: 241–269
DOI 10.1007/s00211-006-0021-6

Zhou-Hong Wang · Ya-Xiang Yuan

A subspace implementation of quasi-Newton trust region methods for unconstrained optimization

Received: 5 January 2005 / Revised: 22 December 2005 / Published online: 18 July 2006
© Springer-Verlag 2006

Abstract This paper studies subspace properties of trust region methods for unconstrained optimization, assuming the approximate Hessian is updated by quasi-Newton formulae and the initial Hessian approximation is appropriately chosen. It is shown that the trial step obtained by solving the trust region subproblem is in the subspace spanned by all the gradient vectors computed. Thus, the trial step can be defined by minimizing the quasi-Newton quadratic model in the subspace. Based on this observation, some subspace trust region algorithms are proposed and numerical results are also reported.

Mathematics Subject Classification (2000) 65K05 · 90C53

1 Introduction

Consider the general unconstrained optimization problem

$$\min_{x \in \Re^n} f(x), \tag{1}$$

where $f(x)$ is a continuously differentiable function defined in $\Re^n$. Trust region methods for the unconstrained optimization problem 1 compute a trial step in each iteration. Such a trial step is required to be within a "trust region", which is a

Z.-H. Wang (✉) · Y.-X. Yuan
State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, Beijing 100080, People's Republic of China
E-mail: wangzhh@bjtu.edu.cn
E-mail: yyx@lsec.cc.ac.cn

Z.-H. Wang
Department of Mathematics, Beijing Jiaotong University, Beijing 100044, People's Republic of China

region near the current iterate point. Different choices of trust regions and different models give different trust region algorithms. The most commonly used model is the quadratic one, which gives the following trust region subproblem:

$$\min_{s\in\Re^n} \phi_k(s) \equiv g_k^T s + \frac{1}{2}s^T B_k s \tag{2}$$

$$\text{s. t.}\|s\| \leq \Delta_k, \tag{3}$$

where $\|\cdot\|$ is the Euclidean norm, $g_k = g(x_k) \equiv \nabla f(x_k)$ is the gradient of the objective function at the current iterate x_k, B_k is an approximate Hessian, and $\Delta_k > 0$ is the trust region radius. In practice, B_k is updated by using quasi-Newton formulae. More detailed discussions on trust region algorithms can be found in Dennis and Schnabel [8], Fletcher [10], Conn, et al. [5] and Yuan [30]. The following is a description of a model trust region algorithm for unconstrained optimization.

Algorithm 1 (A Model Trust Region Algorithm)

Step 1. Given $x_1 \in \Re^n$, $\Delta_1 > 0$, $0 < \tau_3 < \tau_4 < 1 < \tau_1$, $0 \leq \tau_0 \leq \tau_2 < 1$, $\tau_2 > 0$, $\varepsilon \geq 0$, $k := 1$.

Step 2. Solve Eqs. (2) and (3) approximately to get a solution s_k. If $\|s_k\| \leq \varepsilon$ then stop.

Step 3. Compute

$$\eta_k = \frac{\text{Ared}_k}{\text{Pred}_k} = \frac{f(x_k) - f(x_k + s_k)}{\phi_k(0) - \phi_k(s_k)}. \tag{4}$$

Step 4. Set

$$x_{k+1} = \begin{cases} x_k + s_k, & \text{if } \eta_k > \tau_0, \\ x_k, & \text{otherwise.} \end{cases} \tag{5}$$

Choose Δ_{k+1} that satisfies

$$\Delta_{k+1} = \begin{cases} [\tau_3\|s_k\|, \tau_4\Delta_k], & if \quad \eta_k < \tau_2, \\ [\Delta_k, \tau_1\Delta_k], & otherwise. \end{cases} \tag{6}$$

Step 5. Update B_{k+1}. Set $k := k + 1$ and go to Step 2.

In this paper, we study the case when the matrix B_k is updated by quasi-Newton methods, namely B_{k+1} is generated by adding a modification to B_k and satisfies the quasi-Newton equation

$$B_{k+1}s_k = y_k = g(x_k + s_k) - g(x_k) \equiv g_{k+1} - g_k. \tag{7}$$

When the number of variables is very large, it could be very costly to solve Eqs. (2) and (3) exactly. Therefore, various methods for calculating an approximate solution of Eqs. (2) and (3) have been developed, such as the dogleg [18,19] and double dogleg techniques [7], the truncated CG method [25,31], two-dimensional subspaces minimization method [2], subspace CG methods [27] and subspace-iterated methods [3], etc. How to solve the subproblem 2 and 3 as exactly as possible in an acceptable effort is still a problem worth studying. It is Siegel [22] who first proposed to implement Broyden class updates in a sequence of expanding subspace. Gill and Leonard [12] developed a practical reduced Hessian method based on

[22], where the technique of combining lingering with reinitialization is proposed. All these subspace methods are line search methods, and require the approximate Hessian matrix B_k to be positive definite. Motivated by these results, we explore the subspace properties of trust region methods when the approximate Hessian is updated by quasi-Newton formulae. It is found that the trial step s_k defined by the trust region subproblem is also always in the subspace spanned by $g_1, g_2, \ldots, g_k$, even when B_k is not positive definite. Therefore, it is equivalent to solve the trust region subproblem within this subspace. Based on this observation, we can solve a smaller trust region subproblem exactly in early iterations of the algorithm, which would reduce the computations significantly for some large-scale problems, where the dimension of the subspace spanned by $g_1, g_2, \ldots, g_k$ remains far less than the number of variables n.

The paper is organized as follows. The equivalence of the trust region subproblem and that in the subspace is proved in the next section. In Sect. 3, two subspace quasi-Newton trust region algorithms are proposed and their convergence properties are analyzed. Numerical results on problems in CUTEr see [1] and [14] are reported in Sect. 4, where both the BFGS and the SR1 updating formulae are used. The purpose of this paper is to show that subspace techniques can be used to improve the performance of trust region methods, and we do not compare our algorithms with line search subspace methods.

2 Subspace properties of the trial step

In this section, we study the subspace properties of the trial step s_k at the k th iteration, which is assumed to be the solution of the trust region subproblem 2 and 3. The following Lemma, about the trial step s_k, can be found in Gay [11] and Sorensen [24].

Lemma 2.1 *The vector $s_k \in \Re^n$ is a global solution of Eqs.(2) and (3) if and only if $\|s_k\| \leq \Delta_k$ and there exists a scalar $\lambda_k \geq 0$ such that $B_k + \lambda_k I$ is positive semi-definite and*

$$(B_k + \lambda_k I)s_k = -g_k, \tag{8}$$

$$\lambda_k(\Delta_k - \|s_k\|) = 0. \tag{9}$$

Consider a special case when B_k is a two by two block diagonal matrix having the form

$$B_k = \begin{bmatrix} B_k(1:r, 1:r) & 0 \\ 0 & \sigma I \end{bmatrix} \tag{10}$$

where $B_k(1:r, 1:r)$ is the r th leading submatrix of B_k (see, e.g., Golub and Van Loan [13] for the notation). From Eqs. (8) and (10), it is easy to see that $s_k(r+1:n)$ (the last $n-r$ components of s_k) are all zero if $g_k(r+1:n) = 0$. This simple example indicates that if B_k has certain structure, the trial step will be always in a subspace. In the above example, the subspace is spanned by the first r coordinate vectors $e_i (i = 1, \ldots, r)$. A similar result is also true for other subspaces. Let $\mathcal{G}_k = \mathrm{Span}\{g_1,\ g_2, \ldots,\ g_k\}$. If B_k is obtained by Broyden updates and $B_k + \lambda_k I$ is positive definite in Algorithm 1, we would have $s_k \in \mathcal{G}_k$, which follows immediately from Eq. (8) and the theory on line search methods, see, e.g.

[12,22,28]. The main difficulty in using the subspace techniques for trust region methods lies in the "hard case", where $B_k + \lambda_k I$ is not invertible. Let ν_1 be the smallest eigenvalue of B_k, $V_1 = \{x \in \Re^n \mid B_k x = \nu_1 x\}$. Then in "hard case" we have $g_k \in V_1^{\perp}$ and the solution s_k of Eqs. (2) and (3) satisfies (see, e.g. Conn et al. [5] and Sorensen [24])

$$s_k = -(B_k + \lambda_k I)^{+} g_k + \theta q$$

where $(^{+})$ denotes the Moore–Penrose generalized matrix inverse, $q \in V_1$, $\|B_k + \lambda_k I)^{+} g_k\| < \Delta_k$ and $\theta > 0$. In this case it is not easy to see whether $s_k \in \mathcal{G}_k$. In fact the result is still true for trust region methods and we describe it in the following Lemma.

Lemma 2.2 *Let S_k be an r $(1 \le r \le n)$ dimensional subspace in $\Re^n$, and $Z_k \in \Re^{n\times r}$ is an orthonormal basis matrix of S_k, namely*

$$S_k = Span\{Z_k\}, \quad Z_k^T Z_k = I_r. \tag{11}$$

Suppose that $g_k \in S_k$ and B_k is a symmetric matrix satisfying

$$B_k z \in S_k, \ \forall \ z \in S_k, \tag{12}$$
$$B_k u = \sigma u, \ \forall \ u \in S_k^{\perp}, \tag{13}$$

where $\sigma > 0$ is a positive number. Then problem Eqs. (2) and (3) is equivalent to the following problem

$$\min_{\bar{s}\in\Re^r} \bar{\phi}_k(\bar{s}) \equiv \bar{g}_k^T \bar{s} + \frac{1}{2}\bar{s}^T \bar{B}_k \bar{s} \tag{14}$$
$$\text{s. t.} \|\bar{s}\| \le \Delta_k, \tag{15}$$

where $\bar{g}_k = Z_k^T g_k$ and $\bar{B}_k = Z_k^T B_k Z_k$. That is to say, if s_k is a solution of Eqs. (2) *and* (3)*, then $\bar{s}_k = Z_k^T s_k$ is a solution of Eqs.* (14) *and* (15) *and we have $s_k = Z_k \bar{s}_k \in S_k$. On the other hand, if $\bar{s}_k$ is a solution of Eqs.* (14) *and* (15)*, then $s_k = Z_k \bar{s}_k$ must be a solution of Eqs.* (2) *and* (3).

Proof Let $U_k \in \Re^{n\times(n-r)}$ be a matrix such that $[U_k, Z_k]$ is an $n \times n$ orthogonal matrix. Then for each $s \in \Re^n$, there exists one and only one pair $\bar{s} \in \Re^r$, $u \in \Re^{n-r}$ such that $s = Z_k \bar{s} + U_k u$. Thus, it follows that

$$\begin{aligned}\phi_k(s) = g_k^T s + \frac{1}{2}s^T B_k s &= g_Z^T \bar{s} + g_U^T u \\ &+\frac{1}{2}\bar{s}^T Z_k^T B_k Z_k \bar{s} + \frac{1}{2}u^T U_k^T B_k U_k u + \bar{s}^T Z_k^T B_k U_k u,\end{aligned} \tag{16}$$

where $g_Z = Z_k^T g_k, \ g_U = U_k^T g_k$. By $g_k \in \text{Span}\{Z_k\}$ and Eqs. (12) and (13) we get

$$B_k U_k = \sigma U_k, \ Z_k^T B_k U_k = \sigma Z_k^T U_k = 0, \ g_U = U_k^T g_k = 0. \tag{17}$$

Hence, it follows from Eqs. (16) and (17) that

$$\phi_k(s) = \left(g_Z^T \bar{s} + \frac{1}{2}\bar{s}^T Z_k^T B_k Z_k \bar{s}\right) + \frac{1}{2}\sigma u^T u. \tag{18}$$

By the orthonormality of Z_k and U_k, we have $\|s\|^2 = \|\bar{s}\|^2 + \|u\|^2$. Now the above relation indicates that Eqs. (2) and (3) are equivalent to

$$\min_{\bar{s}\in\Re^r, u\in\Re^{n-r}} \left(g_Z^T \bar{s} + \frac{1}{2}\bar{s}^T Z_k^T B_k Z_k \bar{s}\right) + \frac{1}{2}\sigma u^T u \tag{19}$$

$$\text{s. t.} \quad \|\bar{s}\|^2 + \|u\|^2 \le \Delta_k^2, \tag{20}$$

with the relation $s = Z_k\bar{s} + U_k u$. Because $\sigma > 0$, it is easy to see that the above problem is equivalent to Eqs. (14) and (15) with $u = 0$. Thus, Eqs. (2) and (3) is equivalent to Eqs. (14) and (15) with $s = Z_k\bar{s}$. By $Z_k^T Z_k = I_r$(where I_r is the $r \times r$ unit matrix), we get $\bar{s} = Z_k^T s$. □

From the above Lemma, if Eqs. (12) and (13) are satisfied, we can in fact solve the subproblem 14 and 15 in $\Re^r$ instead of solving the subproblem 2 and 3 in $\Re^n$, which can reduce the computation efforts significantly when $k \ll n$.

For line search methods, Siegel [22] proved a subspace property result for Broyden family when B_k is invertible. The following lemma generalized Siegel's result from linear search methods to trust region methods.

Lemma 2.3 *Suppose $B_1 = \sigma I$, $\sigma > 0$. The matrix updating formula is any one chosen from PSB and Broyden family, (where the updates may be singular), and B_k is the kth updated matrix. Let s_k be a solution of Eqs.* (2) *and* (3)*, $g_k = \nabla f(x_k)$, $\mathcal{G}_k = Span\{g_1, g_2, \ldots, g_k\}$ and $x_{k+1} = x_k + s_k$ ($x_1 \in \Re^n$ is any given initial point). Then for all $k \ge 1$, $s_k \in \mathcal{G}_k$. Moreover for any $z \in \mathcal{G}_k$ and any $u \in \mathcal{G}_k^\perp$, we have*

$$B_k z \in \mathcal{G}_k, \quad B_k u = \sigma u. \tag{21}$$

Proof The PSB formula and Broyden family formulae (see, e.g., Dennis and Schnabel [8], Fletcher [10], and Yuan [29]) can be represented as

$$B_{k+1} = B_k + \frac{r_k s_k^T + s_k r_k^T}{s_k^T s_k} - \frac{\left(r_k^T s_k\right) s_k s_k^T}{\left(s_k^T s_k\right)^2}, \tag{22}$$

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{s_k^T y_k} + \theta_k \left(s_k^T B_k s_k\right) w_k w_k^T, \tag{23}$$

where $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$, $r_k = y_k - B_k s_k$, $w_k = (y_k/s_k^T y_k) - (B_k s_k/s_k^T B_k s_k)$. We only prove the results for PSB formula 22 by induction over k. The proof for other update formulae is similar.

By Lemma 2.1 and $\sigma > 0$, it is obvious that the lemma is true for $k = 1$. Now we assume that the lemma is true for $k = i$. In the same way as Siegel [22], Eq. (21) can be proved for $k = i + 1$. Let s_{i+1} be a solution of Eqs. (2) and (3). Then by $g_{i+1} \in \mathcal{G}_{i+1}$ and Lemma 2.2 (where $k = i + 1$), we have $s_{i+1} = Z_{i+1} z_{i+1} \in \mathcal{G}_{i+1}$ (where z_{i+1} is a solution of Eqs. (14) and (15) for $k = i + 1$). Thus, the proof is completed. □

By Lemma 2.2 and Lemma 2.3, we obtain the following theorem.

Theorem 2.1 *Let Z_k be an orthonormal basis matrix of the subspace $\mathcal{G}_k = Span\{g_1, g_2, \ldots, g_k\}$. Suppose $B_1 = \sigma I$, $\sigma > 0$, and B_k is the k th updated matrix given by one formula chosen from PSB and Broyden family. Let s_k be a solution of Eqs.* (2) *and* (3). *Then we have $s_k \in \mathcal{G}_k$ and there exists a solution z_k of Eqs.* (14) *and* (15) *such that $s_k = Z_k z_k$.*

From the above theorem, the trial step s_k is in the subspace $\mathcal{G}_k$. Hence, we can in fact update the approximate Hessian matrix B_k in the subspace $\mathcal{G}_k$ by the PSB formula or any one from the Broyden family. The following result has been given by Siegel [22] and Gill and Leonard [12] for Broyden family. We give it here for completeness.

Lemma 2.4 *Let $Z \in \Re^{n \times r}$ be a column orthogonal matrix such that $Z^T Z = I_r$. Suppose that $s_k \in Span\{Z\}$, and the matrix $B_{k+1} = Update(B_k, s_k, y_k)$ is obtained by the PSB formula or any one from the Broyden family. Denote $\bar{B}_{k+1} = Z^T B_{k+1} Z$, $\tilde{B}_k = Z^T B_k Z$, $\tilde{s}_k = Z^T s_k$, $\tilde{y}_k = Z^T y_k$, then $\bar{B}_{k+1} = Update(\tilde{B}_k, \tilde{s}_k, \tilde{y}_k)$.*

Proof By the assumption $s_k \in \text{Span}\{Z\}$ and $Z^T Z = I_r$, we have $s_k = Z Z^T s_k$, and

$$\begin{aligned} &s_k^T y_k = (Z^T s_k)^T Z^T y_k = \tilde{s}_k^T \tilde{y}_k \\ &s_k^T B_k s_k = (Z^T s_k)^T Z^T B_k Z (Z^T s_k) = \tilde{s}_k^T \tilde{B}_k \tilde{s}_k \\ &Z^T B_k s_k = Z^T B_k Z (Z^T s_k) = \tilde{B}_k \tilde{s}_k \end{aligned} \tag{24}$$

Therefore, multiplying Z^T from left and Z from right to Eqs. (22) and (23), we can see the lemma is true. □

Theorem 2.1 tells us that we can solve the quadratic subproblem 2 and 3 by solving Eqs. (14) and (15) in the subspace with the reduced gradient and the reduced approximate Hessian, provided the initial Hessian approximation is appropriately chosen. It follows from Lemma 2.3 and Lemma 2.4 that in fact the reduced approximate Hessian $\bar{B}_k = Z_k^T B_k Z_k$ of B_k in the subspace $\mathcal{G}_k = \text{Span}\{g_1, g_2, \ldots, g_k\}$ can be obtained by updating the reduced matrix $\tilde{B}_{k-1} = Z_k^T B_{k-1} Z_k$, where Z_k is the orthonormal basis matrix of the subspace $\mathcal{G}_k$. By the relation between Z_k and Z_{k-1}, we can get $\tilde{B}_{k-1}$ by $\bar{B}_{k-1} = Z_{k-1}^T B_{k-1} Z_{k-1}$ easily, which will be described in detail in the next section. These subspace properties can reduce the amount of computation significantly when n is very large and the dimension of the subspace $\mathcal{G}_k$ remains far less than n.

3 The algorithm

Due to the subspace properties studied in the previous section, we can construct subspace trust region algorithms based on the traditional trust region philosophy. Let $\| \cdot \|$ denote Euclidean norm. Suppose at the k th iteration, Z_k have been obtained, which is the orthonormal basis matrix of $\mathcal{G}_k = \text{Span}\{g_1, \dots, g_k\}$, and the dimension of $\mathcal{G}_k$ is r_k. Further, suppose $\bar{s}_k$ is obtained by solving Eqs. (14) and (15) and $s_k = Z_k\bar{s}_k$, $x_{k+1} = x_k + s_k$, $g_{k+1} = \nabla f(x_{k+1})$. Then we have to compute Z_{k+1}, $\bar{g}_{k+1} = Z_{k+1}^T g_{k+1}$, and $\bar{B}_{k+1} = Z_{k+1}^T B_{k+1} Z_{k+1}$ for next iteration. For numerical stability, as proposed by Gill and Leonard [12], we could use reorthogonalization procedure (see Daniel et al. [6], Golub and Van Loan [13], or Stewart [26] for details) to obtain Z_{k+1} by the decomposition

$$g_{k+1} = Z_k u_k + \rho_{k+1} z_{k+1}, \tag{25}$$

where $u_k = Z_k^T g_{k+1}$, $z_{k+1} \perp \text{Span}\{Z_k\}$, $\|z_{k+1}\| = 1$, $\rho_{k+1} = \|(I - Z_k Z_k^T)g_{k+1}\| \geq 0$. From Eq. (25) and the fact that $s_k, g_k \in \text{Span}\{Z_k\}$ we obtain

$$z_{k+1}^T g_{k+1} = \rho_{k+1},\ z_{k+1}^T s_k = 0,\ z_{k+1}^T g_k = 0. \tag{26}$$

Similar to Siegel [22] and Gill and Leonard [12], g_{k+1} is accepted if $\rho_{k+1} > 0$ and we set $Z_{k+1} = [Z_k, z_{k+1}]$, $r_{k+1} = r_k + 1$. Otherwise, $g_{k+1} \in \text{Span}\{Z_k\}$ and we set $Z_{k+1} = Z_k$, $r_{k+1} = r_k$. In this case, g_{k+1} is rejected. If $Z_{k+1} = [Z_k, z_{k+1}]$, it follows from Lemma 2.3 that

$$B_k z_{k+1} = \sigma z_{k+1}. \tag{27}$$

Then by Eqs. (25) – (27) we get

$$\begin{aligned}
\bar{g}_{k+1} &= Z_{k+1}^T g_{k+1} = \begin{bmatrix} Z_k^T g_{k+1} \\ z_{k+1}^T g_{k+1} \end{bmatrix} = \begin{bmatrix} u_k \\ \rho_{k+1} \end{bmatrix}, \\
\tilde{s}_k &= Z_{k+1}^T s_k = \begin{bmatrix} \bar{s}_k \\ 0 \end{bmatrix}, \\
\tilde{y}_k &= Z_{k+1}^T y_k = \begin{bmatrix} Z_k^T (g_{k+1} - g_k) \\ z_{k+1}^T (g_{k+1} - g_k) \end{bmatrix} = \begin{bmatrix} u_k - \bar{g}_k \\ \rho_{k+1} \end{bmatrix}, \\
\tilde{B}_k &= Z_{k+1}^T B_k Z_{k+1} = \begin{bmatrix} Z_k^T B_k Z_k & \sigma Z_k^T z_{k+1} \\ \sigma z_{k+1}^T Z_k & \sigma z_{k+1}^T z_{k+1} \end{bmatrix} = \begin{bmatrix} \bar{B}_k & 0 \\ 0 & \sigma \end{bmatrix}.
\end{aligned} \tag{28}$$

If $Z_{k+1} = Z_k$, by Eqs. (25) and (26) we get

$$\begin{aligned}
\bar{g}_{k+1} &= Z_{k+1}^T g_{k+1} = Z_k^T g_{k+1} = u_k, \\
\tilde{s}_k &= Z_{k+1}^T s_k = Z_k^T s_k = \bar{s}_k, \\
\tilde{y}_k &= Z_{k+1}^T y_k = Z_k^T y_k = u_k - \bar{g}_k, \\
\tilde{B}_k &= Z_{k+1}^T B_k Z_{k+1} = Z_k^T B_k Z_k = \bar{B}_k.
\end{aligned} \tag{29}$$

According to Lemma 2.4, the reduced approximate Hessian $\bar{B}_{k+1} = Z_{k+1}^T B_{k+1} Z_{k+1}$ in the subspace $\text{Span}\{Z_{k+1}\}$ can be obtained by any formula among the PSB and Broyden family (where the SR1 update is included), using $\tilde{B}_k, \tilde{s}_k, \tilde{y}_k$ computed in Eq. (28) or (29). For example, if we choose the BFGS formula and the SR1 formula, and we have computed $\tilde{B}_k, \tilde{s}_k, \tilde{y}_k$ in Eq. (28) or (29), by Lemma 2.4 we obtain

$$\bar{B}_{k+1} = \tilde{B}_k - \frac{\tilde{B}_k \tilde{s}_k \tilde{s}_k^T \tilde{B}_k}{\tilde{s}_k^T \tilde{B}_k \tilde{s}_k} + \frac{\tilde{y}_k \tilde{y}_k^T}{\tilde{s}_k^T \tilde{y}_k} \quad \text{(for BFGS)}, \tag{30}$$

$$\bar{B}_{k+1} = \tilde{B}_k + \frac{(\tilde{y}_k - \tilde{B}_k\tilde{s}_k)^T(\tilde{y}_k - \tilde{B}_k\tilde{s}_k)}{(\tilde{y}_k - \tilde{B}_k\tilde{s}_k)^T\tilde{s}_k} \quad \text{(for SR1)}. \tag{31}$$

Then by Theorem 2.1 we can solve Eqs. (14) and (15) with the reduced matrix $\bar{B}_{k+1}$ and the reduced gradient $\bar{g}_{k+1}$ to obtain the trial step $\bar{s}_{k+1}$ and $s_{k+1} = Z_{k+1}\bar{s}_{k+1}$.

We summarize the above observations in the following algorithm.

Algorithm 2 The Subspace Quasi-Newton Trust Region Algorithm for Unconstrained Optimization

Step 1. Given $x_1 \in \Re^n$, $\Delta_1 > 0$, $0 \le \Delta_{\min} \le 10^{-6}$, $\varepsilon \ge \varepsilon_M \ge 0$, $1 > \nu \ge 0$, $\sigma > 0$, $0 \le \tau_0 < \tau_1 < 0.5 < \tau_2 < 1$, $0 < c_1 < 0.1 < c_2 \le 0.5, c_2 \le c_3 < 1 < c_4$. Choosing one matrix updating formula amongst PSB and Broyden family. Compute $f(x_1)$ and $g_1 = \nabla f(x_1)$. Set $\bar{B}_1 = \sigma$, $\bar{g}_1 = \|g_1\|$, $Z_1 = [g_1/\|g_1\|]$. Set $r := 1, k := 1$.

Step 2. Solve the subproblem 14 and 15 approximately to get $\bar{s}_k$ and $\bar{\phi}(\bar{s}_k)$. If $|\bar{\phi}(\bar{s}_k)| \le \varepsilon_M$ then stop. Otherwise compute $s_k = Z_k\bar{s}_k$ and $\eta_k = f(x_k) - f(x_k + s_k)/(-\bar{\phi}(\bar{s}_k))$.

Step 3. If $\eta_k \le \tau_0$ and $\Delta_k > \Delta_{\min}$, set $\Delta_k := t_1\Delta_k$, where $c_1 \le t_1 \le c_2$, and t_1 is obtained by using the interpolation technique proposed by Dennis and Schnabel [8]. Go to Step 2;

If $\eta_k \le \tau_0$ and $\Delta_k \le \Delta_{\min}$, the algorithm fails and stop; Otherwise, do next step.

Step 4. Set $x_{k+1} = x_k + s_k$ and

$$\Delta_{k+1} = \begin{cases} c_3\Delta_k, & \text{if } \eta_k < \tau_1, \\ c_4\Delta_k, & \text{if } \eta_k > \tau_2 \text{ and } \|s_k\| \ge 0.8\Delta_k, \\ \Delta_k, & \text{otherwise.} \end{cases} \tag{32}$$

Step 5. Compute $g_{k+1} = \nabla f(x_{k+1})$. If $\|g_{k+1}\| \le \varepsilon$, stop; Otherwise do next step.

Step 6. Obtain Eq. (25) by the reorthogonalization procedure.

Step 7. If $\rho_{k+1} > \nu\|g_{k+1}\|$, set $Z_{k+1} = [Z_k, z_{k+1}]$, $r := r+1$. Compute $\tilde{B}_k, \tilde{s}_k, \tilde{y}_k$ according to Eq. (28);

Otherwise set $Z_{k+1} = Z_k$ and compute $\tilde{B}_k, \tilde{s}_k, \tilde{y}_k$ according to Eq. (29).

Step 8. Obtain $\bar{B}_{k+1} = \text{Update}(\tilde{B}_k, \tilde{s}_k, \tilde{y}_k)$ by the chosen matrix updating formula in the subspace Span$\{Z_{k+1}\}$. Set $k := k + 1$ and go to Step 2.

If we set $\nu = 0$ and suppose that there is no floating point error, by Theorem 2.1 and Lemma 2.4 the above algorithm is equivalent to the corresponding traditional trust region algorithm which is superlinearly convergent. In practice, we could set $\nu = 10^{-3}$ as proposed by Siegel [22]. Let r_k denote Dim$(\mathcal{G}_k)$, n denote the number of variables. The reorthogonalization procedure requires approximately $4nr_k$ flops. Since u_k, ρ_k in Eqs. (28) and (29) are obtained by reorthogonalization, the update $\bar{B}_{k+1}$ of $\bar{B}_k$ in Eqs. (30) or (31) requires about $2r_k^2 + O(r_k)$ flops. The computation of a nearly optimal solution of the subproblem Eqs. (14) and (15) (see Moré and Sorensen [16]) requires about $1/6(r_k^3) + 3/2(r_k^2) + O(r_k)$ flops. The computation $s_k = Z_k\bar{s}_k$ in Step 2 requires nr_k flops, with the result that Algorithm 2 requires approximately $5nr_k + 1/6(r_k^3) + 7/2(r_k^2) + O(r_k)$ flops for each iteration when BFGS or SR1 is used. When $r_k = n$, only $Z_k^T g_{k+1}$ is computed during the orthogonalization, and the work drops to $1/6(n^3) + 11/2(n^2) + O(n)$. The corresponding conventional trust region method requires approximately $1/6(n^3)+7/2(n^2)+O(n)$ for each iteration. So Algorithm 2 can reduce the amount of computation significantly in early iterations, especially when n is very large and $r_k \ll n$. In practice, however, the order of the reduced Hessian r_k often remains much less than n when problems with large n are solved, as pointed out in Gill and Leonard [12] and our numerical experiments in Sect. 4.

When $\nu > 0$, the algorithm would be different from the traditional trust region algorithm, but we still have the following convergence result, where dogleg [18, 19], double dogleg [7] and truncated CG methods [25,31] can be used to solve the subproblem 14 and 15.

Theorem 3.1 *Let $\bar{s}_k$ be an approximate solution of Eqs.* (14) *and* (15) *which satisfies (see Powell* [20]*)*

$$-\bar{\phi}(\bar{s}_k) \geq \frac{1}{2}\|\bar{g}_k\| \min\left(\Delta_k, \frac{\|\bar{g}_k\|}{\|\bar{B}_k\|}\right). \tag{33}$$

Suppose that $\{x_k\}$ is generated by Algorithm 2, $f : \Re^n \to R$ is bounded below and $\nabla f(x)$ is uniformly continuous. If $\bar{B}_k$ satisfies

$$\sum_{k=1}^{\infty} 1/M_k = \infty, \tag{34}$$

where M_k is the number

$$M_k = \max_{1\leq i\leq k} \|\bar{B}_i\| + 1, \tag{35}$$

then the gradient norms $\{\|g_k\| : \ k = 1, 2, 3, \ldots\}$ cannot be bounded away from zero, that is

$$\lim_{k\to\infty} \inf\|g_k\| = 0. \tag{36}$$

Proof By Eq. (25), we have

$$\|g_k\|^2 = \|u_{k-1}\|^2 + \rho_k^2, \tag{37}$$

where $u_{k-1} = Z_{k-1}^T g_k$. If $\rho_k > \nu\|g_k\|$, g_k is accepted and $g_k \in \text{Span}\{Z_k\}$, so we have $g_k = Z_k Z_k^T g_k = Z_k \bar{g}_k$ and $\|g_k\| = \|\bar{g}_k\|$; Otherwise g_k is rejected and we have $\rho_k \leq \nu\|g_k\|$, by Eqs. (29) and (37) and $0 \leq \nu < 1$, we get

$$\|\bar{g}_k\| = \|u_{k-1}\| = \sqrt{\|g_k\|^2 - \rho_k^2} \geq \sqrt{1-\nu^2}\|g_k\|, \tag{38}$$

Eq. (38) is also true when $\|g_k\| = \|\bar{g}_k\|$, therefore Eq. (38) always holds. It follows from Eqs. (33) and (38) that

$$-\bar{\phi}(\bar{s}_k) \geq \tau\|g_k\| \min\left(\Delta_k, \frac{\|g_k\|}{\|\bar{B}_k\|}\right). \tag{39}$$

where $\tau = 1/2(1-\nu^2) > 0$ is a positive constant. Then in the same way as Powell [21] and Nocedal and Yuan [17], we can prove Eq. (36) by contradiction. □

The lingering technique has been proposed by Siegel [23] and Gill and Leonard [12] for line search methods. If we choose $0 < \nu < 1$ in the above algorithm, and $\rho_{k+1} \le \nu \|g_{k+1}\|$ such that g_{k+1} is rejected, we are in fact lingering on the subspace $\mathrm{Span}\{Z_k\}$ minimizing $F(z) = f(x_k + Z_k z)$ by a traditional trust region method on that space. Hence, lingering steps in the subspace will not loop infinitely unless the minimizer lies in the manifold $x_k + \mathrm{Span}\{Z_k\}$.

In the above algorithm, we could also use the reinitialization technique proposed by Liu and Nocedal [15] and Gill and Leonard [12]. For reinitialization, as proposed by Gill and Leonard [12], we only need to compute an appropriate value σ_k and replace σ in Eq. (28) with σ_k when g_{k+1} is accepted in Step 7 of Algorithm 2. We have considered the following seven alternatives, i.e.,

$$\begin{aligned} &\sigma_k^{r0} = 1, \quad \sigma_k^{r1} = \frac{y_0^T y_0}{s_0^T y_0}, \quad \sigma_k^{r2} = \frac{s_k^T y_k}{\|s_k\|^2}, \\ &\sigma_k^{r3} = \min_{1 \le i \le k} \left\{ \frac{s_i^T y_i}{\|s_i\|^2} \right\}, \quad \sigma_k^{r4} = \frac{y_k^T y_k}{s_k^T y_k}, \\ &\sigma_k^{r5} = \frac{s_0^T y_0}{s_0^T s_0}, \quad \sigma_k^{r6} = \min_{1 \le i \le k} \left\{ \frac{y_i^T y_i}{s_i^T y_i} \right\}. \end{aligned} \tag{40}$$

By numerical experiments, we find out that in the trust region framework, only considering the reinitialization alone, the performance of the algorithm is not satisfactory and cannot be favorably compared with the algorithm without reinitialization. It is necessary to combine reinitialization with an appropriate implementation of the lingering technique to obtain better numerical results. We propose the following procedure for trust region methods, which combines the ideas proposed in Siegel [23] and Gill and Leonard [12] for line search methods. Suppose g_k is accepted in the last iteration (which means $Z_k = [Z_{k-1}, z_k]$), and we have obtained $\bar{s}_k = (\bar{s}_1^{(k)}, \bar{s}_2^{(k)}, \ldots, \bar{s}_r^{(k)})$ by solving Eqs. (14) and (15), where r is the number of columns of Z_k, which is the orthonormal basis matrix of $\mathrm{Span}\{g_1, \ldots, g_k\}$. If $|\bar{s}_r^{(k)}| \le \mu_1 \|\bar{s}\|$ and $\rho_k / \|g_k\| \le \mu_2$, where $1 > \mu_2 > 0.2 > \mu_1 > 0$, then we think that although g_k is accepted in the last iteration, the direction z_k does not contribute sufficiently to s_k compared with Z_{k-1}. So we can discard z_k and linger on the subspace $\mathrm{Span}\{Z_{k-1}\}$. Then we compute $x_{k+1} = x_k + s_k$ and $g_{k+1} = \nabla f(x_{k+1})$. Suppose g_{k+1} is accepted in Step 5, then $Z_{k+1} = [Z_{k-1}, z_{k+1}]$. In this way, the dimension of $\mathrm{Span}\{Z_k\}$ will not increase too quickly and we can reduce the amount of computation per iteration. If we set $\mu_1 = 0$ and z_k is discarded, we are in fact minimizing f in the subspace $\mathrm{Span}\{Z_{k-1}\}$ as the iteration proceeds. Thus, unless the minimizer lies in the manifold $x_{k-1} + \mathrm{Span}\{Z_{k-1}\}$, lingering steps will not loop infinitely.

Algorithm 3 The Subspace Quasi-Newton Trust Region Algorithm with Lingering and Reinitialization

Step 1. Given $1 > \mu_2 > 0.2 > \mu_1 > 0$ and some positive integer $m > 1$. Other parameters are the same as Step 1 of Algorithm 2.

Step 2–Step 5. The same as Step 2–Step 5 of Algorithm 2.

Step 6. If $|\bar{s}_r^{(k)}| \le \mu_1 \|\bar{s}_k\|$, g_k is accepted in last iteration and $\rho_k < \mu_2 \|g_k\|$, discard z_k and set $Z_k = Z_{k-1}$. Otherwise, Z_k is not changed.

Step 7. The same as Step 6 of Algorithm 2.

Step 8. If g_{k+1} is accepted (i.e., $\rho_{k+1} > \nu\|g_{k+1}\|$), compute σ_k according to one formula in Eq. (40) and replace σ in Eq. (28) with σ_k. Others are the same as Step 7 of Algorithm 2.
Step 9. The same as Step 8 of Algorithm 2.

Algorithm 3 now is quite different from the traditional trust region algorithms. However, we can prove the global convergence of Algorithm 3 in the same way as Theorem 3.1.

4 Numerical results

In this section, Algorithm 2 is referred to the one with reinitialization, where we only replace σ in Eq. (28) with the reinitialization parameter σ_k when g_{k+1} is accepted in Step 7 of Algorithm 2. We have implemented Algorithm 2 and Algorithm 3 with different reinitialization parameters defined in Eq. (40), and compared them with a traditional trust region algorithm whose framework is the same as Algorithm 2 except that Eqs. (2) and (3) are solved instead of Eqs. (14) and (15), where B_k is updated in the whole space. The matrix updating formulae we used are BFGS and SR1. When the BFGS formula is adopted to update the approximate Hessian B_k at each iteration, Algorithm 2 and Algorithm 3 with the reinitialization parameter $r0$–$r6$ defined in Eq. (40) are referred to “Sub1BFGSr0”–“Sub1BFGSr6” and “Sub2BFGSr0”–“Sub2BFGSr6”, respectively, and the corresponding traditional trust region algorithm is referred to “TrBFGS”. The matrix update is skipped for the above implementations if $s_k^T y_k \leq 10^{-12}\|s_k\|\|y_k\|$. Hence there are 15 implementations when BFGS is used, which are “Sub1BFGSr0”–“Sub1BFGSr6”, “Sub2BFGSr0”–“Sub2BFGSr6” and “TrBFGS”. When the SR1 formula is adopted to update the approximate Hessian B_k at each iteration, Algorithm 2 and Algorithm 3 with the reinitialization parameter $r0$–$r6$ defined in Eq. (40) are referred to “Sub1SR1r0”–“Sub1SR1r6” and “Sub2SR1r0”–“Sub2SR1r6”, respectively, and the corresponding traditional trust region algorithm is referred to “TrSR1”. The matrix update is skipped for these algorithms if $\|r_k r_k^T / r_k^T s_k\| \leq 10^{-12}$ according to [4], where $r_k = y_k - B_k s_k$. Therefore, there are also 15 implementations when SR1 is used, which are “Sub1SR1r0”–“Sub1SR1r6”, “Sub2SR1r0”–“Sub2SR1r6”, and “TrSR1”.

In all the 30 implementations mentioned above, we calculate the trial step using the subroutine GQTPAR which is designed based on the ideas described in Moré and Sorensen [16] with the parameters rtol $= 0.1$, atol $= 10^{-16}$. For all the 30 implementations, the parameters in Step 1 is chosen as $\Delta_1 = 1$, $\Delta_{\min} = 10^{-15}$, $\varepsilon_M = 10^{-20}$, $\varepsilon = 10^{-5}$, $\nu = 10^{-8}$, $\tau_0 = 0$, $\tau_1 = 0.25$, $\tau_2 = 0.75$, $c_1 = 10^{-5}$, $c_2 = 0.22$, $c_3 = 0.5$ and $c_4 = 2$, and we set $m = 10$, $\mu_1 = 0.1$, $\mu_2 = 0.8$ for Algorithm 3. Therefore, all the implementations are terminated if $\|g_k\| < 10^{-5}$, or if the trust region radius is too small, i.e., $\Delta_k < 10^{-15}$ at Step 3, or if the minimum of the quadratic subproblem is too close to zero, i.e., $|\bar{\phi}(\bar{s}_k)| \leq \varepsilon_M = 10^{-20}$ at Step 2. The algorithms were coded in C language using double precision, and the tests were performed on SGI O3800 with one CPU and the memory is limited to 800 MB.

We use the performance profile proposed by Dolan and Moré [9] to display the performance of each implementation on the set of test problems, which has some advantages over other existing benchmarking tools, especially for large test sets where tables tend to be overwhelming. Let $t_{p,s}$ denote the time to solve problem p by solver s. Define the performance ratio as

$$r_{p,s} = \frac{t_{p,s}}{t_p^*},$$

where t_p^* is the lowest time required by any solver to solve problem p. Therefore, $r_{p,s} \geq 1$ for all p, s. If a solver does not solve a problem, the ratio $r_{p,s}$ is assigned a large number M, which satisfies $r_{p,s} < M$ for all p, s where solver s succeeds in solving problem p. Then the performance profile for each code s is defined as the cumulative distribution function for the performance ratio $r_{p,s}$, which is

$$\rho_s(\tau) = \frac{\text{no. of problems s.t. } r_{p,s} \leq \tau}{\text{total no. of problems}}.$$

If $\tau < 1$ we have $\rho_s(\tau) = 0$. If $\tau = 1$, then $\rho_s(1)$ represents the percentage of problems for which the solver s's runtime is the best. We must note that when the sets of solvers being compared are different, t_p^* would be different, which would lead to different performance profiles for the same solver which is in different sets of solvers being compared, although the set of test problems is the same one. So we can not compare performance profiles of solvers which are in different sets of solvers. For more details about the performance profile please see Dolan and Moré [9]. The performance profile will also be used to analyze the number of objective function evaluations and gradient evaluations required by the above implementations. For each implementation on each test problem, if the computed $\|g_k\|_2 < 10^{-5}$ or $|f_k - f^*| < 10^{-8}$, where f^* is a local optimal objective function value, then the implementation is regarded successful in solving the problem. Otherwise, the implementation is regarded failed.

The test problems are chosen from CUTEr (see [1,14]) by the *selection* tool. There are 158 unconstrained test problems in the current CUTEr (April 7, 2004). Problems *fletchbv* and *indef* seem unbounded below even for $n = 100$, where n is the number of variables. There are more than one local optimal point for problems *vibrbeam* $(n = 8)$ and *broydn7d* $(n = 500)$, and different implementations get different local optimal points. None of the implementations can solve the problem *jimack* $(n = 1000)$ in 1 h. These five problems are excluded, and left 153 problems suitable for testing.

4.1 Numerical results on problems with medium size

We first test problems whose dimension are not bigger than 1,000. There are 69 problems with $n < 50$ (the number of variables is in the parenthesis):

akiva(2), allinitu(4), bard(3), beale(2), biggs6(6), box3(3), brkmcc(2), brownbs(2), brownden(4), cliff(2), cube(2), denschna(2), denschnb(2), denschnc(2), denschnd(3), denschne(3), denschnf(2), djtl(2), edensch(36), engval2(3), expfit(2), growthls(3), gulf(3), hairy(2), hatfldd(3), hatflde(3), heart6ls(6), heart8ls(8), helix(3), hielow(3), hilberta(2), himmelbb(2), himmelbf(4), himmelbg(2), himmelbh(2), humps(2), jensmp(2), kowosb(4), loghairy(2), maratosb(2), mexhat(2), meyer3(3), osbornea(5), palmer1c(8), palmer1d(7), palmer2c(8), palmer3c(8), palmer4c(8), palmer5c(6), *palmer6c(8), palmer7c(8), palmer8c(8), pfit1ls(3), pfit2ls(3), pfit3ls(3), pfit4ls(3), rosenbr(2), s308(2), sineval(2), sisser(2), snail(2), yfitu(3), zangwil2(2), 3pk(30), hilbertb(10), osborneb(11), parkch(15), stratec(10) , watson(12),*

and 55 problems with $50 \leq n < 500$:

chnrosnb(50), deconvu(61), errinros(50), hydc20ls(99), tointgor(50), tointpsp(50), tointqor(50), vareigvl(50), arglina(200), arglinb(200), arglinc(200), brownal(200), chainwoo(100), cosine (100), curly10(100), curly20(100),curly30(100), dixmaana(300),dixmaanb(300), dixmaanc(300), dixmaand(300), dixmaane(300), dixmaanf(300), dixmaang(300), dixmaanh(300), dixmaani(300), dixmaanj(300), dixmaank(300), dixmaanl(300), dixon3dq(100), eigenals(110), eigenbls(110), eigencls(462), engval1(100), extrosnb(100), fletcbv2(100), fletcbv3(100), fletchcr(100), fminsrf2(121), fminsurf(121), mancino(100), modbeale(200), ncb20(110), noncvxu2(100), noncvxun (100), penalty3(200), scosine(100), scurly10(100), scurly20(100), scurly30(100), sensors(100), sparsine(100), sparsqur(100), vardim(200), woods(100),

and 29 problems with $500 \leq n \leq 1000$:

arwhead(500), bdqrtic(500), brybnd(500), cragglvy(500), dqdrtic(500), dqrtic(500),freuroth(500), genhumps(500), genrose(500), liarwhd(500), morebv(500), ncb20b(500), nondia(500), nondquar (500), nonmsqrt(529), penalty1(500), penalty2(500), powellsg(500), power(500), quartc(500), sbrybnd(500), schmvett(500), sinquad(500), srosenbr(500), tointgss(500), tquartic(500), tridia (500), eg2(1000), testquad(1000).

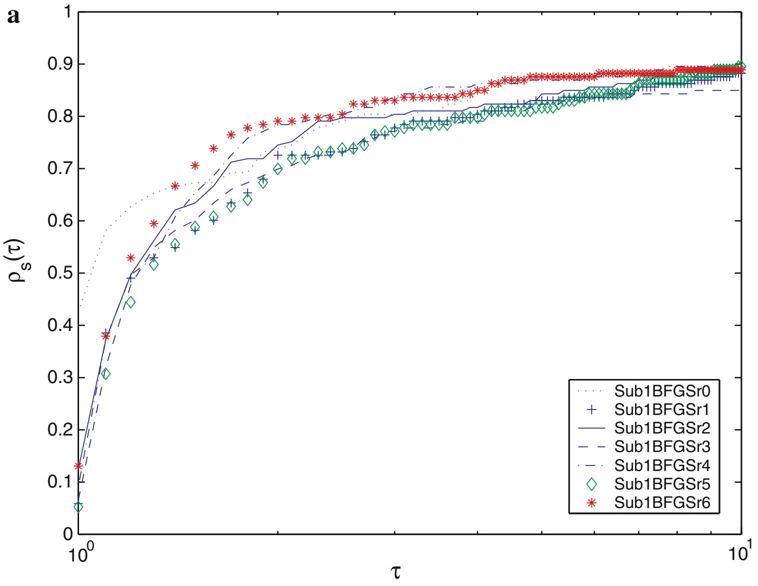
In Table 1, we listed problems on which none of the above 30 implementations can obtain a solution satisfying $\|g_k\|_2 < 10^{-5}$, and all the implementations are regarded failed on these problems. Nevertheless, they are included because it is useful to see that some problems are not solved by all the methods and the performance profile can take care of this without obscuring the other results according to [9].

First, Figs. 1 and 2 show the BFGS results for Algorithm 2 and Algorithm 3 with seven alternatives of σ_k (which are defined in Eq. (40)), where CPU time is used as the performance metric, and τ (which is the variable of the performance profile $\rho_s(\tau)$) is in the interval [1, 10], which should be enough for our interest to compare the performances of these different implementations. Figure 1 shows that reinitialization does not seem to do much for Algorithm 2. There is more of a spread for Algorithm 3 but Fig. 2 makes it clear that reinitialization combined with lingering is helpful. According to Fig. 1a, the performances of Algorithm 2 without reinitialization (i.e., $\sigma_k = \sigma_k^{r0}$) outperform Algorithm 2 with the reinitialization parameters σ_k^{r1}–σ_k^{r6} on about 65% problems, where our τ of interest is 1.5. Except σ_k^{r0}, the most preferred reinitialization parameter should be σ_k^{r6}, and σ_k^{r2} and σ_k^{r4} are the next two most preferred ones. In Fig. 1b, the implementations of Algorithm

Table 1 Hard problems (medium size) to solve

Name(Dim)	Func.Val.	$\|g\|_\infty$
HYDC20LS(99)	0.04175233	62.13303
NONMSQRT(529)	61.32027	0.01179786
PENALTY2(500)	5.380714e+39	3.955348e+16
PENALTY3(100)	0.0009999536	0.005382481
SCOSINE(100)	−99	3956.22
SCURLY10(100)	−10031.63	0.003027471
SCURLY30(100)	−10031.63	0.05164086

a



b

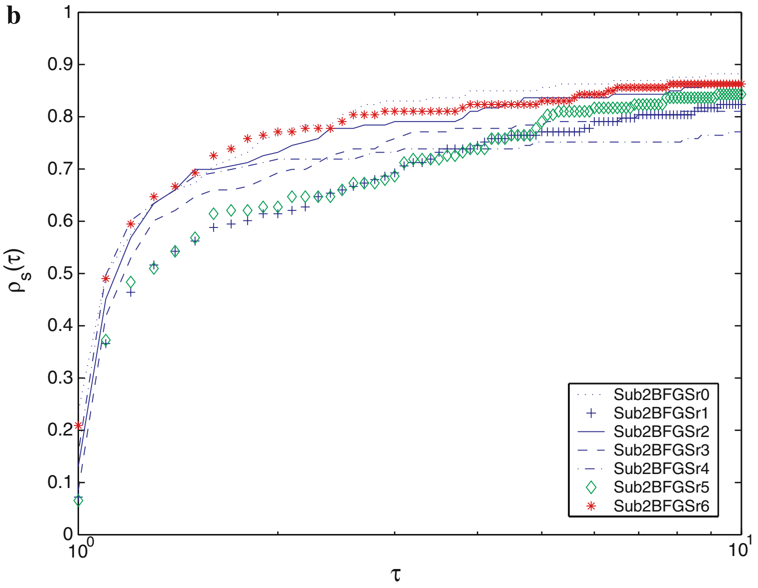


Fig. 1 Performance profile for CPU time on [1, 10] for (a) Algorithm 2 (b) Algorithm 3

3 using seven alternatives of σ_k (i.e. the seven implementations "Sub2BFGSr0"–"Sub2BFGSr0r6") are compared. We can see that the reinitialization parameters σ_k^{r6}, σ_k^{r2}, and σ_k^{r0} are the first three most preferred ones, and the differences among them are not strikingly obvious.

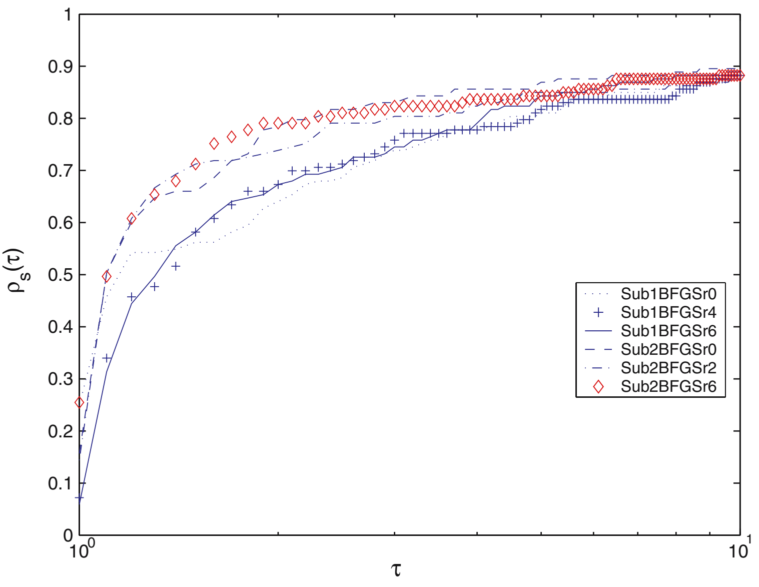


Fig. 2 Performance profile for CPU time on [1, 10]

Figure 1 indicates that “Sub1BFGSr0”, “Sub1BFGSr4”, and “Sub1BFGSr6” are the first three most preferred implementations for Algorithm 2, and “Sub2BFGSr6”, “Sub2BFGSr2”, and “Sub2BFGSr0” are the first three most preferred ones for Algorithm 3. Hence, the six implementations are compared in Fig. 2. We can see that when the BFGS formula is used, Algorithm 3 with reinitialization parameters σ_k^{r0} (which means no reinitialization), σ_k^{r2}, and σ_k^{r6} outperforms Algorithm 2 with reinitialization parameters σ_k^{r0}, σ_k^{r4}, and σ_k^{r6} on about 80% problems. The two implementations “Sub2BFGSr2” and “Sub2BFGSr6” are the preferred ones and the differences between them are not significant. Therefore, it would be helpful to combine reinitialization with lingering when BFGS is used, as pointed out in Gill and Leonard [12] for line search methods.

Figures 3 and 4 present the SR1 results, where the CPU time is used as the performance metric. Figure 3a shows that reinitialization does not seem to do much for Algorithm 2, and σ_k^{r0} is the most preferred one, which is similar to the case where the BFGS formula is used. The differences between σ_k^{r0} and σ_k^{r5} are not obvious, and σ_k^{r1}, σ_k^{r6} are the next two most preferred ones. In Fig. 3b, the seven implementations “Sub2SR1r0”–“Sub2SR1r0r6” are compared. We can see that the reinitialization parameter σ_k^{r0} is the most preferred one on about 50% problems, and σ_k^{r1} and σ_k^{r5} are the next two most preferred ones. Then the six implementations “Sub1SR1r0”, “Sub1SR1r5”, “Sub1SR1r6”, “Sub2SR1r0”, “Sub2SR1r5”, and “Sub2SR1r6” are selected to compare in Fig. 4. It shows that the framework of Algorithm 3 does not work so well with SR1 on this set of test problems. Solver “Sub1SR1r0” is the most preferred one on about 68% problems and implementation “Sub1SR1r5” is the most reliable one.

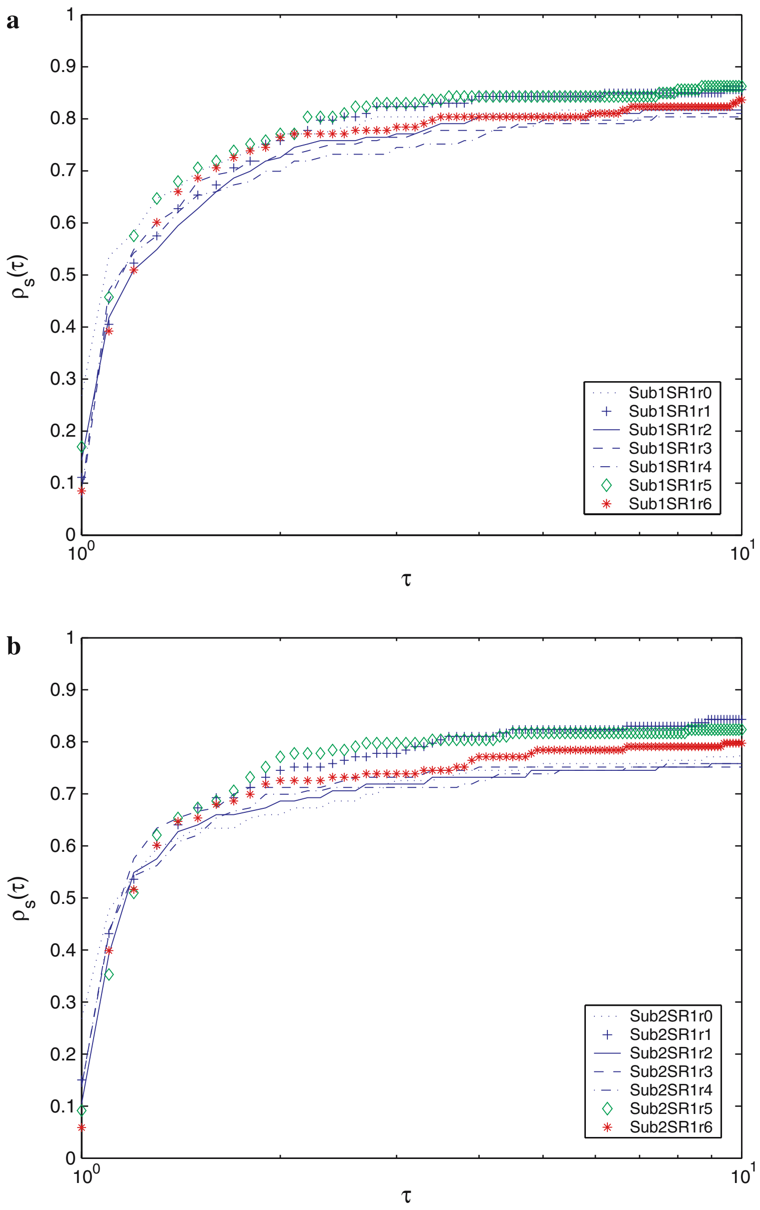


Fig. 3 Performance profile for CPU time on [1, 10] for **a** Algorithm 2 **b** Algorithm 3

Next, according to Figs. 2 and 3, the six implementations “Sub1BFGSr0”, “Sub1BFGSr4”, “Sub2BFGSr6”, “Sub1SR1r0”, “TrBFGS”, and “TrSR1” are selected to compare in Figs 5 and 6. When the number of objective function evaluations and the number of gradient evaluations are used as the performance metrics, Fig. 5 indicates that the implementation “Sub1SR1r0” is the most preferred one on

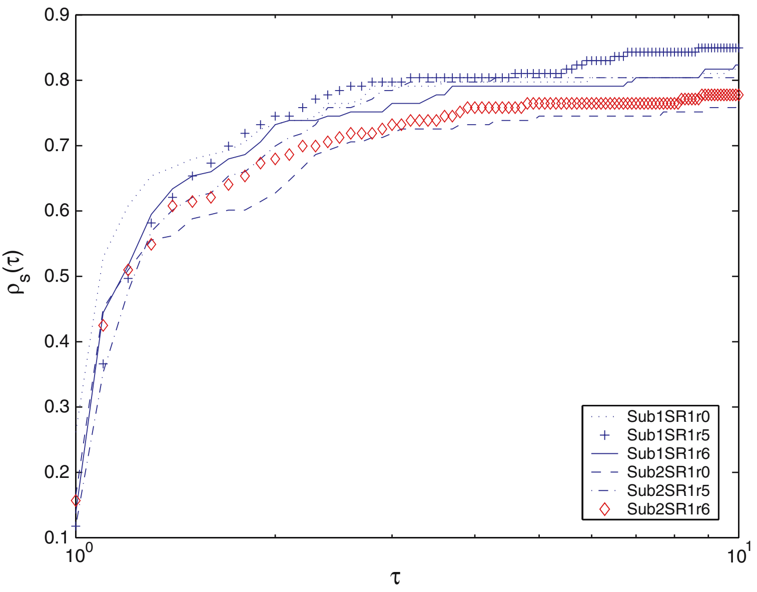


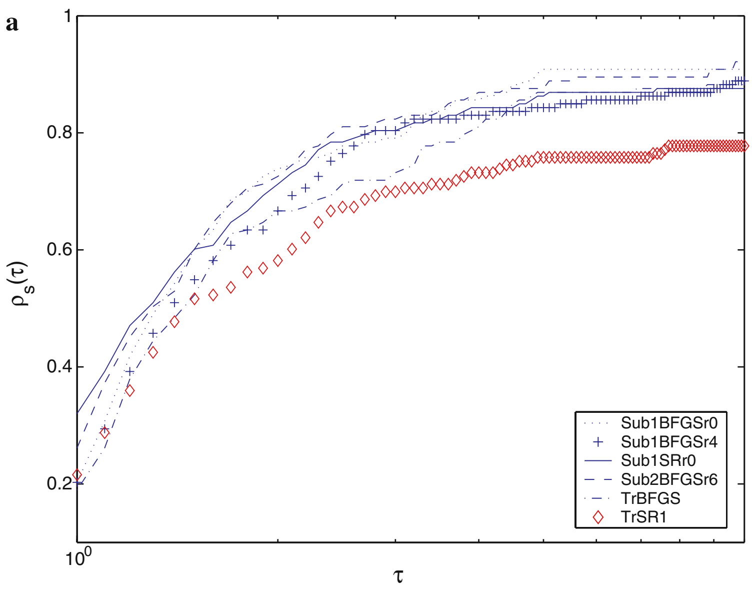
Fig. 4 Performance profile for CPU time on [1, 10]

about 60% problems, and “TrSR1” outperforms “TrBFGS” on about 50% problems. When CPU time is used as the performance metric, Fig. 6 indicates that the implementation “Sub2BFGSr6” is the most preferred one. The implementation “Sub1BFGSr0” outperforms “Sub1SR1r0” and “TrBFGS” outperforms “TrSR1” now. It tells us that when the SR1 formula is used, the CPU time needed to solve the quadratic subproblem will cost much more than that when the BFGS formula is used. In order to improve the performance of implementations where SR1 is used, perhaps we have to give a better estimation for the initial λ before calling the subroutine GQTPAR (see [16] for more details) to solve the quadratic subproblem, or other solvers for quadratic programming should be tried.

By comparing the two implementations “Sub1BFGSr0” and “Sub2BFGSr6” in Figs. 5 and 6, we can see that although “Sub1BFGSr0” outperforms “Sub2BFGSr6” in terms of function and gradient evaluations, “Sub2BFGSr6” wins over “Sub1BFGSr0” in CPU time. This tells us that when BFGS is used, we gained much in CPU time by using the reinitialization and lingering technique, which makes the dimension of subspaces increase more slowly. From the two figures, we can observe that Algorithm 2 which takes a subspace implementation outperforms the two implementations “TrBFGS” and “TrSR1” which are implemented in the full space, even when the number of objective function evaluations and the number of gradient evaluations are used as the performance metrics.

4.2 Numerical results on problems with more variables

Now we will test these implementations on problems with more variables. Because the performance of implementations with the reinitialization parameter



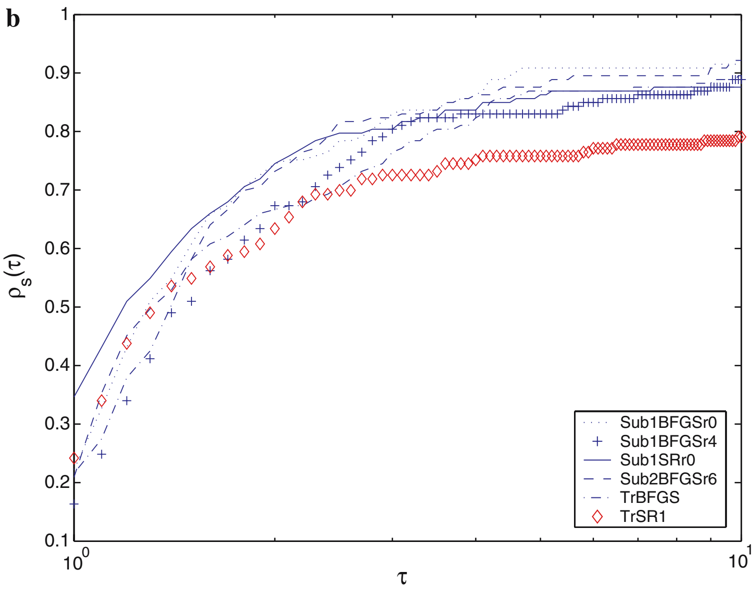


Fig. 5 Performance profile on [1, 10] for the number of **a** function evaluations and **b** gradient evaluations

σ_k^{r5} is almost the same as that with the reinitialization parameter σ_k^{r1}, the implementations with the reinitialization parameter σ_k^{r5} are not listed here in order to save space. For each implementation, an upper limit on the CPU time needed to solve any problem is set as half an hour (i.e. 1,800 s). If the CPU time needed to solve some problem is more than half an hour, the implementation is stopped and is regarded

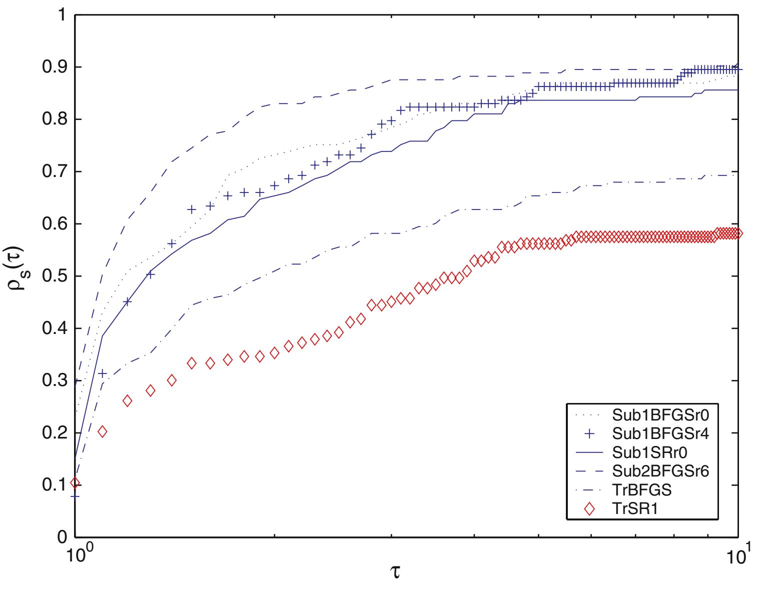
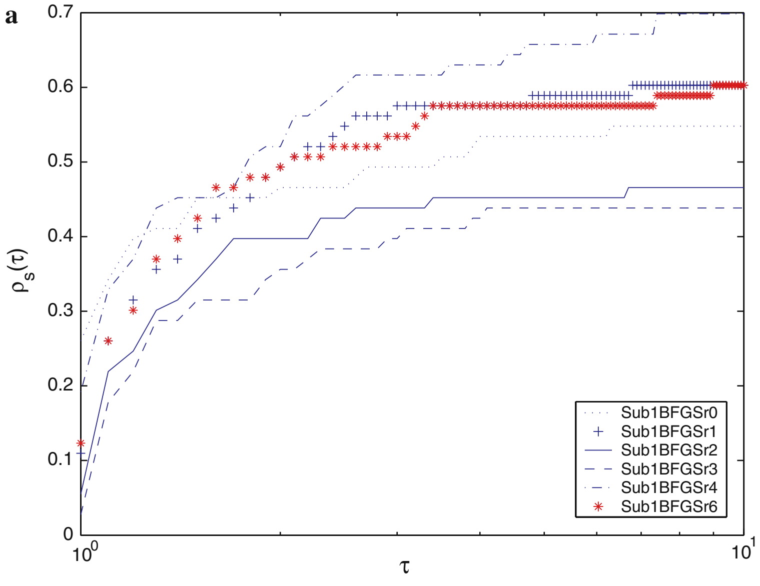


Fig. 6 Performance profile for CPU time on [1, 10]

failed. The following problems are excluded from comparing: (1) Problems that none of the implementations can solve in half an hour. (2) Problems whose dimension are less than 100.[1] (3)Problems on which all implementation's CPU time are less than 0.01 s. After excluding these problems, the following 73 problems are selected. Denote n as the number of variables. There are 10 problems with $100 \leq n \leq 500$: *arglina(200), arglinb(200), arglinc(200), brownal(200), fletchcr(100), mancino(100), penalty3(200), sensors(100), vardim(200), genrose(500)*; 44 problems with the $1000 \leq n \leq 3000$: *broydn7d(1000), chainwoo(1000), cosine(1000), curly10(1000), curly20(1000), curly30(1000), dixmaana(3000), dixmaanb(3000), dixmaanc(3000), dixmaand(3000), dixmaane (3000), dixmaanf(3000), dixmaang(3000), dixmaanh(3000), dixmaani(1500), dixmaanj(3000), dixmaank(1500), dixmaanl(1500), dixon3dq(1000), edensch(2000), eigenals(2550), eigenbls (2550), eigencls(2652), extrosnb(1000), fletcbv2(1000), fletcbv3(1000), modbeale(2000), ncb20b (1000), noncvxu2(1000), noncvxun(1000), nondquar(1000), nonmsqrt(1024), penalty1(1000), penalty2(1000), power(1000), sbrybnd(1000), scosine(1000), scurly10(1000), scurly20(1000), scurly30(1000), sparsqur(1000), testquad(1000), tridia(1000), woods(4000)*, and 19 problems with $n \geq 5000$: *arwhead(5000), brybnd(5000), bdqrtic(5000), cragglvy(5000), dqdrtic(5000), engval1(5000), fminsrf2(5625), freuroth(5000), liarwhd(5000), morebv(5000), ncb20 (5010), nondia(5000), powellsg(5000), schmvett(5000), sinquad(5000), sparsine (5000), srosenbr(5000), tointgss(5000), tquartic(5000)*.

On any problem listed above, every implementation terminated if $\|g_k\|_2 < 10^{-5}$ or $\|g_k\|_\infty < 10^{-6}$. In Table 2, we listed problems on which none of the implementations can obtain a solution satisfying $\|g_k\|_2 < 10^{-5}$ or $\|g_k\|_\infty < 10^{-6}$.

[1] Because the number of variables of about a half of the test problems in last subsection is less than 100, we choose 100 as the divide line.



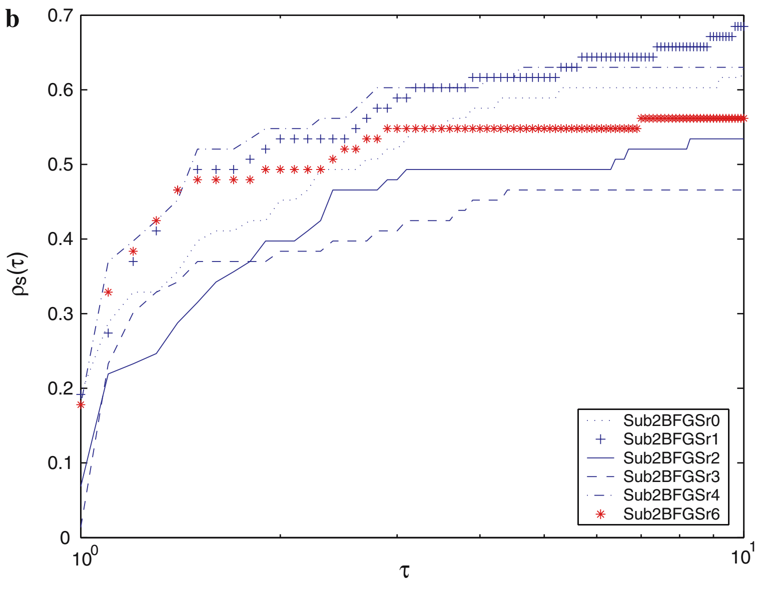


Fig. 7 Performance profile for CPU time on [1, 10] for **a** Algorithm 2 **b** Algorithm 3

Figures 7, 8, and 9 present the BFGS results on this set of test problems. From Fig. 7, the most preferred reinitialization parameter seems to be σ_k^{r4} for both Algorithm 2 and Algorithm 3 on most test problems, which is the same as the result obtained by Liu and Nocedal [15] and Gill and Leonard [12]. From Fig. 7b, the implementation "Sub2BFGSr1" seems to be the most robust one. We should

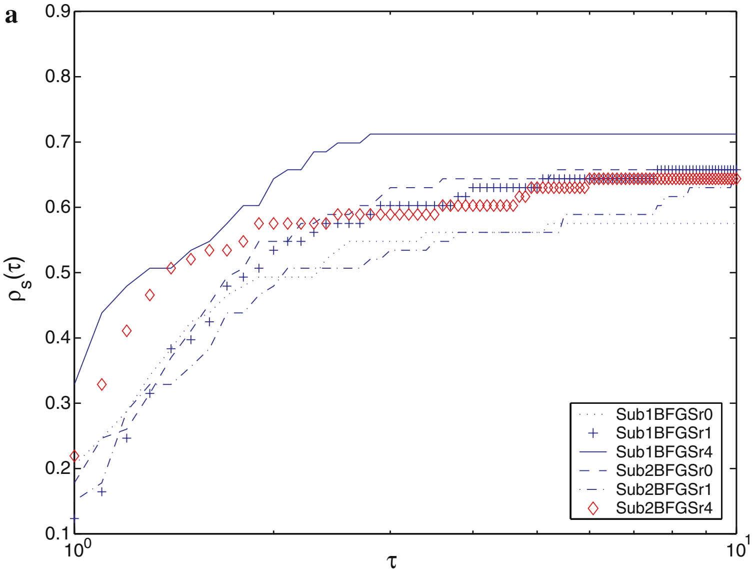
Table 2 Hard problems (large scale) to solve

Name(Dim)	Func.Val.	$\|g\|_\infty$
EIGENBLS(2550)	0.527474	0.2121526
EIGENCLS(2652)	399.6872	30.99925
FLETCBV3(1000)	−2.581354e+11	2.50299
FREUROTH(5000)	608,006.2	0.004854594
NONMSQRT(1024)	92.33829	0.01946075
PENALTY2(1000)	1.446399e+83	2.050731e+38
PENALTY3(200)	0.0009989562	0.08085646
SBRYBND(1000)	3,160.284	229,170.1
SCOSINE(1000)	−979.0675	4004.883
SCURLY10(1000)	−71435.48	1,013400
SCURLY20(1000)	−77530.27	95948.84
SCURLY30(1000)	−82,023.76	3,204,406
SPARSINE(5000)	0.04806616	0.174282

note that we can not compare “Sub1BFGSr4” with “Sub2BFGSr4” according to Fig. 7a , 7b, because the sets of solvers compared are different, which leads to different t_p^* and different performance profile for the same solver. According to Fig. 7, the six implementations “Sub1BFGSr0”, “Sub1BFGSr1”, “Sub1BFGSr4”, “Sub2BFGSr0”, “Sub2BFGSr1”, “Sub2BFGSr4” are selected to compare in Figs. 8 and 9, where the number of objective function evaluations, the number of gradient evaluations, and the CPU time are used as the performance metrics, respectively. In Fig. 8 the implementation “Sub1BFGSr4” outperforms “Sub2BFGSr4” and is the most preferred one in terms of the number of objective function and gradient evaluations. Figure 9 indicates that the implementation “Sub2BFGSr4” is the most preferred one, which outperforms “Sub1BFGSr4” in terms of CPU time. This tells us that we gained much in CPU time by using the lingering technique, which makes the dimension of subspaces increase more slowly. We could also see that the performance profile of “Sub1BFGSR4” of Fig. 9 looks a little different to that of Fig. 7, because the sets of solvers compared are different.

Figures 10, 11 and 12 present the SR1 results. Figure 10 indicates that the reinitialization parameter σ_k^{r1} is the most preferred one for both Algorithm 2 and Algorithm 3. In Fig. 11 and 12, the five implementations “Sub1SR1r0”, “Sub1SR1r1”, “Sub2SR1r1”, “Sub2SR1r3”, and “Sub2SR1r6” are compared. From Fig. 11 (where the number of objective function evaluations and the number of gradient evaluations are used as the performance metrics), the implementation “Sub1SR1r1” is the most preferred one. Figure 12 indicates that implementation “Sub2SR1r1” is the most preferred one in terms of CPU time, and it outperforms “Sub1SR1r1” on about 50% of the problems because of the lingering technique, although the difference between them is not strikingly obvious. This tells us that the technique of combining reinitialization with lingering is beneficial with SR1 on this set of test problems, which is contrary to what has been observed in Fig. 4.

Finally, according to Figs. 9 and 12, the seven implementations “Sub1BFGSr0”, “Sub1BFGSr4”, “Sub2BFGSr4”, “TrBFGS”, “Sub1SR1r0”, “Sub1SR1r1” and “TrSR1” are selected to compare in Figs. 13 and 14. From Fig. 13, the two implementations “Sub1SR1r1” and “Sub1BFGSr4” are the two preferred ones, where the number of objective function evaluations and gradient evaluations are used as performance metrics, and the implementation “Sub1SR1r1” outperforms



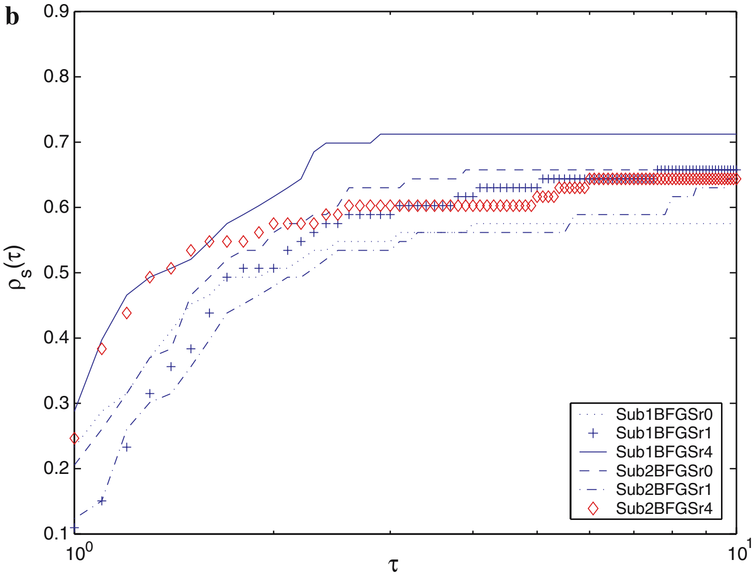


Fig. 8 Performance profile on [1, 10] for the number of **a** function evaluations and **b** gradient evaluations

"Sub1BFGSr4" on about 60% problems. The implementation "Sub1SR1r0" outperforms "Sub1BFGSr0", and "TrSR1" outperforms "TrBFGS", although the differences between them are not significant. When CPU time is used as the performance metric, Fig. 14 shows that the implementation "Sub2BFGSr4" becomes the most preferred one because of the lingering technique. "Sub1SR1r1"

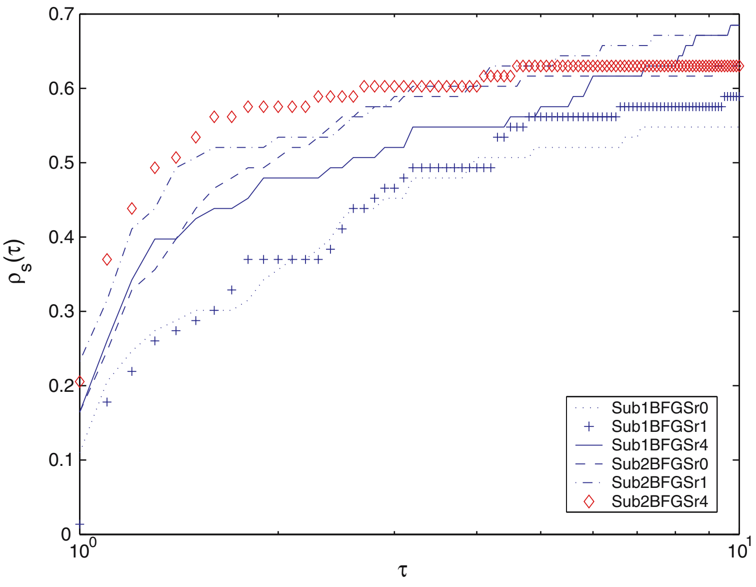


Fig. 9 Performance profile for CPU time on [1, 10]

outperforms “Sub1BFGSr4” on about 50% problems, and “Sub1SR1r0” outperforms “Sub1BFGSr0” on this set of test problems. If one takes robustness as the most important criteria, we can see from Figs. 13 and 14 that the implementation “Sub1BFGSr4” should be the most preferred one. All the five subspace implementations outperform the two solvers “TrBFGS” and “TrSR1” which are implemented in the whole space. By comparing Figs. 6 and 14, we can see that the subspace solvers will win more over the solvers “TrBFGS” and “TrSR1” when the test problems are large-scale. For example, “Sub1BFGSr0” solved the problem *arwhead* with $n = 5,000$ in six iterations, and the CPU time needed is only 0.69 s, but both “TrBFGS” and “TrSR1” failed to solve this problem in half an hour. For most large scale problems tested, the dimension r_k of the subspace spanned by $g_1, g_2, \ldots, g_k$ remains far less than the number of variables n when the problem is solved. For example, among the 30 problems with $n > 1,000$ on which the implementation “Sub1BFGSr4” succeeded, there are 25 problems solved with $r_k \leq 0.1n$ and 29 problems solved with $r_k \leq 0.2n$, where r_k is the dimension of the subspace.

From these numerical results, we can conclude that the subspace trust region methods are more efficient in implementation than the conventional trust region methods which are implemented in the whole space.

5 Conclusions

The subspace properties of trust region methods for unconstrained optimization are studied, provided that the approximate Hessian is updated by quasi-Newton formulae, and the initial Hessian approximation is appropriately chosen. Due

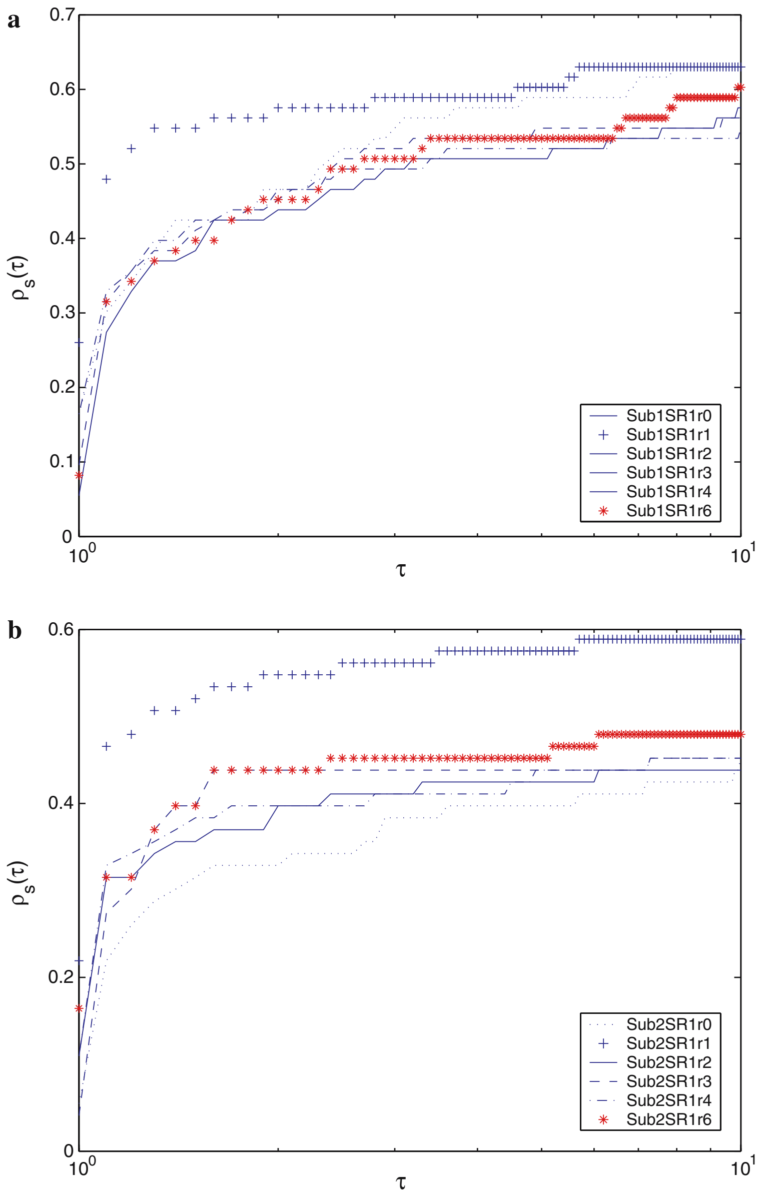
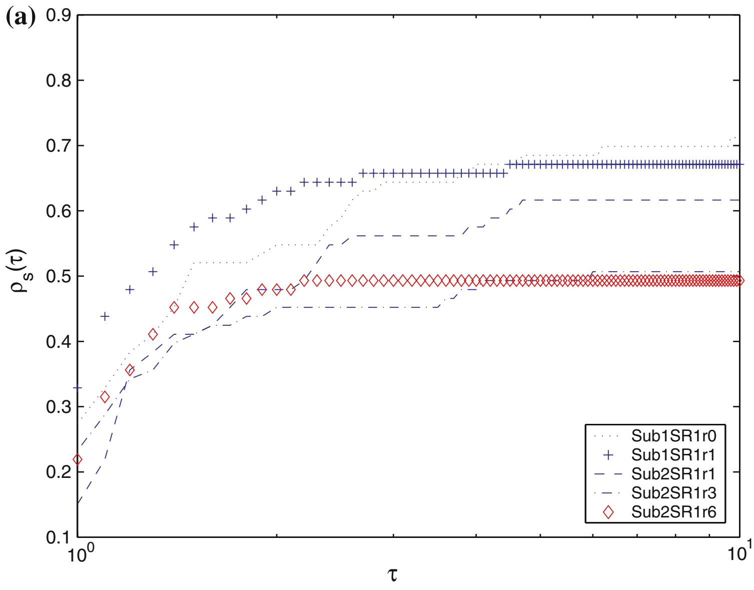


Fig. 10 Performance profile for CPU time on [1, 10] **a** Algorithm 2 **b** Algorithm 3

to the subspace properties, we can solve the quadratic subproblem with reduced Hessian and reduced gradient to obtain the trial step, and we can update the approximate reduced Hessian in the subspace. The equivalence of the subspace trust region methods with the traditional trust region methods is shown in Sect. 2. When the dimension of the subspace is much smaller than the number of variables, the



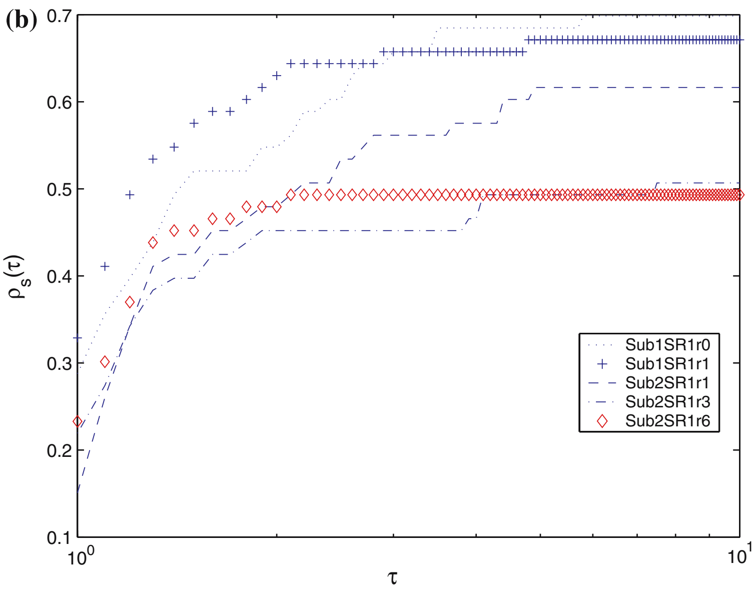


Fig. 11 Performance profile on [1, 10] for the number of **a** function evaluations and **b** gradient evaluations

amount of computation can be reduced significantly. Numerical results in Sect. 4 indicate that the subspace trust region methods not only gained in CPU time, but also require less function and gradient evaluations than the traditional trust region methods which are implemented in the whole space. When trust region methods

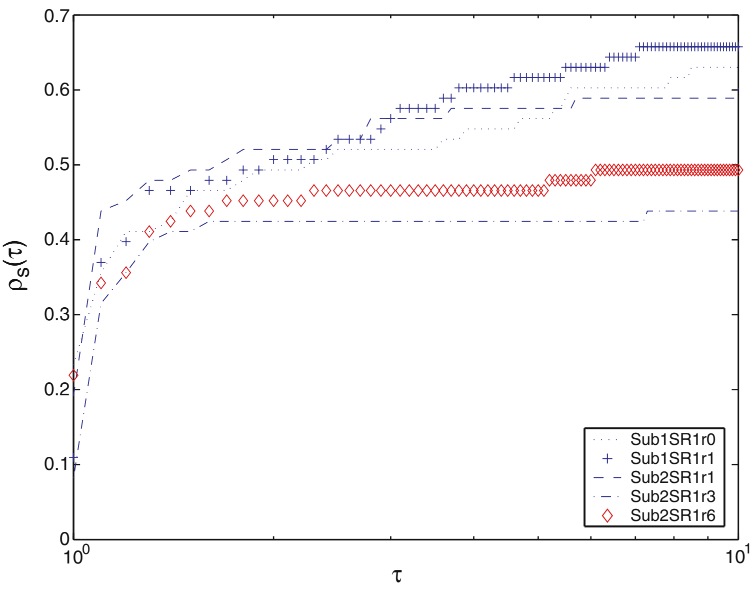
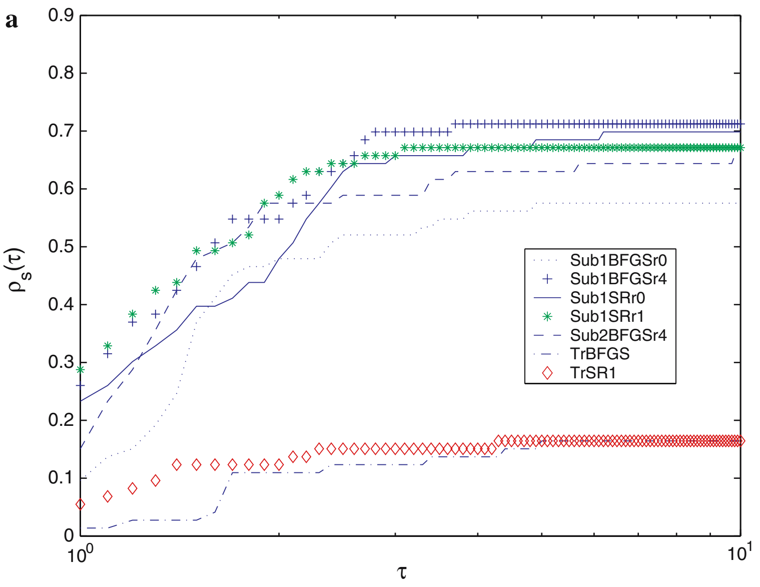


Fig. 12 Performance profile for CPU time on [1, 10]

are implemented in subspaces, CPU time is significantly reduced for large-scale problems.

We could also use the reinitialization technique conveniently in Algorithm 2. But only with reinitialization at each iteration, the numerical results are not as good as the case without reinitialization on the set of 153 test problems listed in Sect. 4.1, although reinitialization should reduce the influence of a poor initial estimate of the Hessian from intuition. But on the set of problems with more variables (i.e. problems listed in Sect. 4.2), the reinitialization technique becomes preferred, and the most preferred reinitialization parameter is $\sigma_k = \sigma_k^{r4} = y_k^T y_k / s_k^T y_k$ for BFGS which has been proposed in Liu and Nocedal [15] and Gill and Leonard [12], or $\sigma_k = \sigma_k^{r1} = y_0^T y_0 / s_0^T y_0$ for SR1.

By combining reinitialization with lingering technique, which has been proposed in Gill and Leonard [12] for line search methods, Algorithm 3 obtains better numerical results. When the BFGS formula is used, the the most preferred reinitialization parameter is $\sigma_k^{r6} = \min_{1\le i\le k}\{y_i^T y_i / s_i^T y_i\}$ on the set of 153 problems whose size ranging from small to medium (i.e., problems listed in Sect. 4.1), or $\sigma_k^{r4} = y_k^T y_k / s_k^T y_k$ on the set of test problems with more variables (i.e. problems listed in Sect. 4.2). When the SR1 formula is used, the framework of Algorithm 3, which uses reinitialization in conjunction with lingering, does not work so well on the set of 153 problems listed in Sect. 4.1. But on the set of test problems with more variables (i.e. problems listed in Sect. 4.2), Algorithm 3 with SR1 and the reinitialization parameter $\sigma_k^{r1} = y_0^T y_0 / s_0^T y_0$ becomes the most preferred one, which means that the technique of combining reinitialization with lingering would be beneficial for large-scale problems.



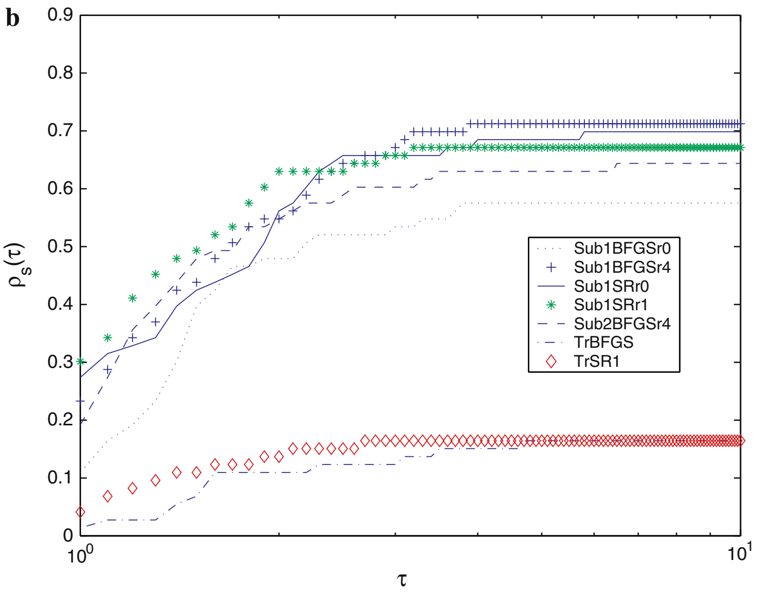


Fig. 13 Performance profile on [1, 10] for the number of **a** function evaluations and **b** gradient evaluations

From these numerical results, we can conclude that reinitialization and lingering techniques are useful for large-scale problems, but the effect is not favorably supported when the test problem's size is small and medium.

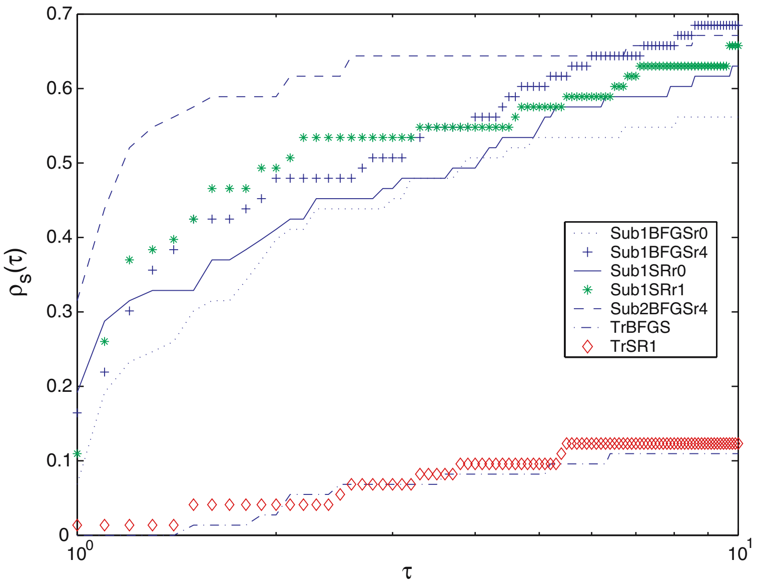


Fig. 14 Performance profile for CPU time on [1, 10]

Acknowledgements This work was supported by Chinese NSF grant 10231060 and the Personnel Fund of Beijing Jiaotong University LIJ05001.We would like to thank Professor A. R. Conn and another anonymous referee for their kind advice and modifications, which are very important in improving the quality of the paper.

References

1. Bongartz, I., Conn, A.R., Gould, N.I.M., Toint, Ph.L.: CUTE: constrained and unconstrained testing environment. ACM Trans. Math. Softw. **21**, 123–160 (1995)
2. Byrd, R., Schnabel, R.B., Shultz, G.A.: Approximation solution of the trust region problem by minimization over two-dimensional subspaces. Math. Prog. **40**, 247–263 (1988)
3. Conn, A.R., Gould, N.I.M., Sartenaer, A., Toint, Ph.L.: On the iterated-subspace minimization methods for nonlinear optimization with a combination of general equality and linear constraints. In: Adams, L., Nazareth, J.L. (eds.) Proceedings on Linear and Nonlinear Conjugate Gradient-Related Methods, pp. 50–78. SIAM (1996)
4. Conn, A.R., Gould, N.I.M., Toint, Ph.L.: Testing a class of methods for solving minimization problems with simple bounds on the variables. Math. Comput. **50**, 399–430 (1988)
5. Conn, A.R., Gould, N.I.M., Toint, Ph.L.: Trust-Region Methods. SIAM Publications, Philadelphia, Pennsylvania (2000)
6. Daniel, W., Gragg, W.B., Kaufman, L., Stewart, G.W.: Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization. Math. Comput. **30**, 772–795 (1976)
7. Dennis, J.E., Mei, H.H.W.: Two new unconstrained optimization algorithms which use function and gradient values. J. Optim. Appl. **28**, 453–482 (1979)
8. Dennis, J.E., Schnabel, R.B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Englewood Cliffs, NJ (1983)
9. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Prog. **91**, 201–213 (2002)
10. Fletcher, R.: Practical Methods of Optimization, 2nd edn. Wiley, New York (1987)
11. Gay, D.M.: Computing optimal locally constrained steps. SIAM J. Sci. Stat. Comput. **2**, 186–197 (1981)

12. Gill, P.E., Leonard, M.W.: Reduced-Hessian quasi-Newton methods for unconstrained optimization. SIAM J. Optim. **12**, 209–237 (2001)
13. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore (1996)
14. Gould, N.I.M., Orban, D., Toint, Ph.L.: CUTEr and SifDec: a constrained and unconstrained testing environment, revisited. ACM Trans. Math. Softw. **29**, 373–394 (2003)
15. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Math. Prog. **45**, 503–528 (1989)
16. Moré, J.J., Sorensen, D.C.: Computing a trust region step. SIAM J. Sci. Stat. Comput. **4**, 553–572 (1983)
17. Nocedal, J., Yuan, Y.: Combining trust region and line search techniques. In: Yuan, Y. (ed.) Advances in Nonlinear Programming, Proceedings of the 1996 International Conference on Nonlinear Programming, pp. 153–176, Kluwer, Dordrecht (1998)
18. Powell, M.J.D.: A new algorithm for unconstrained optimization. In: Rosen, J.B., Mangasarian, O.L., Ritter, K.(eds.) Nonlinear Programming, pp. 31–66. Academic, New York (1970)
19. Powell, M.J.D.: A hybrid method for nonlinear equations. In: Robinowitz, P. (ed.) Numerical Methods for Nonlinear Algebraic Equations, pp. 87–144. Gordon and Breach Science, London (1970)
20. Powell, M.J.D.: Convergence properties of a class of minimization algorithms. In: Mangasarian, O.L., Meyer, R.R., Robinson S.M. (eds.) Nonlinear Programming, 2, pp. 1–27. Academic, New York (1975)
21. Powell, M.J.D.: On the global convergence of trust region algorithms for unconstrained minimization. Math. Prog. **29**, 297–303 (1984)
22. Siegel, D.: Implementing and modifying Broyden class updates for large scale optimization. Report DAMPT 1992/NA12, University of Cambridge, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, England (1992)
23. Siegel, D.: Modifying the BFGS update by a new column scaling technique. Math. Prog. **66**, 45–78 (1994)
24. Sorensen, D.C.: Newton's method with a model trust region modifications. SIAM J. Numer. Anal. **19**, 409–426 (1982)
25. Steihaug, T.: The conjugate gradient method and trust regions in large scale optimization. SIAM J. Numer. Anal. **20**, 626–637 (1983)
26. Stewart, G.W.: Matrix Algorithms. Volume 1: Basic Decompostions. SIAM, Philadelphia, PA (1998)
27. Stoer, J., Yuan, Y.: A subspace study on conjugate gradient algorithms. ZAMM Z. Angew. Math. Mech. **75**, 69–77 (1995)
28. Vlc̃ek, J., Luks̃an, L.: New variable metric methods for unconstrained minimization covering the large-scale case. Technical Report No. V876, October 2002, Institute of Computer Science, Academy of Sciences of the Czech Republic (2002)
29. Yuan, Y.: Numerical Methods for Nonlinear Optimization. Shanghai Science and Technology Press, Shanghai (1994)
30. Yuan, Y.: A review of trust region algorithms for optimization. In: Ball, J.M., Hunt, J.C.R. (eds.) ICM99: Proceedings of the Fourth International Congress on Industrial and Applied Mathematics, pp. 271–282. Oxford University Press, Oxford (2000)
31. Yuan, Y.: On the truncated conjugate gradient method. Math. Prog. **87**, 561–571 (2000)