J Glob Optim (2007) 39:221–234
DOI 10.1007/s10898-007-9134-4

ORIGINAL PAPER

An improved inexact Newton method

Haibin Zhang · Naiyang Deng

Received: 3 May 2006 / Accepted: 2 January 2007 / Published online: 30 January 2007

Abstract For unconstrained optimization, an inexact Newton algorithm is proposed recently, in which the preconditioned conjugate gradient method is applied to solve the Newton equations. In this paper, we improve this algorithm by efficiently using automatic differentiation and establish a new inexact Newton algorithm. Based on the efficiency coefficient defined by Brent, a theoretical efficiency ratio of the new algorithm to the old algorithm is introduced. It has been shown that this ratio is greater than 1, which implies that the new algorithm is always more efficient than the old one. Furthermore, this improvement is significant at least for some cases. This theoretical conclusion is supported by numerical experiments.

Keywords Cholesky factorization · Unconstrained optimization · Newton equation · Automatic differentiation · Preconditioned conjugate gradient method

Mathematics subject classification 90C30 · 65K05

1 Introduction

Consider unconstrained optimization problem

$$\min \quad f(x), \qquad x \in R^n. \tag{1.1}$$

It is well known that, under proper assumptions, Newton's method is quadratically convergent. But its efficiency is reduced by its expensive computational cost, especially, for the middle-large scale problems. A number of modifications have been proposed to improve the original Newton's method. In recent years, iterative type

H. Zhang (✉)
College of Applied Science, Beijing University of Technology, Beijing 100022, China
e-mail: zhanghaibin@bjut.edu.cn

N. Deng
College of Science, China Agricultural University, Beijing 100083, China

Newton-like methods have received much attention. Steihaug and Toint proposed Newton-CG like algorithms (see [7] and [8]), in which the Newton equation is solved by CG method approximately. An inexact Newton algorithm (Algorithm 1) is established in Ref. [4]. It is shown that, for the problems where the computation cost to evaluate the gradient and the Hessian matrices is small, its average computation cost of every iteration is much less than that of Newton's method. However, the improvement is just a little when the computation cost is large.

This paper improves Newton's method further by modifying Algorithm 1 above. The key point is to evaluate the gradient and the Hessian more efficiently. For this evaluation, it is well known that, there are four kinds of approaches. The first one is to make the manual development of code for evaluating analytic derivatives of a function. It is tedious and error-prone activity, although it is likely to result in the most efficient code.

The second approach is to estimate the derivatives using divided differences, such estimates are prone to truncation error when the differencing intervals are numerically large, and to round off error when they are small. In addition, if the run-time cost to evaluate a gradient $\nabla f(x)$ by divided differences is denoted as Q_g^{DD}, then

$$Q_g^{DD} = (n+1)Q_f,$$

where Q_f denote the run-time cost to evaluate $f(x)$, n is the dimension of $x = (x_1, x_2, \ldots, x_n)^T \in R^n$. Therefore, the run-time requirements of a divided difference approach are often unacceptably high, particularly for problems when the dimension n is a large number (thousands), see [1].

The third approach is symbolic differentiation. It can be used to differentiate individual functions by symbol manipulation programs. But its implementation is almost impossible for the middle and large scale problems and for problems involving conditional statements, loops and subroutine calls. Moreover, the run-time cost by symbolic differentiation is also very large; in general, if the cost to evaluate a gradient $\nabla f(x)$ is expressed by Q_g^{SD}, then

$$Q_g^{SD} \approx nQ_f,$$

where Q_f is the cost to evaluate $f(x)$.

At last, the fourth approach is Automatic Differentiation (AD) techniques. It calculates derivatives by transforming the function source codes into the codes that calculate numerical values for derivatives of the underlying function with about the same accuracy and efficiency as the function values themselves (see [1,3,5]).

In this paper, applying automatic differentiation, we improve the inexact Newton algorithm (Algorithm 1) in Ref. [4] and establish a new Newton-PCG algorithm (Algorithm 2), where the Hessian-vector can be evaluated at the several times (independent of n) cost of the underling functions without evaluating the Hessian matrices. The results by theoretic analysis and numerical experiments in this paper show that Algorithm 2 is more efficient than Algorithm 1.

The rest of the paper is organized as follows: In Sect. 2, Algorithm 2 is established while Algorithm 1 is recalled. Efficiency analysis is discussed in Sect. 3 and numerical examples are described in Sect. 4. At last, some conclusions are given in Sect. 5.

2 The improved inexact Newton algorithm

Suppose that problem (1.1) has a solution x^* and satisfies the following assumptions.

Assumption (A1) $\nabla^2 f(x)$ is Lipschitz continuous with the constant L in a neighborhood of the solution x^* to (1.1);

Assumption (A2) $\nabla^2 f(x^*)$ is positive definite.

Let us first describe AD algorithms.

2.1 Automatic differentiation (AD)

Automatic Differentiation can be used to evaluate the gradient, the directional gradient and the second-order directional gradient of the function see e.g., [5]. The following algorithm, called Algorithm ADR, is used to evaluate the gradient $\nabla f(x)$ by the reverse mode of AD.

Algorithm ADR

Step 0 Set $x \in R^n$.

Step 1 Evaluate $f(x)$ using General Evaluation Procedure.

Step 2 Evaluate $\nabla f(x)$ using Reverse Propagation of Gradients. □

In addition, Table 3.2 in Ref. [5], Ch.3, Sect. 3.1, gives an algorithm which efficiently evaluates a gradient ∇f and m Hessian-vector $\nabla^2 f \cdot \dot{x}_i$ with $i = 1, \ldots, m$, where $\dot{x}_i \in R^n$ and m is any positive integer. This algorithm exploits the forward mode of AD after the gradient is calculated. It can be described briefly by the following Algorithm ADF(m) with a parameter m.

Algorithm ADF(m)

Step 0 Set $x, \dot{x}_1, \ldots, \dot{x}_m \in R^n$.

Step 1 Calculate $\nabla f(x)$ by Algorithm ADR.

Step 2 Evaluate $\nabla^2 f(x) \cdot \dot{x}_i, i = 1, \ldots, m$, using Forward Propagation of Tangents. □

Let Q^{ADR} be the computation cost to evaluate a gradient ∇f by Algorithm ADR. Then, by (3.24) in Ref. [5], Ch.3, Sect. 3.4, we have

$$Q^{\mathrm{ADR}} \leq 4Q_f, \tag{2.1}$$

where Q_f is the computation cost to evaluate a function value $f(x)$. Furthermore, let $Q^{\mathrm{ADF}}(m)$ be the computation cost involved in Algorithm ADF, by Griewank [5], we have

$$Q^{\mathrm{ADF}}(m) \leq (4 + 6m)Q_f. \tag{2.2}$$

It should be noted that

(1) By Algorithm ADF(1), the Hessian-vector can be evaluated at about ten times computational cost of the function without evaluating the Hessian matrix.
(2) The evaluation of the Hessian matrix $\nabla^2 f(x)$ can be completed by using Algorithm ADF(n) with $\dot{x}_i = e_i, i = 1, \ldots, n$, where e_i is the ith Cartensian basic vector in R^n. The computational cost is $O(n)$ times cost of the function, not $O(n^2)$.

Above two points can be used efficiently to improve Algorithm 1 in Ref. [4].

2.2 Algorithm 1 (The CF-PCG Algorithm in Ref. [4])

Algorithm 1 includes a sub-algorithm — Algorithm PCG1(C, A, b, l, e) (see Algorithm PCG(C, A, b, l, e) in Ref. [4]), which was used to solve the linear system

$$As = b, \tag{2.3}$$

where C is preconditioner, l is maximum number of subiterations, and e is a scalar used in the termination criterion (2.4). Now we cite the sub-algorithm and algorithm as follows.

Algorithm PCG1(C, A, b, l, e)

Step 0 Initial data: set the initial point $s_0 = 0$, $r_0 = -b$. Set $i = 1$.

Step 1 Termination test: if

$$\|r_{i-1}\| \le \|b\|^{1+e} \qquad \text{or} \qquad i - 1 = l, \tag{2.4}$$

go to Step 4.

Step 2 Subiteration:

(a) set $z = Cr_{i-1}, t_{i-1} = z^T r_{i-1}$;
(b) if $i = 1$, then $q = z$; else $\beta = t_{i-1}/t_{i-2}$ and $q = z + \beta q$;
(c) set $s_i = s_{i-1} + \lambda q$, where $\lambda = t_{i-1}/q^T w$ and $w = Aq$;
(d) set $r_i = r_{i-1} + \lambda w$.

Step 3 Set $i = i + 1$ and go to Step 1.

Step 4 Set $\tilde{s} = s_{i-1}$.

(1) Find the optimal solution σ^* to the one-dimensional optimization problem

$$\max \qquad v_1(\sigma) = \frac{\ln(2+\sigma)}{(1+\overline{p}(\sigma))Q_{Hg} + Q_D + \sigma Q_I}, \tag{2.5}$$

$$\text{s.t.} \qquad \sigma \text{ is a nonnegative integer}, \tag{2.6}$$

where

$$\overline{p}(\sigma) = \lceil \frac{\ln(2+\sigma)}{\ln 2} - 1 \rceil,$$

is the smallest integer not smaller than $\frac{\ln(2+\sigma)}{\ln 2} - 1$. The computation costs Q_{Hg}, Q_D, and Q_I are defined as follows.

$Q_{Hg} = Q_H + Q_g$;
$Q_H =$ the computation cost to evaluate a Hessian $\nabla^2 f$;
$Q_g =$ the computation cost to evaluate a gradient ∇f;

$$Q_D = \frac{1}{6}n^3 + \frac{3}{2}n^2 - \frac{2}{3}n, \tag{2.7}$$

is the computation cost in a CF Step;

$$Q_I = 2n^2 + 6n + 2, \tag{2.8}$$

is an upper bound of the computation cost in one subiteration of a PCG Step.

(2) $p = p^* = \overline{p}(\sigma^*)$.

(3) if $\sigma^* = 1$, $l_1 = l_1^* = 1$; if $\sigma^* \ge 2$,

$$l_m = l_m^* = \begin{cases} 2^m, & m = 1, \ldots, p^* - 1; \\ \sigma^* - 2^{p^*} + 2, & m = p^*. \end{cases}$$

Algorithm 1

Step 0 Initial data: set the initial point $x^0 \in R^n$. Set $k = 0$.

Step 1 Termination test: if $\nabla f(x^k) = 0$, stop.

Step 2 Switch test: if $k = (p+1)k_0$ for some positive integer k_0, go to Step 3; otherwise go to Step 4.

Step 3 CF Step: set

$$B^k = \nabla^2 f(x^k). \tag{2.9}$$

Find the solution s^k to the Newton equation

$$\nabla^2 f(x^k)s = -\nabla f(x^k) \tag{2.10}$$

by Cholesky factorization $\nabla^2 f(x^k) = L_k D_k L_k^T$. Set $m = 0$, and go to Step 5.

Step 4 PCG Step: set $B^k = B^{k-1}, m = m+1$. Find $\tilde{s}$ by Algorithm PCG1$((B^k)^{-1}, \nabla^2 f(x^k), -\nabla f(x^k), l_m, l_m/2^m)$. Set $s^k = \tilde{s}$.

Step 5 Update the iteration: set $x^{k+1} = x^k + s^k$. Set $k = k+1$ and go to Step 1.

2.3 Algorithm 2

Our improved algorithm also includes a sub-algorithm — Algorithm PCG$(M, \nabla f(x), l, e)$, which is used to solve the Newton equation

$$\nabla^2 f(x)s = -\nabla f(x), \tag{2.11}$$

where M is the preconditioner, l is the maximum number of subiterations and e is a scalar.

Algorithm PCG$(M, \nabla f(x), l, e)$

Step 0 Initial Data. Set $s_0 = 0, r_0 = -\nabla f(x), i = 1$.

Step 1 Termination Test. If

$$\|r_{i-1}\| \le \|\nabla f(x)\|^{1+e} \ \ or \ \ i-1 = l, \tag{2.12}$$

then terminate the iteration by taking $\bar{s} = s_{i-1}$.

Step 2 Subiteration:

(a) solve the equation $Mz = r_{i-1}$ for z, set $t_{i-1} = z^T r_{i-1}$;
(b) if $i = 1$, then $\beta = 0$ and $q = z$; else, $\beta = t_{i-1}/t_{i-2}$, $q = z + \beta q$ and evaluate

$$w = \nabla^2 f(x) \cdot q \tag{2.13}$$

by Algorithm ADF(1) with $\dot{x}_1 = q$.
(c) set $s_i = s_{i-1} + \alpha q$, where $\alpha = t_{i-1}/q^T w$ and $w = \nabla^2 f(x)q$;
(d) set $r_i = r_{i-1} + \alpha w$.

Step 3 Set $i = i+1$, and go to Step 1.

Now we are in a position to describe our improved algorithm — Algorithm 2. Based on Algorithm ADR, Algorithm ADF, and Algorithm PCG$(\cdot)$, The main steps of Algorithm 2 are as follows: an exact Newton step with Cholesky factorization (CF step) and p inexact Newton steps with preconditioned conjugate gradient subiterations (PCG steps), in which the parameter σ^* is the solution to the following optimization problem

$$\max \ v_2(n, Q_f, \sigma) = \frac{\ln(2+\sigma)}{(6n+4+4p+6\sigma)Q_f + Q_D + \sigma Q_I^-}, \tag{2.14}$$

$$\text{s.t. } \sigma \text{ is a nonnegative integer,} \tag{2.15}$$

$$p = \bar{p}(\sigma) = \lceil \frac{\ln(2+\sigma)}{\ln 2} - 1 \rceil, \tag{2.16}$$

where $\lceil \cdot \rceil$ is the smallest integer not smaller than $\cdot$, Q_f is the computation cost to compute a function value f, Q_D is the computation cost to solve the Newton equation by CF and Q_I^- is the computation cost to execute one PCG subiteration, where

$$Q_D = \frac{1}{6}n^3 + \frac{3}{2}n^2 - \frac{2}{3}n, \quad Q_I^- = n^2 + 6n + 2, \tag{2.17}$$

Algorithm 2 (*Improved Inexact Newton Algorithm*)

Step 0 Initial Data. Set the initial point $x_0 \in R^n$. If $\sigma^* = 1$, set $p = 1$ and $l_1 = 1$. If $\sigma^* \geq 2$, set

$$l_m = \begin{cases} 2^m, & m = 1, \ldots, p-1; \\ \sigma^* - 2^p + 2, & m = p, \end{cases}$$

where σ^* is the solution to the optimization problem (2.14)–(2.16) and $p = \bar{p}(\sigma^*)$ is defined by (2.16), and set $k = 0$.

Step 1 Evaluate $\nabla f(x_k)$ by Algorithm ADR. If $\nabla f(x_k) = 0$, then terminate the iteration by taking $x^* = x_k$.

Step 2 Switch Test. If k can be divided by $p+1$ with no remainder, go to Step 3; otherwise, go to Step 4.

Step 3 CF Step. Evaluate $\nabla^2 f(x_k)$ by using Algorithm ADF(n) with setting $\dot{x}_i = e_i$, $i = 1, \ldots, n$, where e_i is the ith Cartensian basic vector in R^n. Set

$$B_k = \nabla^2 f(x_k).$$

Find the solution s_k to Newton equation (2.11) by CF $\nabla^2 f(x_k) = L_k D_k L_k^T$. Set $m = 0$ and go to Step 5.

Step 4 PCG Step. Set $B_k = B_{k-1}$, $m = m + 1$, and

$$M = B_k$$

Find the approximate solution s_k to the Newton equation (2.11) by the Algorithm PCG $(M, \nabla f(x_k), l_m, 1 + \frac{l_m}{2^m})$.

Step 5 Update Solution Estimation. Set $x_{k+1} = x_k + s_k$. Set $k = k + 1$, and go to Step 1.

3 The efficiency analysis of algorithm 2

In this section, we analyze the efficiency of Algorithm 2. Let's first consider its computational cost based on its structure: one CF step is followed by p PCG steps.

Solving the Newton equation by CF. Denote the corresponding arithmetic computation cost as Q_D.

Thus, by (2.2), the total computation cost of one CF step with an extra gradient evaluation is

$$Q^{\text{ADF}}(n) + Q_D \leq (4 + 6n)Q_f + Q_D. \tag{3.1}$$

For the PCG steps, denote $Q_I^- = Q_I^-(n)$ as the arithmetic computation cost in one PCG subiteration. The total computation cost of p PCG steps after a CF step with p extra gradient evaluations has the upper bound

$$\sum_{t=1}^{p}[Q^{\mathrm{ADF}}(l_t) + (l_t)Q_I^-] \le 4p + 6\left(\sum_{t=1}^{p} l_t\right)Q_f + \left(\sum_{t=1}^{p} l_t\right)Q_I^-, \tag{3.2}$$

where l_t is the tth PCG subiteration number.

Combining (3.1) and (3.2), the total computation cost in the $p+1$ steps which are a CF step and p PCG steps has the upper bound

$$(6n + 4 + 4p + 6\sigma)Q_f + Q_D + \sigma Q_I^-, \tag{3.3}$$

where

$$\sigma = \sum_{t=1}^{p} l_t.$$

Now, we analyze and compare the efficiency of Algorithm 2 and Algorithm 1. Our analysis is based on the efficiency coefficient given by Brent [2].

Definition 3.1 Efficiency coefficient: suppose that the sequence $\{x^0, x^1, \cdots, x^k, \cdots\}$ is generated by an algorithm. If $\{x^k\}$ converges to the solution x^* to (1.1), then the efficiency coefficient Γ of the algorithm is defined by

$$\Gamma = \liminf_{k\to\infty} \frac{\ln(-\ln\|x^k - x^*\|)}{\sum_{i=1}^{k} Q[x^{i-1}, x^i]}, \tag{3.4}$$

where $Q[x^{i-1}, x^i]$ is the computation cost required to compute x^i from x^{i-1}.

Theorem 3.1 *If the initial point x^0 is close enough to the solution x^*, Algorithm 2 is well-defined.*

Proof To prove Algorithm 2 to be well-defined, we only need to show the existence of the global solution σ^* to (2.14)–(2.16). In fact, it is easy to see that

$$\lim_{\sigma\to\infty} v_2(\sigma) = 0$$

and

$$v_2(0) = \frac{\ln 2}{(6n+4)Q_f + Q_D} > 0,$$

we conclude that the series $\{v_2(\sigma), \sigma = 0, 1, 2, \ldots\}$ has a finite maximum point σ^*. □

The efficiency coefficient of new algorithm is estimated by the following theorem.

Theorem 3.2 *The efficiency coefficient Γ_2 of Algorithm 2 satisfies*

$$\Gamma_2 \ge v_2^* \stackrel{\mathrm{def}}{=} v_2(\sigma^*), \tag{3.5}$$

where $v_2(\cdot)$ is defined by (2.14), and σ^* is the global solution to (2.14)–(2.16).

The proof of the theorem is the same as that of Theorem 4.2 in Ref. [4]. □

Theorem 3.3 *The efficiency coefficient* Γ_1 *of Algorithm 1 satisfies*

$$\Gamma_1 \geq v_1^* \stackrel{def}{=} v_1(\sigma^*), \tag{3.6}$$

where $v_1(\cdot)$ *is defined by (2.5), and* σ^* *is the global solution to (2.5)–(2.6).*

This is Theorem 4.2 in Ref. [4]. □

For comparison, the efficiency of Newton's method is given in the following theorem, which can be considered as a special case of either Algorithm 1 or Algorithm 2 with $\sigma = 0$.

Theorem 3.4 *The efficiency coefficient* Γ_N *of Newton's method satisfies*

$$\Gamma_N \geq v_N \stackrel{\text{def}}{=} \frac{\ln 2}{Q_{Hg} + Q_D} = \frac{\ln 2}{(4+6n)Q_f + Q_D}. \tag{3.7}$$

Proof See Theorem 4.4 in Ref. [4]. □

Now let us compare the lower bound of the efficiency coefficient of Algorithm 2, v_2^*, with that of Algorithm 1, v_1^*. We need the following lemma.

Lemma 3.5 *Suppose that* σ_1^* *and* σ_2^* *are, respectively, the global solution to (2.5)–(2.6) and (2.14)–(2.16). When* $n \geq 20$*, we have*

$$\sigma_1^* \leq \sigma_2^* \tag{3.8}$$

and

$$\sigma_1 - 1 \leq \sigma_1^* \leq \sigma_1 + 1, \quad \sigma_2 - 1 \leq \sigma_2^* \leq \sigma_2 + 1, \tag{3.9}$$

where σ_1 *and* σ_2 *are the solution of the unconstrained problems, respectively, with the objective function* $v_1(\sigma)$ *in (2.5) and* $v_2(\sigma)$ *in (2.14) with continuous variable* σ*, and* σ_1 *and* σ_2 *satisfy*

$$(2+\sigma_1)\ln(2+\sigma_1) - \sigma_1 = \frac{Q_D}{Q_I} \tag{3.10}$$

and

$$(2+\sigma_2)\ln(2+\sigma_2) - \sigma_2 = \frac{Q_D + 6nQ_f}{Q_I^- + 6Q_f}. \tag{3.11}$$

Proof Consider maximizing the objective function $v_1(\sigma)$ in (2.5) and $v_2(\sigma)$ in (2.14) with continuous variable σ. Let $v_1'(\sigma) = 0$ and $v_2'(\sigma) = 0$, we can get (3.10) and (3.11). Obviously, the conclusion (3.9) is right. Because the function $(2+\sigma)\ln(2+\sigma) - \sigma$ is strictly increasing with $\sigma \geq 0$, and

$$\frac{Q_D}{Q_I} < \frac{Q_D + 6nQ_f}{Q_I^- + 6Q_f},$$

by (3.10) and (3.11), we have $\sigma_1 \leq \sigma_2$, thus the conclusion (3.8) is proved. □

Note that v_1^*, v_2^* and v_N (are, respectively, defined by (3.6), (3.5), and (3.7)) are the functions of n and Q_f. For comparison, we introduce the following notations:

$$R_1(n, Q_f) \stackrel{\text{def}}{=} \frac{v_1^*}{v_N}, \quad R_2(n, Q_f) \stackrel{\text{def}}{=} \frac{v_2^*}{v_N}. \tag{3.12}$$

Theorem 3.6 *For fixed $n \geq 20$, we have the following conclusions:*

(1) $R_1(n, Q_f)$ is decreasing with Q_f, and $R_2(n, Q_f)$ is increasing with Q_f.
(2) $R_1(n, Q_f) \leq R_1(n, 0) < R_2(n, 0) \leq R_2(n, Q_f)$
(3) When $n \to \infty$, $R_1(n, 0) \sim \ln n / \ln 2$.
(4) When n is fixed,

$$\lim_{Q_f \to \infty} R_1(n, Q_f) = 1. \tag{3.13}$$

Proof Consider $R_1(n, Q_f), R_2(n, Q_f)$ as the functions with continuous variable Q_f. From the Eq. 3.10, we obtain $\partial\sigma_1/\partial Q_f = 0$. By

$$R_1(n, Q_f) = \frac{\ln(2+\sigma_1)[(6n+4)Q_f + Q_D]}{[(p(\sigma_1)+1)(6n+4)Q_f + Q_D + \sigma_1 Q_I]\ln 2}$$

and

$$p(\sigma_1) + 1 = \ln(2+\sigma_1)/\ln 2,$$

we can get

$$\frac{\partial R_1}{\partial Q_f} = \frac{\ln(2+\sigma_1)}{\ln 2} \cdot \frac{(6n+4)(\sigma_1 Q_I - pQ_D)}{[(p(\sigma_1)+1)(6n+4)Q_f + Q_D + \sigma_1 Q_I]^2}.$$

When $n \geq 20$, $\sigma_1 \geq 1, p \geq 1$. Let $y_1(\sigma) = \ln(2+\sigma) - \ln 2 - \sigma/(2+\sigma)$. By $y_1'(\sigma) > 0$ and $y_1(0) = 0$, we can get when $\sigma > 0$, $y_1(\sigma) = \ln(2+\sigma) - \ln 2 - \sigma/(2+\sigma) > 0$

$$p = \frac{\ln(2+\sigma_1)}{\ln 2} - 1 > \frac{\sigma_1}{(2+\sigma_1)\ln 2},$$

$$\frac{\sigma_1}{p} < (2+\sigma_1)\ln 2,$$

$$\frac{(p+1)\sigma_1}{p} < (2+\sigma_1)(p+1)\ln 2 = (2+\sigma_1)\ln(2+\sigma_1).$$

So,

$$\frac{\sigma_1}{p} < (2+\sigma_1)\ln(2+\sigma_1) - \sigma_1 = \frac{Q_D}{Q_I}.$$

That is, $\sigma_1 Q_I - pQ_D < 0$, therefore,

$$\frac{\partial R_1}{\partial Q_f} < 0. \tag{3.14}$$

Denote $A = Q_I^- + 6Q_f$ and $B = Q_D + 6nQ_f$. By (3.11),

$$(2+\sigma_2)\ln(2+\sigma_2) - \sigma_2 = B/A$$

and

$$\bar{p}(\sigma_2) + 1 = \frac{\ln(2+\sigma_2)}{\ln 2},$$

we have

$$R_2(n, Q_f) = \frac{4Q_f + B}{4Q_f + A(2+\sigma_2)\ln 2}.$$

And by

$$\frac{\partial A}{\partial Q_f} = 6, \quad \frac{\partial B}{\partial Q_f} = 6n$$

and

$$\frac{\partial \sigma_2}{\partial Q_f} = \frac{6(nA - B)}{A^2 \ln(2+\sigma_2)},$$

we obtain

$$\begin{aligned}
\frac{\partial R_2}{\partial Q_f} &= \frac{(6n+4)[4Q_f + A(2+\sigma_2)\ln 2] - (4Q_f + B)[4 + 6(2+\sigma_2)\ln 2 + A\partial\sigma_2/\partial Q_f \ln 2]}{[4Q_f + A(2+\sigma_2)\ln 2]^2} \\
&= \frac{(6n+4)[4Q_f \ln(2+\sigma_2) + (A\sigma_2 + B)\ln 2] - (4Q_f + B)[4\ln(2+\sigma_2) + 6(n+\sigma_2)\ln 2]}{[4Q_f + A(2+\sigma_2)\ln 2]^2 \ln(2+\sigma_3)} \\
&= \frac{(6n+4)\sigma_2 Q_I^- \ln 2 + 4Q_D \ln 2 - [4\ln(2+\sigma_2) + 6\sigma_2 \ln 2]Q_D}{[4Q_f + A(2+\sigma_2)\ln 2]^2 \ln(2+\sigma_2)} \\
&= \frac{[(5\ln 2)n^3 + (31\ln 2)n^2 + (40\ln 2)n + 8\ln 2]\sigma_2 - \left[\frac{2}{3}n^3 + 6n^2 - \frac{8}{3}n\right]\ln(2+\sigma_2)}{[4Q_f + A(2+\sigma_2)\ln 2]^2 \ln(2+\sigma_3)} \\
&> 0.
\end{aligned}$$

Therefore,

$$\frac{\partial R_2}{\partial Q_f} > 0. \tag{3.15}$$

By (3.14) and (3.15), the conclusion (1) is obtained.

Now, let's prove the conclusion (2). In fact, by (1) and $Q_f \geq 0$, the following inequalities are right:

$$R_1(n, Q_f) \leq R_1(n, 0),$$

$$R_2(n, 0) \leq R_2(n, Q_f).$$

In addition, by (3.8),

$$R_1(n,0) = \frac{Q_D \ln(2+\sigma_1^*)}{[Q_D + \sigma_1^* Q_I]\ln 2} < \frac{Q_D \ln(2+\sigma_2^*)}{[Q_D + \sigma_2^* Q_I^-]\ln 2} = R_2(n,0),$$

the conclusion (2) is obtained.

The conclusion (3) is actually the conclusion of Theorem 4.5 in Ref. [4].

Finally, let's proof the conclusion (4), when n is fixed,

$$\lim_{Q_f \to \infty} Q_D/((6n+4)Q_f) = 0$$

and by (3.10) and (3.9),

$$\lim_{Q_f \to \infty} (Q_D + \sigma_1^* Q_I)/((6n+4)Q_f) = 0.$$

And by

$$R_1(n, Q_f) = \frac{v_1^*}{v_N} = \frac{\ln(2+\sigma_1^*)[(6n+4)Q_f + Q_D]}{[(p(\sigma_1^*)+1)(6n+4)Q_f + Q_D + \sigma_1^* Q_I]\ln 2}$$

and

$$p(\sigma_1^*) + 1 = \ln(2+\sigma_1^*)/\ln 2,$$

we have

$$\begin{aligned}
&\lim_{Q_f\to\infty} R_1(n, Q_f) \\
&= \lim_{Q_f\to\infty} \frac{\ln(2+\sigma_1^*)[1 + Q_D/((6n+4)Q_f)]}{[(p(\sigma_1^*)+1) + (Q_D + \sigma_1^* Q_I)/((6n+4)Q_f)]\ln 2} \\
&= \frac{\ln(2+\sigma_1^*)[1+0]}{[(p(\sigma_1^*)+1)+0]\ln 2} \\
&= 1.
\end{aligned}$$

To compare Algorithm 2 and Algorithm 1, we need the following definition.

Definition 3.2 The efficiency ratio r of Algorithm 2 to Algorithm 1 is defined as

$$r = r(n, Q_f) = \frac{v_2^*}{v_1^*}, \tag{3.16}$$

where v_1^*, v_2^* are, respectively, defined in (3.6) and (3.5), and $r = r(n, Q_f)$ is the function of n and Q_f. □

In fact, by (3.12), we have

$$r = \frac{v_2^*}{v_1^*} = \frac{v_2^*/v_N}{v_1^*/v_N} = \frac{R_2(n, Q_f)}{R_1(n, Q_f)}.$$

Therefore, the efficiency ratio r of Algorithm 2 over Algorithm 1 is just the improvement ratio of Algorithm 2 and Algorithm 1 over Newton's method.

Theorem 3.7 *Comparing the efficiency of Algorithm* 2 *and that of Algorithm* 1*, we have*

(1) *when* $n \geq 20$, $r = r(n, Q_f) > 1$.
(2) $r = r(n, Q_f)$ *is increasing with respect to* Q_f.

□

Proof By (3.12) and the conclusion (2) in Theorem 3.6, the conclusion (1) is obtained.
By (3.12) and the conclusion (1) in Theorem 3.6, we get (2).

Remark 3.8 Theorem 3.7 shows that Algorithm 2 is more efficient than Algorithm 1, and their efficiency ratio is increasing with respect to the complexity of the target functions. The theoretical value of the efficiency ratio $r(n, Q_f)$ of Algorithm 2 to Algorithm 1 is given in Table 1 when $n = 100, 200, \ldots, 1,000$, and $Q_f = n, 3n, 10n, n^2$. The table shows that Algorithm 2 is theoretically more efficient than Algorithm 1 in evidence. □

Table 1 The theoretical value of the efficiency ratio $r(n, Q_f)$ of Algorithm 2 to Algorithm 1

n	$Q_f = n$	$Q_f = 3n$	$Q_f = 10n$	$Q_f = n^2$
100	1.6374	2.0957	2.7762	3.6129
200	1.5630	1.9915	2.7977	4.3236
300	1.4795	1.8678	2.6924	4.7511
400	1.4334	1.7832	2.5895	5.0672
500	1.4095	1.7261	2.5006	5.3155
600	1.3909	1.6851	2.4242	5.5183
700	1.3627	1.6318	2.3405	5.6891
800	1.3405	1.5905	2.2690	5.8404
900	1.3237	1.5578	2.2074	5.9745
1,000	1.3113	1.5315	2.1537	6.0950

4 Numerical experiments

In this section, we test our algorithm by all of the numerical examples in Ref. [4]: the Extended Rosenbrock function, the Extended Powell singular function, Penalty function 1, Penalty function 2 and Variably dimensioned function. These test problems are quoted from the unconstrained optimization problems in Ref. [6].

Algorithm is executed by C++ routines with double precision. The initial points of these problems are standard start points. Notice that these algorithms are local algorithm and our theoretical results are valid in a neighborhood of x^*, where

$$\|\nabla f(x)\| < 1.$$

However, the standard starting points may be rather far away from x^*. So, to prevent the earlier mature, in our code the termination criterion (2.12) in PCG step is modified as

$$\|r_{i-1}\| \leq \min\{\| \bigtriangledown f(x)\|^{1+e}, 0.9\}.$$

In addition, the condition

$$\|\nabla f(x)\| \leq 10^{-6}$$

is used for the termination test.

We test the five problems with different dimensions $n = 100, 200, \ldots, 1,000$. The numerical results for the top three problems are listed in Tables 2–4 and the results for the fifth one in Table 5. Note that, there is no the results corresponding to the fourth problem because Algorithm 1 and/or Algorithm 2 are not convergent for many values of n, and the lack of some rows in Table 5 is because of the same reason.

The main value we are interested in is the ratio

$$r_{\text{prac}} = \frac{\text{The CPU time by Algorithm 1}}{\text{The CPU time by Algorithm 2}} \tag{4.1}$$

which shows the practical improvement of Algorithm 2 over Algorithm 1 and is listed in column 4. $r_{\text{theo}} = r$ defined in (3.16) is the theoretical efficiency ratio of Algorithm 2 over Algorithm 1 and listed in column 5. The parameter σ_1^*, σ_2^* of Algorithm 1, 2 defined in lemma 3.5 are, respectively, listed in column 2, 3. The total step numbers of Algorithm 1 and Algorithm 2 are, respectively, denoted as I_1 and I_2 and listed in column 6 and column 7. The gradient norms of the optimum by Algorithm 1 and

Table 2 Extended Rosenbrock function

n	σ_1^*	σ_2^*	r_{prac}	r_{theo}	I_1	I_2	$\|g_{opt}\|_1$	$\|g_{opt}\|_2$
100	6	6	**1.5000**	**1.7773**	6	6	5.85888e−008	5.85895e−008
200	6	14	**1.8333**	**1.6856**	6	6	9.02600e−008	9.02535e−008
300	14	14	**1.3500**	**1.5871**	6	6	1.10552e−007	1.10553e−007
400	14	30	**1.2340**	**1.5285**	6	6	1.17177e−007	1.17176e−007
500	14	30	**1.1932**	**1.4944**	6	6	1.31009e−007	1.31011e−007
600	28	30	**1.1931**	**1.4714**	6	6	1.43516e−007	1.43513e−007
700	30	30	**1.1667**	**1.4343**	6	6	1.55009e−007	1.55017e−007
800	30	57	**2.0430**	**1.4060**	6	6	1.65717e−007	1.65720e−007
900	30	62	**2.0190**	**1.3848**	6	6	1.75764e−007	1.75760e−007
1,000	30	62	**2.0085**	**1.3686**	6	6	1.85280e−007	1.85263e−007

Table 3 Extended Powell singular function

n	σ_1^*	σ_2^*	r_{prac}	r_{theo}	I_1	I_2	$\|g_{opt}\|_1$	$\|g_{opt}\|_2$
100	6	6	**1.2000**	**2.0032**	18	18	7.01119e−007	7.01119e−007
200	6	14	**1.3500**	**1.8984**	18	18	9.91532e−007	9.91532e−007
300	14	14	**1.1897**	**1.7804**	19	19	3.59814e−007	3.59814e−007
400	14	30	**1.3810**	**1.7028**	19	19	4.15478e−007	4.15478e−007
500	14	30	**1.3557**	**1.6524**	19	19	4.64519e−007	4.64519e−007
600	14	30	**1.3596**	**1.6171**	19	19	5.08855e−007	5.08855e−007
700	30	30	**1.1534**	**1.5683**	19	19	5.49626e−007	5.49626e−007
800	30	62	**1.1405**	**1.5309**	19	19	5.87575e−007	5.87575e−007
900	30	62	**1.1228**	**1.5017**	19	19	6.23217e−007	6.23217e−007
1,000	30	62	**1.1146**	**1.4786**	19	19	6.56928e−007	6.56928e−007

Table 4 Penalty function 1

n	σ_1^*	σ_2^*	r_{prac}	r_{theo}	I_1	I_2	$\|g_{opt}\|_1$	$\|g_{opt}\|_2$
100	6	6	**1.6250**	**1.8979**	32	32	1.67817e−009	1.67817e−009
200	14	14	**1.4737**	**1.7968**	33	33	2.30255e−008	2.30255e−008
300	14	14	**1.3302**	**1.6872**	34	34	7.25541e−009	7.25541e−009
400	14	30	**1.5077**	**1.6181**	35	35	1.52741e−007	1.52741e−007
500	14	30	**1.3046**	**1.5752**	35	36	9.97291e−008	9.97291e−008
600	14	30	**1.3117**	**1.5463**	36	36	7.83579e−007	7.83579e−007
700	30	30	**1.2014**	**1.5025**	37	37	1.76759e−007	1.76759e−007
800	30	62	**1.3094**	**1.4695**	37	38	7.61998e−010	7.61999e−010
900	30	62	**1.3083**	**1.4441**	38	38	1.72922e−009	1.72922e−009
1,000	30	62	**1.3078**	**1.4243**	39	39	5.04016e−011	5.04015e−011

Table 5 Variably dimensioned function

n	σ_1^*	σ_2^*	r_{prac}	r_{theo}	I_1	I_2	$\|g_{opt}\|_1$	$\|g_{opt}\|_2$
100	6	6	**1.5000**	**1.8979**	25	25	0.00000e+000	0.00000e+000
200	6	14	**1.6061**	**1.7968**	28	28	0.00000e+000	4.08724e−010
300	14	14	**1.4804**	**1.6872**	30	30	0.00000e+000	0.00000e+000
400	14	30	**1.6019**	**1.6181**	32	32	1.57208e−010	0.00000e+000
500	14	30	**1.7760**	**1.5752**	35	33	3.95346e−007	0.00000e+000

Algorithm 2 are, respectively, denoted as $\|g_{opt}\|_1$ and $\|g_{opt}\|_2$ and listed in column 8 and column 9.

From above tables, we see that all of the values of r_{prac} are greater than 1.1. This shows that Algorithm 2 is more efficient than Algorithm 1 significantly. In many cases, $r_{\text{prac}} \approx r_{\text{theo}}$. That is, the experimental results basically support our theoretical results.

5 Conclusions

In this paper, using efficiently automatic differentiation, a new inexact Newton algorithm (Algorithm 2) is established by improving Algorithm 1 in Ref. [4], in which the preconditioned conjugate gradient method is applied to solve the Newton equations. Based on the efficiency coefficient defined by Brent in Ref. [2], a efficiency ratio of Algorithm 2 to Algorithm 1 is introduced in Definition 3.2. This ratio is a function of n (the dimension) and Q_f (the cost to evaluate $f(x)$). Theorem 3.7 implies that the theoretical ratio of Algorithm 2 is larger than 1, that is, Algorithm 2 is always more efficient than Algorithm 1. Moreover, some typical values of the theoretical ratio are calculated and listed in Table 1. It shows that the improvement is significant at least for some cases. The validity of the theoretical ratios is supported by the practical ratios obtained by our numerical experiments, see Tables 2–5.

Acknowledgements The work was supported by the Key National Nature Science Foundation (Grant No.10631070) and the Mathematics and Physics Foundation of Beijing University of Technology (Grant No.Kz0603200381).

References

1. Bartholomew-Biggs, M., Brown, S., Christianson, B., Dixon, L.C.W.: Automatic differentiation of algorithms. J. Comput. Appl. Math. **12**, 171–190 (2000)
2. Brent, R.: Some efficient algorithms for solving systems of nonlinear equation. SIAM J. on Numerical Analy. **10**, 327–344 (1973)
3. Bucker, H.M., Corliss, G., Hovland, P., Naumann, U., Norris, B. (eds.): Automatic differentiation: Applications, Theory, and Tools. Lecture Notes in Computational Science and Engineering. Springer,Berlin Heldelberg New York (2005)
4. Deng, N., Xue, Y., Zhang, J.: An inexact Newton method derived from efficiency analysis. J. Glob. Optom. **31**(2), 287–315 (2005)
5. Griewank, A.: Evaluating Derivatives Principles and Techniques of Algorithmic Differentiation. Frontiers in Applied Mathematics 19. SIAM, Philadephia (2000)
6. Moré, J.J., Garbow, B.S., Hillstrom, K.E.: Testing unconstrained optimization software. ACM Trans. Math. Softw. **7**, 17–41 (1981)
7. Steihaug, T.: The conjugate gradient method and trust region in large scale optimization. SIAM J. Numerical Anal. **20**, 626–637 (1983)
8. Toint, P.L. : Towards an Efficient Sparsity Exploiting Newton Method for Minimization. In: Duff, I.S. (ed.) Sparse Matrices and Their Uses, pp. 57–88. Academic Press, London, UK (1981)