A BFGS trust-region method for nonlinear equations

Article *in* Computing · August 2011

DOI: 10.1007/s00607-011-0146-z · Source: DBLP

3 authors, including:

Gonglin Yuan
Guangxi University
102 PUBLICATIONS **1,675** CITATIONS

SEE PROFILE

Zengxin Wei
Guangxi University
75 PUBLICATIONS **1,902** CITATIONS

SEE PROFILE

Computing (2011) 92:317–333
DOI 10.1007/s00607-011-0146-z

A BFGS trust-region method for nonlinear equations

Gonglin Yuan · Zengxin Wei · Xiwen Lu

Received: 25 February 2010 / Accepted: 31 January 2011 / Published online: 16 February 2011

Abstract In this paper, a new trust-region subproblem combining with the BFGS update is proposed for solving nonlinear equations, where the trust region radius is defined by a new way. The global convergence without the nondegeneracy assumption and the quadratic convergence are obtained under suitable conditions. Numerical results show that this method is more effective than the norm method.

Keywords Trust region method · BFGS update · Global convergence · Nonlinear equations

Mathematics Subject Classification (2000) 65K05 · 90C26

1 Introduction

Consider the following system of nonlinear equations:

$$g(x) = 0, \quad x \in \Re^n, \tag{1.1}$$

This work is supported by China NSF grands 10761001, the Scientific Research Foundation of Guangxi University (Grant No. X081082), and Guangxi SF grands 0991028.

G. Yuan (✉) · Z. Wei
College of Mathematics and Information Science, Guangxi University,
Nanning, 530004 Guangxi, People's Republic of China
e-mail: glyuan@gxu.edu.cn

X. Lu
School of Science, East China University of Science and Technology,
200237 Shanghai, People's Republic of China

where $g : \Re^n \to \Re^n$ is continuously differentiable. The problem (1.1) has many applications in engineering, such as nonlinear fitting, function approximating and parameter estimating. Many algorithms have been presented for solving the problem (1.1), for examples, Gauss–Newton method [1,9,12,15], Levenberg–Marquardt method [6,20,31], trust region method [3,21,30], quasi-Newton method [8,32], etc. For symmetric nonlinear equations whose Jacobian matrix $\nabla g(x)$ of $g(x)$ is symmetric for all $x \in \Re^n$, the Gauss–Newton-based BFGS methods are proposed by Li and Fukushima [10,11]. These years we also made a further study and got some results (see [19,22–29]).

Let φ be the norm function defined by $\varphi(x) = \frac{1}{2}\|g(x)\|^2$. Suppose that $g(x)$ has a zero, then the nonlinear equation problem (1.1) is equivalent to the following global optimization problem

$$\min \varphi(x), \quad x \in \Re^n. \tag{1.2}$$

For the traditional trust region methods, at each iterative point x_k, the trial step d_k is obtained by solving the following trust-region subproblem

$$\text{Minimize } p_k(d) \text{ such that } \|d\| \leq \triangle, \tag{1.3}$$

where $p_k(d) = \frac{1}{2}\|g(x_k) + \nabla g(x_k)d\|^2$. The trust region methods are globally and superlinearly convergent under the condition that $\nabla g(x^*)$(x^* is a solution of (1.1)) is nondegenerate, where nondegeneracy means nonsingularity. Nondegeneracy of $\nabla g(x^*)$ seems a too stringent requirement for the purpose of ensuring superlinear convergence. Then Zhang and Wang [31] give a new trust region method:

$$\text{Minimize } \phi_k(d) \text{ such that } c^p\|g(x_k)\|^\gamma, \tag{1.4}$$

where $\phi_k(d) = \frac{1}{2}\|g(x_k) + \nabla g(x_k)d\|^2, 0 < c < 1$, p is a nonnegative integer, and $0.5 < \gamma < 1$. The superlinear convergence is obtained under the local error bound condition which is weaker than the nondegeneracy assumption (see [20]). However, the global convergence also need the nondegeneracy of $\nabla g(x^*)$. Presently, there is no algorithm which has the property that the iterative sequence generated by the algorithm satisfies $\|g(x_k)\| \to 0$ without the assumption that $\nabla g(x^*)$ is nondegenerate. The trust-region method will be very useful with the situation when the exact Jacobian or Hessian computation is inexpensive or possible. However this case is very infrequent in many practices. It is not difficult to see that one common drawback of the above two methods is to compute the Jacobian matrix $\nabla g(x)$ at every iteration, which obviously increase the workload and time, especially for large-scale problems.

These observations motivate us to present a new method, which not only possesses the global convergence without the nondegeneracy assumption and the quadratic convergence under suitable conditions, but also does not compute the Jacobian matrix $\nabla g(x)$ at every iteration. Naturally we can use the update matrix generated by the quasi-Newton method instead of Jacobian matrix. In this paper, at each iterative point x_k, the trial step is obtained by solving the following subproblem

$$\text{Minimize } q_k(d) \text{ such that } \|d\| \leq \triangle_k, \tag{1.5}$$

where $q_k(d) = \frac{1}{2}\|g(x_k) + B_k d\|^2$, the radius of trust region $\triangle_k$ is defined by $\triangle_k = c^p\|g(x_k)\|$, $c \in (0, 1)$, p is a nonnegative integer, and B_k is generated by the following BFGS formula

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k{}^T}{y_k{}^T s_k}, \tag{1.6}$$

where $s_k = x_{k+1} - x_k$, $y_k = g(x_{k+1}) - g(x_k)$, x_{k+1} is the next iteration, and B_0 is an initial symmetric positive definite matrix. By $y_k = g(x_{k+1}) - g(x_k)$, we have the approximate relations

$$y_k = g(x_{k+1}) - g(x_k) \approx \nabla g(x_{k+1}) s_k. \tag{1.7}$$

Since B_{k+1} satisfies the secant equation $B_{k+1}s_k = y_k$, we have approximately

$$B_{k+1}s_k \approx \nabla g(x_{k+1}) s_k. \tag{1.8}$$

This means that B_{k+1} approximates ∇g_{k+1} along direction s_k. Here and throughout this paper, we use the following notations. $\|.\|$ denote the Euclidian norm of vectors or its induced matrix norm, and $g(x_k)$ is replaced by g_k.

This paper is organized as follows. In the next section, the algorithm is represented. In Sect. 3, we prove some convergent results. The numerical results of the method are reported in Sect. 4.

2 The new method

Let d_k^p be the solution of (1.5) corresponding to p. Then we define the actual reduction as

$$Ared_k(d_k^p) = \varphi(x_k + d_k^p) - \varphi(x_k), \tag{2.1}$$

and the predict reduction as

$$Pred_k(d_k^p) = q_k(d_k^p) - q_k(0). \tag{2.2}$$

Now we represent our algorithm for solving (1.1). The algorithm is given as follows.

- **Algorithm 1.**
  Initial: Given constants $\rho, c \in (0, 1)$, $p = 0$, $\epsilon > 0$, $x_0 \in \Re^n$, $B_0 \in \Re^n \times \Re^n$ is symmetric and positive definite. Let $k := 0$;
  Step 1: If $\|g_k\| < \epsilon$, stop. Otherwise, go to step 2;
  Step 2: Solve the trust region subproblem (1.5) with $\triangle = \triangle_k$ to get d_k;

Step 3: Calculate $Ared_k(d_k^p)$, $Pred_k(d_k^p)$, and the radio of actual reduction over predict reduction as

$$r_k^p = \frac{Ared_k(d_k^p)}{Pred_k(d_k^p)}. \tag{2.3}$$

If $r_k^p < \rho$, then we let $p = p + 1$, go to step 2. Otherwise, go to step 4;
Step 4: Let $x_{k+1} = x_k + d_k^p$, $y_k = g_{k+1} - g_k$, if $y_k^T d_k^p > 0$, update B_{k+1} by (1.6), otherwise let $B_{k+1} = B_k$.
Step 5: Set $k := k + 1$ and $p = 0$. Go to step 1.

Remark (i) In this algorithm, the procedure of "Step 2-Step 3-Step 2" is named as inner cycle.

(ii) It is well known that B_{k+1} will inherit the positive property of B_k if and only if the condition $s_k^T y_k > 0$ holds (see [4,30], etc.). Before we give the proof, the definition of linear dependence is formulated: A collection of vectors $a_1, a_2, \ldots, a_n$ is called linearly dependent if there exist $\pi_1, \pi_2, \ldots, \pi_n$, not all zero, such that $\sum_{j=1}^n \pi_j a_j = 0$. From the definition of the linear dependence, it is not difficult to deduce that $a_1 = \theta^* a_2$ ($\theta^* \neq 0$) if a_1 and a_2 are linearly dependent. Now the proof is presented as follows:

$\Rightarrow$ If B_{k+1} is symmetric and positive definite, we have $s_k^T B_{k+1} s_k > 0$. Then

$$\begin{aligned} s_k^T B_{k+1} s_k &= s_k^T \left[B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k{}^T}{y_k{}^T s_k} \right] s_k \\ &= s_k^T B_k s_k - \frac{s_k^T B_k s_k s_k^T B_k s_k}{s_k^T B_k s_k} + \frac{s_k^T y_k y_k{}^T s_k}{y_k{}^T s_k} \\ &= s_k^T y_k > 0. \end{aligned}$$

$\Leftarrow$ We will prove that $x^T B_{k+1} x > 0$ holds for arbitrary $x \neq 0$ by induction. It is easy to see that B_0 is symmetric and positive definite. For $k \geq 0$, we assume that B_k is symmetric and positive definite. With the Cholesky decomposition $B_k = LL^T$, let

$$a = L^T x, \quad b = L^T s_k,$$

then we get

$$\begin{aligned} x^T B_{k+1} x &= x^T \left(B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} \right) x + x^T \frac{y_k^T y_k}{s_k^T y_k} x \\ &= \left[a^T a - \frac{(a^T b)^2}{b^T b} \right] + \frac{(x^T y_k)^2}{s_k^T y_k}. \end{aligned} \tag{2.4}$$

It follows that from the Cauchy inequality

$$a^T a - \frac{(a^T b)^2}{b^T b} \geq 0. \tag{2.5}$$

By $s_k^T y_k > 0$, we have

$$x^T B_{k+1} x \geq 0.$$

Using $x \neq 0$, then $a^T a - \frac{(a^T b)^2}{b^T b} = 0 \Leftrightarrow a \| b \Leftrightarrow x \| s_k$. If $x \| s_k$, then $x = \beta s_k,\ \beta \neq 0$, which implies

$$\frac{(x^T y_k)^2}{s_k^T y_k} > 0.$$

Therefore, $x^T B_{k+1} x > 0$ holds. Overall, for $x \neq 0$, we can always obtain

$$x^T B_{k+1} x > 0,$$

this means that B_{k+1} is positive definite. It is not difficult to deduce that B_{k+1} is symmetric. The proof is complete. Then the Step 4 in Algorithm 1 ensures that the matrix sequence $\{B_k\}$ is symmetric and positive definite.

Similar to Moré [13], Yuan and Sun [30], or Zhang and Wang [31], we have the following result.

Lemma 2.1 *If d_k^p is the solution of (1.5), then*

$$-Pred_k(d_k^p) \geq \frac{1}{2} \| B_k g_k \| \min \left\{ \triangle_k, \frac{\| B_k g_k \|}{\| B_k \|^2} \right\}. \tag{2.6}$$

Proof Since d_k^p is the solution of (1.5), for any $\alpha \in [0, 1]$, we have

$$\begin{aligned} -Pred_k(d_k^p) &\geq -Pred_k \left(-\alpha \frac{\triangle_k}{\| B_k g_k \|} B_k g_k \right) \\ &= \alpha \triangle_k \| B_k g_k \| - \frac{1}{2} \alpha^2 \triangle_k^2 (B_k B_k g_k)^T (B_k B_k g_k) / \| B_k g_k \|^2 \\ &\geq \alpha \triangle_k \| B_k g_k \| - \frac{1}{2} \alpha^2 \triangle_k^2 \| B_k B_k \|. \end{aligned}$$

Therefore, we obtain

$$\begin{aligned} -Pred_k(d_k^p) &\geq \max_{0 \leq \alpha \leq 1} \left[\alpha \triangle_k \| B_k g_k \| - \frac{1}{2} \alpha^2 \triangle_k^2 \| B_k \|^2 \right] \\ &\geq \frac{1}{2} \| B_k g_k \| \min \left\{ \triangle_k, \frac{\| B_k g_k \|}{\| B_k \|^2} \right\}. \end{aligned}$$

The proof is complete.

3 Convergence analysis

This section will give some convergence results under the following assumption conditions.

Assumption i (A) Let the level set Ω

$$\Omega = \{x|\varphi(x) \leq \varphi(x_0)\} \tag{3.1}$$

be bounded.

(B) $g(x)$ is twice continuously differentiable on an open convex set Ω_1 containing Ω.

(C) The following relation

$$\|[\nabla g(x_k) - B_k]g_k\| = O(\|d_k^p\|) \tag{3.2}$$

holds.

(D) The matrices $\{B_k\}$ are uniformly bounded on Ω_1, which means that there exist positive constants $0 < M_0 \leq M$ such that

$$M_0 \leq \|B_k\| \leq M \quad \forall k. \tag{3.3}$$

Assumption i(B) implies that there exists $M_1 > 0$ such that

$$\|\nabla g(x_k)^T \nabla g(x_k)\| \leq M_1, \quad \forall k. \tag{3.4}$$

Now let us see the following lemmas.

Lemma 3.1 *If d_k^p is the solution of (1.5). Let Assumption i hold and $\{x_k\}$ be generated by Algorithm 1. Then*

$$|Ared_k(d_k^p) - Pred_k(d_k^p)| = O(\|d_k^p\|^2).$$

Proof By (2.1), (2.2), and Assumption i, we have

$$\begin{aligned}
|Ared_k(d_k^p) - Pred_k(d_k^p)| &= |\varphi(x_k + d_k^p) - q_k(d_k^p)| \\
&= \frac{1}{2}|\|g_k + \nabla g(x_k)d_k^p + O(\|d_k^p\|^2)\|^2 - \|g_k + B_k d_k^p\|^2| \\
&= |g_k^T \nabla g(x_k) d_k^p - g_k^T B_k d_k^p + O(\|d_k^p\|^2) + O(\|d_k^p\|^4)| \\
&\leq \|[\nabla g(x_k) - B_k]g_k\|\|d_k^p\| + O(\|d_k^p\|^2) + O(\|d_k^p\|^4) \\
&= O(\|d_k^p\|^2).
\end{aligned}$$

The proof is complete.

Lemma 3.2 *Let Assumption i hold and $\{x_k\}$ be generated by Algorithm 1. Then Algorithm 1 does not circle in the inner cycle infinitely.*

Proof If Algorithm 1 circles in the inner cycle at x_k infinitely, i.e., $p \to \infty, r_k^p < \rho$ and $c^p \to 0$. Obviously, $\|g_k\| \geq \epsilon$, otherwise the algorithm stops. Then we get $\|d_k^p\| \leq \triangle_k = c^p \|g_k\| \to 0$.

By Lemmas 2.1 and 3.1, we have

$$|r_k^p - 1| = \frac{|Ared_k(d_k^p) - Pred_k(d_k^p)|}{|Pred_k(d_k^p)|} \leq \frac{2O(\|d_k^p\|^2)}{\triangle_k \|B_k g_k\|} \to 0.$$

Therefore, for p sufficiently large

$$r_k^p \geq \rho, \tag{3.5}$$

this contradicts the fact $r_k^p < \rho$. This completes the proof.

Lemma 3.3 *Let Assumption i hold and* $\{x_k\}$ *be generated by Algorithm* 1. *Then* $\{x_k\} \subset \Omega$. *Moreover,* $\{\varphi(x_k)\}$ *converges.*

Proof By the definition of the algorithm, we have

$$r_k^p \geq \rho > 0. \tag{3.6}$$

Which together with Lemma 2.1 implies

$$\varphi(x_{k+1}) \leq \varphi(x_k) \leq \cdots \leq \varphi(x_0).$$

This means $\{x_k\} \subset \Omega$. Considering $\varphi(x_k) \geq 0$, then we conclude that $\{\varphi(x_k)\}$ converges. The proof is complete.

Theorem 3.1 *Let Assumption i hold,* $\{x_k\}$ *be generated by the Algorithm* 1. *Then the algorithm either stops finitely or generates an infinite sequence* $\{x_k\}$ *such that*

$$\lim_{k\to\infty} \|g_k\| = 0. \tag{3.7}$$

Proof Assume that the algorithm does not stop after finitely many steps. Now we suppose that the following relation

$$\lim_{k\to\infty} \|B_k g_k\| = 0 \tag{3.8}$$

is true. By (3.3), we can obtain (3.7). Thus, in order to get this lemma, we must show (3.8).

Assume, on the contrary, that there exists a constant $\varepsilon > 0$ and an subsequence $\{k_j\}$ satisfying

$$\|B_{k_j} g_{k_j}\| \geq \varepsilon. \tag{3.9}$$

Let $K = \{k \mid \|B_k g_k\| \geq \varepsilon\}$. Meanwhile, by Assumption i and $\|B_k g_k\| \geq \varepsilon (k \in K)$, $\|g_k\| (k \in K)$ is bounded away from 0. Without loss of generality, we can assume $\|g_k\| \geq \varepsilon, \forall k \in K$.

Using the definition of Algorithm 1 and Lemma 2.1, we get

$$\sum_{k\in K}[\varphi(x_k) - \varphi(x_{k+1})] \geq -\sum_{k\in K} \rho . Pred_k(d_k^{p_k}) \geq \sum_{k\in K} \rho . \frac{1}{2} \min\left\{c^{p_k}\varepsilon, \frac{\varepsilon}{M^2}\right\} .\varepsilon,$$

where p_k is the largest p value obtained in the inner circle at the iterative point x_k.

By Lemma 3.3, we know that $\{\varphi(x_k)\}$ is convergent, then we have

$$\sum_{k\in K} \rho . \frac{1}{2} \min\left\{c^{p_k}\varepsilon, \frac{\varepsilon}{M^2}\right\} .\varepsilon < +\infty.$$

Thus, $p_k \to +\infty$ as $k \to +\infty$ and $k \in K$. So we can assume that $p_k \geq 1$ for all $k \in K$.

According to the determination of $p_k (k \in K)$ in the inner circle, the solution d'_k corresponding to the following subproblem

$$\begin{aligned} \min \ \ & q_k(d) = \frac{1}{2}\|g(x_k) + B_k d\|^2 \\ \text{s.t.} \quad & \|d\| \leq c^{p_k - 1}\|g_k\|, \end{aligned} \tag{3.10}$$

is unacceptable. Let $x'_{k+1} = x_k + d'_k$, we have

$$\frac{\varphi(x_k) - \varphi(x'_{k+1})}{-Pred_k(d'_k)} < \rho. \tag{3.11}$$

From Lemma 2.1, we get

$$-Pred_k(d'_k) \geq \frac{1}{2} \min\left\{c^{p_k - 1}\varepsilon, \frac{\varepsilon}{M^2}\right\} .\varepsilon.$$

By Lemma 3.1, we obtain

$$\varphi(x'_{k+1}) - \varphi(x_k) - Pred_k(d'_k) = O(\|d'_k\|^2) = O(c^{2(p_k-1)}).$$

Therefore,

$$\left|\frac{\varphi(x'_{k+1}) - \varphi(x_k)}{Pred_k(d'_k)} - 1\right| \leq \frac{O(c^{2(p_k-1)})}{0.5 \min\{c^{p_k-1}\varepsilon, \frac{\varepsilon}{M^2}\}.\varepsilon}.$$

Since $p_k \to +\infty$ as $k \to +\infty$ and $k \in K$, we have

$$\frac{\varphi(x_k) - \varphi(x'_{k+1})}{-Pred_k(d'_k)} \to 1, \quad k \in K,$$

this contradicts (3.11). The proof is complete.

Theorem 3.1 says that the iterative sequence $\{x_k\}$ generated by Algorithm 1 satisfies $\|g(x_k)\| \to 0$ without the assumption that $\nabla g(x^*)$ is nondegenerate, where x^* is a cluster point of $\{x_k\}$.

In order to establish the quadratic convergence of Algorithm 1. The following assumption is further needed.

Assumption ii (A) $x_k \to x^*$, x^* is a solution of (1.1);
(B) There exist $b_0 \in (0, 1)$ and $c_0 > 0$ such that

$$\|\nabla g(y)(x - y) - g(x) + g(y)\| \le c_0\|x - y\|^2,$$
$$\forall\, x, y \in N(x^*, b_0) = \{x \mid \|x - x^*\| \le b_0\};$$

(C) There exists $c_1 \in (1, \infty)$ satisfying

$$c_1\|x - x^*\| \le \|g(x)\| = \|g(x) - g(x^*)\|, \quad \forall\, x \in N(x^*, b_0).$$

(D) B_k is a good approximation to $\nabla g(x_k)$, i.e.,

$$\|\nabla g(x_k) - B_k\| = O(\|x_k - x^*\|).$$

Assumption ii(B) holds when $g(x)$ is continuously differentiable and $\nabla g(x)$ is Lipschitz continuous. By Assumption ii(B), there exists $L > 0$ such that

$$\|g(x) - g(y)\| \le L\|x - y\|, \quad \forall\, x, y \in N(x^*, b_0). \tag{3.12}$$

Theorem 3.2 *Let Assumptions i and ii hold, $\{x_k\}$ be generated by the Algorithm* 1. *Then, for k sufficiently large the iteration formula is as follows*

$$x_{k+1} = x_k + d_k^0,$$

where d_k^0 is the solution of (1.5) *corresponding to $p = 0$, and*

$$\|x_{k+1} - x^*\| = O(\|x_k - x^*\|^2),$$

i.e., Algorithm 1 *is quadratically convergent.*

Proof By Assumption ii(A) and (C), we have that for k sufficiently large

$$\|x_k - x^*\| \le \|g_k\|.$$

Thus, $x^* - x_k$ is a feasible point of (1.5).

Since p starts from 0 at each iterative point x_k, $x^* - x_k$ is a feasible point of (1.5) corresponding to $p = 0$ for k sufficiently large. Hence it follows from Assumption ii(B)

and (D) that

$$\begin{aligned} q_k(d_k^0) \le q_k(x^* - x_k) &= \frac{1}{2}\|g(x_k) + B_k(x^* - x_k)\|^2 \\ &= \frac{1}{2}\|g(x_k) + B_k(x^* - x_k) - g(x^*)\|^2 \\ &= \frac{1}{2}\|g(x_k) - g(x^*) + \nabla g(x_k)(x^* - x_k) + (B_k - \nabla g(x_k))(x^* - x_k)\|^2 \\ &= O(\|x_k - x^*\|^4). \end{aligned}$$

Thus, we get

$$q_k(d_k^0) = \frac{1}{2}\|g(x_k) + B_k d_k^0\|^2 = O(\|x_k - x^*\|^4). \tag{3.13}$$

Using (1.5), we have

$$\|d_k^0\| \le \|g(x_k)\| = O(\|x_k - x^*\|).$$

Therefore, for k sufficiently large, we get $x_k + d_k^0 \in N(x^*, b_0)$. So

$$\begin{aligned} c_1\|x_k + d_k^0 - x^*\| &\le \|g(x_k + d_k^0)\| \\ &\le \|g(x_k) + \nabla g(x_k)d_k^0\| + O(\|d_k^0\|^2) \\ &= \|g(x_k) + B_k d_k^0 + \nabla g(x_k)d_k^0 - B_k d_k^0\| + O(\|d_k^0\|^2) \\ &\le O(\|x_k - x^*\|^2) + \|\nabla g(x_k) - B_k\|\|d_k^0\| + O(\|d_k^0\|^2) \\ &= O(\|x_k - x^*\|^2) + O(\|x_k - x^*\|)\|d_k^0\| + O(\|d_k^0\|^2) \\ &= O(\|x_k - x^*\|^2). \end{aligned}$$

Thus,

$$\|g(x_k + d_k^0)\| = O(\|x_k - x^*\|^2) \tag{3.14}$$

and

$$\|x_k + d_k^0 - x^*\| = O(\|x_k - x^*\|^2).$$

In the following we prove that for k sufficiently large the iteration formula is as follows

$$x_{k+1} = x_k + d_k^0. \tag{3.15}$$

For k large enough, (3.13) and (3.14) imply

$$\begin{aligned} |Ared_k(d_k^0) - Pred_k(d_k^0)| &= \left|\frac{1}{2}\|g(x_k + d_k^0)\|^2 - q_k(d_k^0)\right| \\ &\le \frac{1}{2}\|g(x_k + d_k^0)\|^2 + q_k(d_k^0) \\ &= O(\|x_k - x^*\|^4). \end{aligned} \tag{3.16}$$

By Assumption ii(C) and (3.13), we have

$$\begin{aligned} |Pred_k(d_k^0)| &= \left|q_k(d_k^0) - \frac{1}{2}\|g(x_k)\|^2\right| \\ &\ge \frac{1}{2}c_1^2\|x_k - x^*\|^2 + O(\|x_k - x^*\|^4) = O(\|x_k - x^*\|^2). \end{aligned} \tag{3.17}$$

Combining (3.16) and (3.17), we get

$$\lim_{k\to\infty} |r_k^0 - 1| = \lim_{k\to\infty} \frac{|Ared_k(d_k^0) - Pred_k(d_k^0)|}{|Pred_k(d_k^0)|} = 0.$$

Thus, $r_k^0 > \rho$ for k sufficiently large. So the iteration formula is (3.15) for k sufficiently large. Therefore, we get the result of the quadratic convergence.

4 Numerical results

In this section, we report results of some numerical experiments with the proposed method. We also give another algorithm where the trust region subproblem is defined by (1.4) and the predict reduction is

$$Pred_k(d_k^p) = \phi_k(d_k^p) - \phi_k(0),$$

and we call it Algorithm 2. We now list the test functions

$$g(x) = (f_1(x), f_2(x), \ldots, f_n(x))^T,$$

where these functions have the associated initial guess x_0. These functions are stated as follows.

Function 1 Logarithmic function

$$f_i(x) = \ln(x_i + 1) - \frac{x_i}{n}, \quad i = 1, 2, 3, \ldots, n.$$

Initial guess: $x_0 = (1, 1, \ldots, 1)^T$.

Function 2 Broyden Tridiagonal function ([7], pp. 471–472)

$$\begin{aligned} f_1(x) &= (3 - 0.5x_1)x_1 - 2x_2 + 1, \\ f_i(x) &= (3 - 0.5x_i)x_i - x_{i-1} + 2x_{i+1} + 1, \quad i = 2, 3, \ldots, n-1, \\ f_n(x) &= (3 - 0.5x_n)x_n - x_{n-1} + 1. \end{aligned}$$

Initial guess: $x_0 = (-1, -1, \ldots, -1)^T$.

Function 3 Strictly convex function 1 ([17], p. 29) $g(x)$ is the gradient of $h(x) = \sum_{i=1}^{n}(e^{x_i} - x_i)$.

$$f_i(x) = e^{x_i} - 1, \quad i = 1, 2, 3, \ldots, n$$

Initial guess: $x_0 = (\frac{1}{n}, \frac{2}{n}, \ldots, 1)^T$.

Function 4 Penalty function

$$\begin{aligned} f_i(x) &= \sqrt{10^{-5}}(x_i - 1), \quad i = 1, 2, 3, \ldots, n-1, \\ f_n(x) &= \left(\frac{1}{4n}\right)\sum_{j=1}^{n} x_j^2 - \frac{1}{4}. \end{aligned}$$

Initial guess: $x_0 = (\frac{1}{3}, \frac{1}{3}, \ldots, \frac{1}{3})^T$.

Function 5 Variable dimensioned function

$$\begin{aligned} f_i(x) &= x_i - 1, \quad i = 1, 2, 3, \ldots, n-2, \\ f_{n-1}(x) &= \sum_{j=1}^{n-2} j(x_j - 1), \\ f_n(x) &= \left(\sum_{j=1}^{n-2} j(x_j - 1)\right)^2. \end{aligned}$$

Initial guess: $x_0 = (1 - \frac{1}{n}, 1 - \frac{2}{n}, \ldots, 0)^T$.

Function 6 Extended Freudentein and Roth function (n is even) [2]. For $i = 1, 2, \ldots, n/2$

$$\begin{aligned} f_{2i-1}(x) &= x_{2i-1} + ((5 - x_{2i})x_{2i} - 2)x_{2i} - 13, \\ f_{2i}(x) &= x_{2i-1} + ((1 + x_{2i})x_{2i} - 14)x_{2i} - 29. \end{aligned}$$

Initial guess: $x_0 = (6, 3, 6, 3, \ldots, 6, 3)$.

Function 7 Discrete boundary value problem [14].

$$\begin{aligned} f_1(x) &= 2x_1 + 0.5h^2(x_1 + h)^3 - x_2, \\ f_i(x) &= 2x_i + 0.5h^2(x_i + hi)^3 - x_{i-1} + x_{i+1}, \quad i = 2, 3, \ldots, n-1 \\ f_n(x) &= 2x_n + 0.5h^2(x_n + hn)^3 - x_{n-1}, \\ h &= \frac{1}{n+1}. \end{aligned}$$

Initial guess: $x_0 = (h(h-1), h(2h-1), \ldots, h(nh-1))$.

Function 8 The discretized two-point boundary value problem similar to the problem in [16]

$$g(x) = Ax + \frac{1}{(n+1)^2} F(x) = 0,$$

when A is the $n \times n$ tridiagonal matrix given by

$$A = \begin{bmatrix} 8 & -1 & & & & \\ -1 & 8 & -1 & & & \\ & -1 & 8 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & -1 \\ & & & & -1 & 8 \end{bmatrix},$$

and $F(x) = (F_1(x), F_2(x), \ldots, F_n(x))^T$ with $F_i(x) = \sin x_i - 1$, $i = 1, 2, \ldots, n$, and $x = (50, 0, 50, 0, \ldots)$.

In the experiments, the parameters were chosen as $\rho = 0.0001$, $\epsilon = 10^{-5}$, $c = 0.1$, $\gamma = 0.7$, B_0 is the unit matrix. We obtain d_k by (1.5) and (1.4) from *Dogleg* method [18]. In the inner circle of Algorithm 1, we will accept the trial step if $p > 5$ holds. We also stop the program if the iteration number is larger than 1,500. The program was coded in MATLAB 7.0. The columns of the tables have the following meaning:

Dim: the dimension of the problem.
NI: the total number of iterations.
NG: the number of the norm function evaluations.

From Tables 1 and 2, we can see that our algorithm performs quite well from these problems, and the dimension does not influence the performance of the presented method obviously. Algorithm 2 can also successfully solve some of these problems, moreover, it is better than Algorithm 1 for some of them. However it fails to some problems.

Dolan and Moré [5] gave a new tool to analyze the efficiency of algorithms. They introduced the notion of a performance profile as a means to evaluate and compare the

Table 1 Test results for Algorithm 1

Functions	Dim	*NI*/*NG*	Functions	Dim	*NI*/*NG*
Function 1	$n = 10$	6/7	Function 5	$n = 10$	1/2
	$n = 100$	6/7		$n = 100$	1/2
	$n = 200$	6/7		$n = 200$	1/2
	$n = 600$	6/7		$n = 600$	1/2
Function 2	$n = 10$	70/86	Function 6	$n = 10$	453/859
	$n = 100$	88/149		$n = 100$	928/1,819
	$n = 200$	78/119		$n = 200$	1,145/2,231
	$n = 600$	85/116		$n = 600$	1,446/2,792
Function 3	$n = 10$	6/7	Function 7	$n = 10$	35/47
	$n = 100$	6/7		$n = 100$	41/59
	$n = 200$	6/7		$n = 200$	21/26
	$n = 600$	6/7		$n = 600$	20/29
Function 4	$n = 10$	11/12	Function 8	$n = 10$	37/55
	$n = 100$	2/49		$n = 100$	43/54
	$n = 200$	12/49		$n = 200$	46/59
	$n = 600$	12/49		$n = 600$	44/55

Table 2 Test results for Algorithm 2

Functions	Dim	*NI*/*NG*	Functions	Dim	*NI*/*NG*
Function 1	$n = 10$	$NI > 1{,}500$	Function 5	$n = 10$	1/2
	$n = 100$	$NI > 1{,}500$		$n = 100$	1/2
	$n = 200$	$NI > 1{,}500$		$n = 200$	1/2
	$n = 600$	$NI > 1{,}500$		$n = 600$	1/2
Function 2	$n = 10$	9/10	Function 6	$n = 10$	$NI > 1{,}500$
	$n = 100$	23/24		$n = 100$	$NI > 1{,}500$
	$n = 200$	26/27		$n = 200$	$NI > 1{,}500$
	$n = 600$	27/28		$n = 600$	$NI > 1{,}500$
Function 3	$n = 10$	24/25	Function 7	$n = 10$	2/3
	$n = 100$	25/26		$n = 100$	1/2
	$n = 200$	26/27		$n = 200$	1/2
	$n = 600$	26/27		$n = 600$	1/2
Function 4	$n = 10$	7/8	Function 8	$n = 10$	3/4
	$n = 100$	6/7		$n = 100$	3/4
	$n = 200$	6/7		$n = 200$	3/4
	$n = 600$	6/7		$n = 600$	3/4

performance of the set of solvers S on a test set P. Assuming that there exist n_s solvers and n_p problems, for each problem p and solver s, they defined $t_{p,s}$ = computing time (the number of function evaluations or others) required to solve problem p by solver s.

Requiring a baseline for comparisons, they compared the performance on problem p by solver s_0 with the best performance by any solver on this problem; that is, using the performance ratio

$$r_{p,s_0} = \frac{t_{p,s_0}}{\min\{t_{p,s} : s \in S\}}.$$

Suppose that a parameter $r_M \geq r_{p,s}$ for all p, s is chosen, and $r_{p,s} = r_M$ if and only if solver s does not solve problem p.

The performance of solver s on any given problem might be of interest, but we would like to obtain an overall assessment of the performance of the solver, then they defined

$$\rho_s(t) = \frac{1}{n_p} size\{p \in P : r_{p,s} \leq t\},$$

thus $\rho_s(t)$ was the probability for solver $s \in S$ that a performance ratio $r_{p,s}$ was within a factor $t \in \Re$ of the best possible ratio. Then function ρ_s was the (cumulative) distribution function for the performance ratio. The performance profile $\rho_s : \Re \mapsto [0, 1]$ for a solver was a nondecreasing, piecewise constant function, continuous from the right at each breakpoint. The value of $\rho_s(1)$ was the probability that the solver would win over the rest of the solvers.

According to the above rules, we know that one solver whose performance profile plot is on top right will win over the rest of the solvers.

Figures 1 and 2 show that the performance of these methods is relative to *NI*, *NG* of Tables 1 and 2, respectively. From these two figures it is clear that the given method has won (has the higher probability of being the optimal solver).

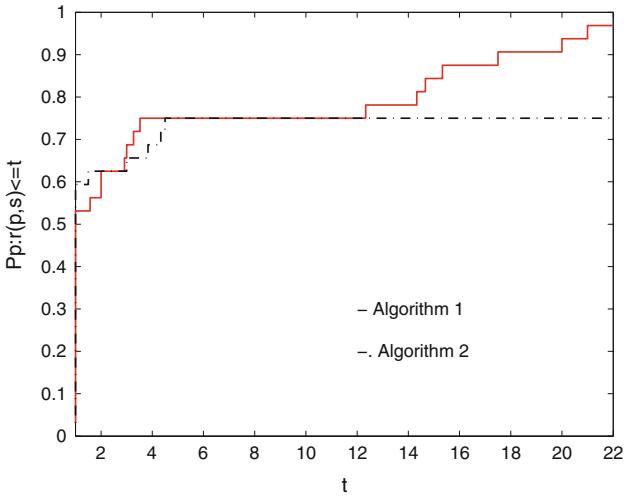

Fig. 1 Performance profiles of these methods (*NI*)

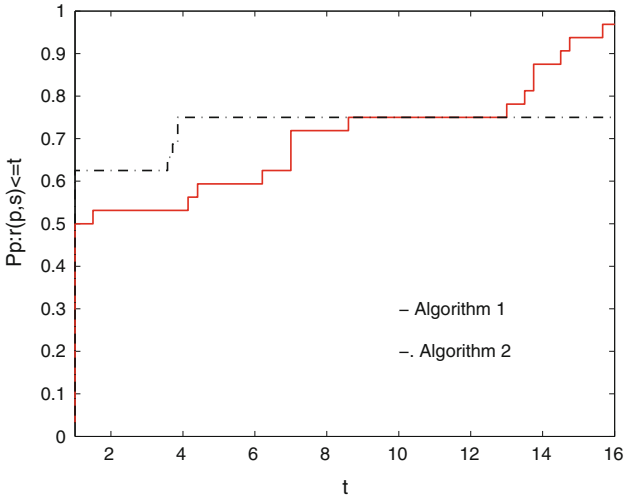

Fig. 2 Performance profiles of these methods (*NG*)

Figures 1 and 2 show that the proposed method outperforms the normal method about 10% test problems. Moreover, the presented method solves about 97% of the test problems and Algorithm 2 solves about 75% of the test problems successfully, respectively. Overall, the numerical results is interesting. We hope this method could be further investigated.

Acknowledgments We would like to thank these referees and the editors for giving us many valuable suggestions and comments which improve this paper greatly.

References

1. Bertsekas DP (1995) Nonlinear programming. Athena Scientific, Belmont
2. Bing Y, Lin G (1991) An efficient implementation of Merrill's method for sparse or partially separable systems of nonlinear equations. SIAM J Optim 2:206–221
3. Conn AR, Gould NIM, Toint PL (2000) Trust region method. Society for Industrial and Applied Mathematics, Philadelphia
4. Dennis JE, Moré JJ (1974) A characterization of superlinear convergence and its application to quasi-Newton methods. Math Comput 28:549–560
5. Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. Math Program 91:201–213
6. Fan JY (2003) A modified Levenberg-Marquardt algorithm for singular system of nonlinear equations. J Comput Math 21:625–636
7. Gomez-Ruggiero M, Martinez JM, Moretti A (1992) Comparing algorithms for solving sparse nonlinear systems of equations. SIAM J Sci Stat Comput 23:459–483
8. Griewank A (1986) The 'global' convergence of Broyden-like methods with a suitable line search. J Aust Math Soc Ser B 28:75–92
9. Levenberg K (1944) A method for the solution of certain nonlinear problem in least squares. Q Appl Math 2:164–166
10. Li D, Fukushima M (1999) A global and superlinear convergent Gauss-Newton-based BFGS method for symmetric nonlinear equations. SIAM J Numer Anal 37:152–172

11. Li D, Qi L, Zhou S (2002) Descent directions of quasi-Newton methods for symmetric nonlinear equations. SIAM J Numer Anal 40(5):1763–1774
12. Marquardt DW (1963) An algorithm for least-squares estimation of nonlinear inequalities. SIAM J Appl Math 11:431–441
13. Moré JJ (1983) Recent development in algorithm and software for trust region methods. In: Bachem A, Grotschel M, Kortz B (eds) Mathematical programming: the state of the art. Spinger, Berlin, pp 258–285
14. Moré JJ, Garbow BS, Hillström KE (1981) Testing unconstrained optimization software. ACM Trans Math Softw 7:17–41
15. Nocedal J, Wright SJ (1999) Numerical optimization. Spinger, New York
16. Ortega JM, Rheinboldt WC (1970) Iterative solution of nonlinear equations in several variables. Academic Press, New York
17. Raydan M (1997) The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. SIAM J Optim 7:26–33
18. Wang YJ, Xiu NH (2004) Theory and algoithm for nonlinear programming. Shanxi Science and Technology Press, Xian
19. Wei Z, Yuan G, Lian Z (2004) An approximate Gauss-Newton-based BFGS method for solving symmetric nonlinear equations. Guangxi Sci 11(2):91–99
20. Yamashita N, Fukushima M (2001) On the rate of convergence of the Levenberg-Marquardt Method. Computing 15:239–249
21. Yuan Y (1998) Trust region algorithm for nonlinear equations. Information 1:7–21
22. Yuan G, Chen C, Wei Z (2010) A nonmonotone adaptive trust-region algorithm for symmetric nonlinear equations. Nat Sci 2(4):373–378
23. Yuan G, Li X (2004) An approximate Gauss-Newton-based BFGS method with descent directions for solving symmetric nonlinear equations. OR Trans 8:10–26
24. Yuan G, Li X (2010) A rank-one fitting method for solving symmetric nonlinear equations. J Appl Funct Anal 5:389–407
25. Yuan G, Lu X (2008) A new backtracking inexact BFGS method for symmetric nonlinear equations. Comput Math Appl 55:116–129
26. Yuan G, Lu X, Wei Z (2009) BFGS trust-region method for symmetric nonlinear equations. J Comput Appl Math 230(1):44–58
27. Yuan G, Meng S, Wei Z (2009) A trust-region-based BFGS method with line search technique for symmetric nonlinear equations. Adv Oper Res 2009:1–20
28. Yuan G, Wang Z, Wei Z (2009) A rank-one fitting method with descent direction for solving symmetric nonlinear equations. Int J Commun Netw Syst Sci 6:555–561
29. Yuan G, Wei Z, Lu X (2009) A nonmonotone trust region method for solving symmetric nonlinear equations. Chin Q J Math 24(4):574–584
30. Yuan G, Sun W (1997) Optimization theory and methods. Science Press, Beijing
31. Zhang J, Wang Y (2003) A new trust region method for nonlinear equations. Math Methods Oper Res 58:283–298
32. Zhu D (2005) Nonmonotone backtracking inexact quasi-Newton algorithms for solving smooth nonlinear equations. Appl Math Comput 161:875–895