NEW ADAPTIVE BARZILAI–BORWEIN STEP SIZE AND ITS APPLICATION IN SOLVING LARGE-SCALE OPTIMIZATION PROBLEMS

TING LI\textsuperscript{1} and ZHONG WAN\textsuperscript{1}

(Received 8 February, 2018; accepted 30 July, 2018; first published online 3 December 2018)

Abstract

We propose a new adaptive and composite Barzilai–Borwein (BB) step size by integrating the advantages of such existing step sizes. Particularly, the proposed step size is an optimal weighted mean of two classical BB step sizes and the weights are updated at each iteration in accordance with the quality of the classical BB step sizes. Combined with the steepest descent direction, the adaptive and composite BB step size is incorporated into the development of an algorithm such that it is efficient to solve large-scale optimization problems. We prove that the developed algorithm is globally convergent and it R-linearly converges when applied to solve strictly convex quadratic minimization problems. Compared with the state-of-the-art algorithms available in the literature, the proposed step size is more efficient in solving ill-posed or large-scale benchmark test problems.


Keywords and phrases: nonlinear program, step size, algorithm, convergence, large-scale optimization.

1. Introduction

It is well known that large-scale optimization techniques are applied more and more to the fields of industrial engineering and management sciences, such as signal processing [25], machine learning [22] and newsboy problems [5, 20, 28]. On the other hand, it is still a challenge to develop an efficient numerical algorithm for solving these large-scale optimization problems, especially for improvement of computational efficiency which meets practical requirements [26, 29]. In the past two decades, it has attracted much attention from applied mathematicians and experts in engineering, since the existing optimizers cannot be directly applied to solve these problems.

We consider the following large-scale optimization problem:

\[
\min f(x), \quad x \in \mathbb{R}^n,
\]  

(1.1)
where \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is continuously differentiable and \( n \geq 1000 \) in general. To solve problem (1.1), a popular iterative format is constructed as

\[
x_{k+1} = x_k + \alpha_k d_k,
\]

where \( \alpha_k \) is a step size and \( d_k \) is a search direction. Thus, \( \{x_k\} \) is an approximate solution sequence of problem (1.1).

One of the most popular choices for the search directions is generated by the Newton method or the quasi-Newton method (see the articles by Dennis and Moré [12] and Wan et al. [24]). Although these methods have superlinear convergence, they are not suitable for solving large-scale optimization problems, since it is difficult to compute and store the (approximate) Hessian matrices of \( f \) at each iteration (see the articles by Deng and Wan [11] and Huang et al. [18]). To overcome the drawbacks in the Newton-type or the quasi-Newton-type methods, numerous spectral gradient methods [6, 17, 19] and conjugate gradient methods [11] have been presented to solve (1.1). Numerical experiments are often employed to show the advantages of their numerical efficiency in solving large-scale optimization problems.

The most direct and the simplest search direction is \( d_k = -\nabla f(x_k) \), which is referred to as the steepest descent method. However, owing to its poor convergence rate [17, 18], this method is not often applicable to solve problem (1.1). To exploit its advantages to the full, choice of a suitable step size is often necessary such that its numerical efficiency may beat that of the other methods. Actually, it has been shown that the steepest descent method with the Barzilai–Borwein (BB) step size is R-superlinearly convergent for the two-dimensional quadratic minimization problems [2]. For quadratic minimization problems with any dimension, it has also been proved that the method is still globally convergent [21] and the convergence rate is R-linear [10].

The so-called BB step sizes were first proposed by Barzilai and Borwein [2], where two formats of the step sizes were constructed by approximations to the secant equations. Specifically, the BB step sizes are given by

\[
\alpha_k^{BB1} = \frac{s_{k-1}^T y_{k-1}}{s_{k-1}^T y_{k-1}} \quad \text{(1.2)}
\]

and

\[
\alpha_k^{BB2} = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}, \quad \text{(1.3)}
\]

where \( s_{k-1} = x_k - x_{k-1} \) and \( y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1}) \). Denote \( g_k = \nabla f(x_k) \). Then \( y_{k-1} = g_k - g_{k-1} \). The two classical BB step sizes, \( \alpha_k^{BB1} \) and \( \alpha_k^{BB2} \), have been successfully applied for solving many problems, such as convex constrained optimization [3, 9], nonnegative matrix factorization [15], compressive sensing [14] and image restoration [4].
Inspired by the BB methods [2], Zhou et al. [30] presented an adaptive Barzilai and Borwein (ABB) step size, given by

$$
\alpha_{k}^{ABB} = \begin{cases} 
\alpha_{k}^{BB2} & \text{if } \alpha_{k}^{BB2} / \alpha_{k}^{BB1} < \kappa, \\
\alpha_{k}^{BB1} & \text{otherwise},
\end{cases}
$$

where $\kappa \in (0, 1)$. It is clear that the above ABB method can adaptively choose a small step size or a large step size at each iteration. Although the ABB method in [30] is still linearly convergent in the quadratic case, numerical experiments on some typical test problems indicate that it outperforms the classical BB methods. Huang et al. [16] used the step size (1.4) in nonnegative matrix factorization, combined with a descent direction on the basis of a projected gradient strategy.

Dai et al. [8] conducted an analysis on the geometrical mean of $\alpha_{k}^{BB1}$ and $\alpha_{k}^{BB2}$ for the two-dimensional strictly convex quadratic minimization problems, and R-superlinear convergence was established. Very recently, using a new approximation model, Huang and Wan [17] also obtained the BB step size

$$
\alpha_{k}^{NBB} = \sqrt{\frac{s_{k-1}^T s_{k-1}}{y_{k-1}^T y_{k-1}}} = \sqrt{\alpha_{k}^{BB1} / \alpha_{k}^{BB2}}. 
$$

(1.5)

It turns out to be the geometrical mean of $\alpha_{k}^{BB1}$ and $\alpha_{k}^{BB2}$, which was directly presented by Dai et al. [8]. Apart from their results, the global convergence has been established, as the step size (1.5) is applied to solving the quadratic minimization problems with any size of dimension [17]. Furthermore, based on a new nonmonotone line search, $\alpha_{k}^{NBB}$ was employed to develop an efficient algorithm for solving nonsmooth and nonlinear systems of equations [17].

Motivated by the superlinear convergence behaviour of the BB methods in the literature, here we intend to construct a new adaptive and composite BB step size by the weighted mean of two classical BB step sizes. To integrate the advantages of the existing BB step sizes, we update the weights at each iteration in accordance with the quality of two types of BB step sizes. Thus, our step size is adaptive and composite, which is different from the other BB step sizes. We are also interested in the convergence properties and numerical performance of the developed algorithm in this paper, which utilizes the simplest gradient method and the proposed step size. Owing to lower cost of computation and storage, the developed algorithm presented in the paper is suitable for solving large-scale optimization problems.

The rest of this paper is organized as follows. In the next section, we present the new adaptive and composite BB step size and develop an algorithm. In Section 3, convergence theory is established as the developed algorithm is applied to solve quadratic minimization problems. Numerical performance of the algorithm is reported in Section 4. Finally, we draw some conclusions in Section 5.
2. Adaptive and composite BB step size and algorithm

In this section, we propose a new adaptive and composite BB step size. Then, combined with the steepest descent direction, a new gradient algorithm has been developed.

Following Barzilai and Borwein [2], we know that $\alpha_k^{BB1}$ and $\alpha_k^{BB2}$ are the optimal solutions of the optimization problems

$$\min_{\alpha} ||\alpha^{-1}s_{k-1} - y_{k-1}||^2 \quad \text{and} \quad \min_{\alpha} ||\alpha y_{k-1} - s_{k-1}||^2,$$

respectively.

Clearly, compared with each other, the potential inferiority of $\alpha_k^{BB1}$ and $\alpha_k^{BB2}$ can be observed in

$$||\alpha_k^{BB1}y_{k-1} - s_{k-1}||^2 \geq ||\alpha_k^{BB2}y_{k-1} - s_{k-1}||^2$$

and

$$||(\alpha_k^{BB2})^{-1}s_{k-1} - y_{k-1}||^2 \geq ||(\alpha_k^{BB1})^{-1}s_{k-1} - y_{k-1}||^2,$$

respectively. Therefore, to integrate the individual advantages of $\alpha_k^{BB1}$ and $\alpha_k^{BB2}$, we construct a new step size

$$\alpha_k^{CBB} = \frac{R_2}{R_1 + R_2} \alpha_k^{BB1} + \frac{R_1}{R_1 + R_2} \alpha_k^{BB2}, \quad (2.1)$$

where

$$\begin{align*}
R_1 &= ||\alpha_k^{BB1}y_{k-1} - s_{k-1}||^2 - ||\alpha_k^{BB2}y_{k-1} - s_{k-1}||^2, \\
R_2 &= ||(\alpha_k^{BB2})^{-1}s_{k-1} - y_{k-1}||^2 - ||(\alpha_k^{BB1})^{-1}s_{k-1} - y_{k-1}||^2. \\
\end{align*} \quad (2.2)$$

Note that

$$||\alpha_k^{BB2}y_{k-1} - s_{k-1}||^2 = 0, \quad ||(\alpha_k^{BB1})^{-1}s_{k-1} - y_{k-1}||^2 = 0,$$

$$R_1 = ||\alpha_k^{BB1}y_{k-1} - s_{k-1}||^2, \quad R_2 = ||(\alpha_k^{BB2})^{-1}s_{k-1} - y_{k-1}||^2.$$

Denote $\mu_k = R_2/(R_1 + R_2)$. Then the equation (2.1) can be written as

$$\alpha_k^{CBB} = \mu_k \alpha_k^{BB1} + (1 - \mu_k) \alpha_k^{BB2}, \quad (2.3)$$

where $0 \leq \mu_k \leq 1$. Observe that $\alpha_k^{CBB}$ is the weighted mean of $\alpha_k^{BB1}$ and $\alpha_k^{BB2}$. Therefore, $\alpha_k^{CBB}$ is a composite step size. On the other hand, the weight $\mu_k$ may be different at each iteration, which is updated by integrating the advantages of $\alpha_k^{BB1}$ and $\alpha_k^{BB2}$. From this viewpoint, $\alpha_k^{CBB}$ is adaptive.

**Remark 2.1.** The step size $\alpha_k^{ABB}$ in (1.4) is also called the ABB step size [30], since one can choose a small step size $\alpha_k^{BB2}$ or a large step size $\alpha_k^{BB1}$ at each iteration. However, by definition, the meaning of the word “adaptive” in our paper is different from that of Zhou et al. [30]. In particular, our step size $\alpha_k^{CBB}$ in (2.3) can integrate the advantages of $\alpha_k^{BB1}$ and $\alpha_k^{BB2}$, since it is the weighted mean of $\alpha_k^{BB1}$ and $\alpha_k^{BB2}$ based on a measure of their qualities by (2.2), rather than an either–or decision as done by Zhou et al. [30].
Remark 2.2. Additionally, as shown by Barzilai and Borwein [2], both \( \alpha^\text{BB1}_k I \) and \( \alpha^\text{BB2}_k I \) are scalar approximations to the inverse of \( \nabla^2 f(x_k) \). From (2.3), we know that \( \alpha^\text{CBB}_k \) is also an approximation to the inverse of \( \nabla^2 f(x_k) \).

In virtue of an earlier idea [30], we modify \( \alpha^\text{ABB}_k \) in (1.4) as

\[
\alpha^\text{CABB}_k = \begin{cases} 
\alpha^\text{BB2}_k & \text{if } \alpha^\text{BB2}_k / \alpha^\text{BB1}_k < \kappa, \\
\alpha^\text{CBB}_k & \text{otherwise.}
\end{cases}
\]  

(2.4)

Similar to \( \alpha^\text{ABB}_k \), note that \( \alpha^\text{CABB}_k \) is also adaptive.

With the help of the new BB step size (2.3) or (2.4), we develop Algorithm 1 on the basis of the simplest steepest descent method.

Algorithm 1 New gradient method with adaptive and composite BB step size

Step 0 (Initialization). Given initial \( x_0 \in \mathbb{R}^n \) and \( \alpha_0 > 0 \in \mathbb{R} \), choose a tolerance \( \varepsilon > 0 \). Set \( k := 0 \).

Step 1 (Termination). If \( \| \nabla f(x_k) \| \leq \varepsilon \), then the algorithm stops. Otherwise, go to Step 2.

Step 2 (Search direction). Compute \( d_k := -\nabla f(x_k) \). Go to Step 3.

Step 3 (Update). Set \( x_{k+1} := x_k + \alpha_k d_k \).

Step 4 (CBB step size). Compute \( s_k, y_k \) and \( \alpha_{k+1}^\text{CBB} \) by (2.3). Go to Step 5.

Step 5 (Update). Set \( k := k + 1 \). Go to Step 1.

A special case of problem (1.1) is the following quadratic minimization model:

\[
\min f(x) = \frac{1}{2} x^T Ax - b^T x, \quad x \in \mathbb{R}^n,
\]  

(2.5)

where \( A \in \mathbb{R}^{n \times n} \) is a symmetric and positive-definite (SPD) matrix and \( b \in \mathbb{R}^n \) is a given vector. In the next section, we will give some convergence analyses about Algorithm 1 for solving model (2.5). We first present a basic property for \( \alpha^\text{CBB}_k \) and \( \alpha^\text{CABB}_k \).

Let \( x_k, x_{k-1} \in \mathbb{R}^n \) be two given points. Then, since \( g_k = \nabla f(x_k) = Ax_k - b \), we have \( y_{k-1} = As_{k-1} \). Now, from (1.2) and (1.3),

\[
\alpha^\text{BB1}_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T As_{k-1}},
\]  

(2.6)

\[
\alpha^\text{BB2}_k = \frac{s_{k-1}^T A s_{k-1}}{s_{k-1}^T A^2 s_{k-1}}.
\]  

(2.7)

Proposition 2.3. Let \( \alpha_k \) be defined by (2.3) or (2.4). Then, for any SPD matrix \( A \), the following inequalities hold:

\[
\begin{cases}
0 < \lambda_n^{-1} \leq \alpha^\text{CBB}_k \leq \lambda_1^{-1} & \text{for all } k, \\
0 < \lambda_n^{-1} \leq \alpha^\text{CABB}_k \leq \lambda_1^{-1} & \text{for all } k,
\end{cases}
\]  

(2.8)

where \( \lambda_1 \) and \( \lambda_n \) are the smallest and the largest eigenvalues of \( A \), respectively.
Proof. From (2.6), (2.7) and by definition [23, Definition 1.2.9], it follows that 
\((\alpha_k^{BB1}I)^{-1}\) is the Rayleigh quotient of \(A\) at \(s_{k-1}\) and 
\((\alpha_k^{BB2}I)^{-1}\) is that of \(A\) at \(\sqrt{A}s_{k-1}\). Since the matrix \(A\) is SPD, using the properties of the Rayleigh quotient [23, Theorem 1.2.10],

\[0 < \lambda_1 \leq (\alpha_k^{BB1})^{-1} \leq \lambda_n, \quad 0 < \lambda_1 \leq (\alpha_k^{BB2})^{-1} \leq \lambda_n \quad \text{for all } k,\]

where \(\lambda_1\) and \(\lambda_n\) are the smallest and the largest eigenvalues of \(A\), respectively. Consequently,

\[0 < \lambda_n^{-1} \leq \alpha_k^{BB1} \leq \lambda_1^{-1}, \quad 0 < \lambda_n^{-1} \leq \alpha_k^{BB2} \leq \lambda_1^{-1} \quad \text{for all } k.\]

Thus, for \(\alpha_k^{CBB}\) defined by (2.3),

\[0 < \lambda_n^{-1} = \mu_k\lambda_1^{-1} + (1 - \mu_k)\lambda_1^{-1} \leq \alpha_k^{CBB} \leq \mu_k\lambda_1^{-1} + (1 - \mu_k)\lambda_1^{-1} = \lambda_1^{-1}.\]

Since both \(\alpha_k^{CBB}\) and \(\alpha_k^{BB2}\) satisfy the inequalities in (2.8), it follows from the definition of \(\alpha_k^{CABB}\) that

\[0 < \lambda_n^{-1} \leq \alpha_k^{CABB} \leq \lambda_1^{-1} \quad \text{for all } k,\]

which completes the proof. \(\square\)

3. Convergence analysis

In this section, we establish the global convergence of Algorithm 1 as it is applied to solve the quadratic minimization problems. In addition, its R-linear convergence is also established. Before establishing the convergence theory, we first state the following results.

Lemma 3.1. Let \(f\) be defined by (2.5) and let \(x^*\) be the unique minimizer of \(f\). Suppose that \(\{x_k\}\) is a sequence generated by Algorithm 1. Denote \(e_k = x^* - x_k\). Then:

1. \(Ae_k = d_k;\)
2. \(e_{k+1} = \alpha_k((1/\alpha_k)I - A)e_k.\)

Lemma 3.2. Let \(f\) be defined by (2.5). Let \(\{v_1, v_2, \ldots, v_n\}\) be the orthonormal eigenvectors of \(A\), corresponding to the eigenvalues \(\{\lambda_1, \lambda_2, \ldots, \lambda_n\}\), respectively, where \(\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n\). Suppose that \(\{x_k\}\) is a sequence generated by Algorithm 1. Then

\[e_{k+1} = \sum_{i=1}^{n} c_i^{(k+1)} v_i, \quad (3.1)\]

where

\[c_i^{(k+1)} = \prod_{j=0}^{k} (1 - \alpha_j\lambda_i) c_i^0. \quad (3.2)\]

Proofs of Lemmas 3.1 and 3.2 are similar to those of Huang and Wan [17] and hence omitted here.
Remark 3.3. By Lemma 3.2, we know that the convergence of the sequence \( \{e_k\} \) depends on the behaviour of each of the sequences \( \{c_i^{(k)}\}, i = 1, 2, \ldots, n \). In Theorem 3.6, we prove that all of these sequences converge to zero, that is, \( e_k \to 0 \) as \( k \to \infty \).

Lemma 3.4. Let \( \{c_1^{(k)}\} \) be a sequence defined by (3.2). Then \( c_1^{(k)} \) converges \( Q \)-linearly to zero with convergence factor \( \hat{c} = 1 - (A_1/A_n) \).

Since \( A_k^{CBB} \) satisfies (2.8), it follows directly from [21] that Lemma 3.4 is true. Using Lemmas 3.1, 3.2 and 3.4, we now further prove the following result.

Lemma 3.5. Let \( \{c_1^{(k)}\}, \{c_2^{(k)}\}, \ldots, \{c_l^{(k)}\} \) be the sequences defined by (3.2), \( l = 1, \ldots, n \). If all the sequences \( \{c_1^{(k)}\}, \{c_2^{(k)}\}, \ldots, \{c_l^{(k)}\} \) converge to zero, then

\[
\liminf_{k \to \infty} |c_{l+1}^{(k)}| = 0.
\]

Proof. By contradiction, suppose that there exists a constant \( \epsilon > 0 \) such that for all \( k \),

\[
(c_{l+1}^{(k)})^2 \lambda_{l+1}^2 > \epsilon.
\] (3.3)

From Lemma 3.1,

\[
s_k = x_{k+1} - x_k = \alpha_k d_k = \alpha_k A e_k \quad \text{and} \quad y_k = A s_k = \alpha_k A^2 e_k.
\]

Using equation (3.1) and the orthonormality of \( \{v_1, v_2, \ldots, v_n\} \) yields

\[
\alpha_{k+1}^{BB1} = \frac{s_k^T s_k}{s_k^T y_k} = \frac{(A e_k)^T (A e_k)}{(A e_k)^T (A^2 e_k)} = \frac{\sum_{i=1}^n (c_i^{(k)})^2 \lambda_i^2}{\sum_{i=1}^n (c_i^{(k)})^2 \lambda_i^2} \quad (3.4)
\]

and

\[
\alpha_{k+1}^{BB2} = \frac{s_k^T y_k}{y_k^T y_k} = \frac{(A e_k)^T (A^2 e_k)}{(A^2 e_k)^T (A^2 e_k)} = \frac{\sum_{i=1}^n (c_i^{(k)})^2 \lambda_i^2}{\sum_{i=1}^n (c_i^{(k)})^2 \lambda_i^2}.
\]

Since the sequences \( \{c_1^{(k)}\}, \{c_2^{(k)}\}, \ldots, \{c_l^{(k)}\} \) are all convergent to zero, there exists \( \hat{k} \) sufficiently large such that for any \( k \geq \hat{k} \),

\[
\sum_{i=1}^l (c_i^{(k)})^2 \lambda_i^2 < \frac{\epsilon}{2}. \quad (3.5)
\]

From (3.4) and (3.5),

\[
\alpha_{k+1}^{BB1} < \frac{\epsilon/2 + \sum_{i=l+1}^n (c_i^{(k)})^2 \lambda_i^2}{\sum_{i=l+1}^n (c_i^{(k)})^2 \lambda_i^2} \quad (3.6)
\]

Note that the function \( q : R^+ \to R \), given by

\[
q(x) = \frac{a + x}{bx}, \quad a, b > 0,
\]
is monotonically decreasing in $(0, \infty)$. Combined with (3.6) and, since
\[
\sum_{i=l+1}^{n} (c_i^{(k)})^2 \lambda^2_i \geq \left( \sum_{i=l+1}^{n} (c_i^{(k)})^2 \lambda^2_i \right) \lambda_{l+1} \geq \sum_{i=l+1}^{n} (c_i^{(k)})^2 \lambda^2_i \geq (c_{l+1}^{k})^2 \lambda_{l+1}^2 > \epsilon,
\]
we have
\[
\alpha_{BB1}^{(k+1)} < \frac{(\epsilon/2) + \sum_{i=l+1}^{n} (c_i^{(k)})^2 \lambda^2_i}{\sum_{i=l+1}^{n} (c_i^{(k)})^2 \lambda_{l+1}^2} \leq \frac{(\epsilon/2) + \epsilon}{\epsilon \lambda_{l+1}} = \frac{3}{2} \lambda_{l+1}^{-1}.
\]
Consequently, for all $k \geq \hat{k}$,
\[
\lambda_{n}^{-1} \leq \alpha_{k+1}^{BB1} \leq \frac{3}{2} \lambda_{l+1}^{-1}.
\]
Similarly, we can prove that for all $k \geq \hat{k}$,
\[
\alpha_{BB2}^{(k+1)} < \frac{(\epsilon/2) \lambda_{l+1} + \sum_{i=l+1}^{n} (c_i^{(k)})^2 \lambda^3_i}{\sum_{i=l+1}^{n} (c_i^{(k)})^2 \lambda_{l+1}^4} < \frac{(\epsilon/2) \lambda_{l+1} + \epsilon \lambda_{l+1}}{\epsilon \lambda_{l+1}^2} = \frac{3}{2} \lambda_{l+1}^{-1}
\]
and, therefore,
\[
\lambda_{n}^{-1} \leq \alpha_{k+1}^{BB2} \leq \frac{3}{2} \lambda_{l+1}^{-1}.
\]
By definition of $\alpha_k$ in (2.3), it follows that for all $k \geq \hat{k} + 1$,
\[
\lambda_{n}^{-1} \leq \alpha_k \leq \frac{3}{2} \lambda_{l+1}^{-1}.
\]
Using the fact that $c_{l+1}^{(k+1)} = (1 - \alpha_{l+1} \lambda_{l+1})c_{l+1}^{(k)}$, we know that for all $k \geq \hat{k} + 1$,
\[
|c_{l+1}^{(k+1)}| \leq |1 - \alpha_{l+1} \lambda_{l+1}||c_{l+1}^{(k)}| \leq \hat{c}|c_{l+1}^{(k)}|,
\]
where
\[
\hat{c} = \max\left\{1 - \frac{1}{2}, 1 - \frac{\alpha_{l+1}}{\lambda_{l+1}}\right\} < 1.
\]
Thus, $c_{l+1}^{(k)}$ converges to zero as $k \to \infty$, which contradicts (3.3). Therefore, we have proved that
\[
\liminf_{k \to \infty} |c_{l+1}^{(k)}| = 0.
\]

Based on Proposition 2.3 and Lemmas 3.1, 3.2 and 3.5, now we establish the global convergence theory of Algorithm 1.

**Theorem 3.6.** Let $\{x_k\}$ be a sequence generated when Algorithm 1 is used to solve model (2.5). Let $x_*$ be the unique minimizer of (2.5). Then either there exists an integer $j$ such that $x_j = x_*$, or the sequence $\{x_k\}$ converges to $x_*$. 

**Proof.** If there exists an integer $j$ such that $x_j = x_*$, then the result holds. Suppose that there is no finite integer $j$ such that $x_j = x_*$. We are going to prove that the sequence $\{e_k\}$ converges to zero.
From (3.1) and the orthonormality of the eigenvectors,

$$\|e_k\|^2 = \sum_{i=1}^{n} (c_i^{(k)})^2.$$  

Thus, the error sequence \( \{e_k\} \) converges to zero if and only if each of the sequences \( \{c_i^{(k)}\} \), \( i = 1, 2, \ldots, n \), converges to zero.

By contradiction, we assume that some of the sequences of \( \{c_i^{(k)}\} \) do not converge to zero. Suppose that \( \{c_i^{(p)}\} \) is the first sequence which does not converge to zero. By Lemma 3.4, we have \( p \geq 2 \). Since all the sequences \( \{c_i^{(k)}\} \), \( i = 1 \), \( 2 \), \ldots, \( n \), converge to zero, for a given \( \epsilon > 0 \), there exists \( \hat{k} \) sufficiently large such that for all \( k \geq \hat{k} \), the following inequality holds:

$$\sum_{i=1}^{p-1} (c_i^{(p)})^2 \lambda_i^2 < \frac{\epsilon}{2^p}. \quad (3.7)$$

By Lemma 3.5, we know that \( \lim \inf_{k \to \infty} |c_p^{(k)}| = 0 \). Thus, there exists \( k_p \geq \hat{k} \) such that

$$\lambda_{p}^2 < \epsilon.$$  

Let us analyse the properties of \( \{c_i^{(k)}\} \) for \( k \geq k_p \). Our aim is to prove that there exists a positive constant \( C \) such that

$$\lambda_{p}^2 < \frac{\epsilon}{2^p} \quad \forall \ k \geq k_p.$$  

We conduct the analysis by the following three cases.

**Case (a):** \( k_p \leq k \leq k_0 - 1 \).

Let \( k_0(> k_p) \) be the first positive integer satisfying

$$\lambda_{p}^2 < \epsilon.$$  

Thus, for all \( k \leq k_0 - 1 \),

$$\max_{k_p \leq k \leq k_0 - 1} |(c_p^{(k)})^2| < \frac{\epsilon}{\lambda_p^2}. \quad (3.9)$$

**Case (b):** \( k_0 \), \( k_0 + 1 \) and \( k_0 + 1 < k \leq j + 1 \).

If \( j \) is the first positive integer greater than \( k_0 \) that satisfies

$$\lambda_{p}^2 > \epsilon,$$  

then we can replace \( \hat{k} \) in Lemma 3.5 by \( k_0 \) and, for \( k_0 + 1 \leq k \leq j \), it follows from (3.7) and (3.10) that

$$\alpha_k \leq \frac{3}{2} \lambda_{p}^{-1}.$$  

(3.11)
From Lemma 3.2, we obtain that for all \( k' > 0 \),
\[
(c_p^{(k+1)})^2 = (1 - \alpha_k \lambda_p)^2 (c_p^{(k)})^2 \\
\leq \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^2 (c_p^{(k)})^2 \\
\leq \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^4 (c_p^{(k-1)})^2.
\] (3.12)

As for \( k_0 \), from (3.8) and (3.12),
\[
(c_p^{(k_0)})^2 \leq \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^2 (c_p^{(k_0-1)})^2 \\
\leq \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^2 \epsilon \frac{1}{\lambda_2^2}. \tag{3.13}
\]

As for \( k_0 + 1 \), from (3.8) and (3.12),
\[
(c_p^{(k_0+1)})^2 \leq \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^4 \epsilon \frac{1}{\lambda_2^2}. \tag{3.14}
\]

Since \( \alpha_k \) satisfies (3.11), where \( k_0 + 1 \leq k \leq j \),
\[
(c_p^{(k+1)})^2 \leq \max \left\{ \frac{1}{2}, 1 - \frac{\lambda_p}{\lambda_n} \right\} (c_p^{(k)})^2 \\
\leq (c_p^{(k+1)})^2 \\
\leq \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^4 \epsilon \frac{1}{\lambda_2^2}. \tag{3.15}
\]

From the inequalities (3.9) and (3.13)–(3.15),
\[
\max_{k_p \leq k \leq j+1} (c_p^{(k)})^2 \leq \max \left\{ 1, \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^2, \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^4 \epsilon \frac{1}{\lambda_2^2} \right\}.
\]

**Case (c):** \( k \geq j + 2 \).

Repeating the processes of Cases (a) and (b), as for \( k \geq j + 2 \), we can also prove that
\[
\max_{k \geq j+2} (c_p^{(k)})^2 \leq \max \left\{ 1, \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^2, \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^4 \epsilon \frac{1}{\lambda_2^2} \right\}.
\]

Combining Cases (a)–(c), we conclude that there exists
\[
C = \max \left\{ 1, \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^2, \left( \frac{\lambda_n - \lambda_1}{\lambda_1} \right)^4 \right\}
\]
such that
\[(c_p^{(k)})^2 \leq \frac{C}{\lambda_p^2} \quad \text{for all } k \geq k_p.\]

Define
\[M_\epsilon = \sup_{k \geq k_p} (c_p^{(k)})^2.\]

Then, as \(\epsilon(>0) \to 0\), \(M_\epsilon \to 0\), which implies that \(\limsup_{k \to \infty} (c_p^{(k)})^2 = 0\). Consequently,
\[\lim_{k \to \infty} |c_p^{(k)}| = 0,
which contradicts that \(\{c_p^{(k)}\}\) does not converge to zero. Therefore, the sequence \(\{e_k\}\) converges to zero. This completes the proof. \(\square\)

Next, we study local convergence of Algorithm 1 and establish its R-linear convergence result.

**Theorem 3.7.** Let model (2.5) be solved by Algorithm 1 and \(\{x_k\}\) be a sequence generated by this algorithm. Then either \(g_k = 0\) for some finite \(k\), or the sequence \(\|g_k\|\) R-linearly converges to zero.

**Proof.** Recall that \(g_k = \nabla f(x_k)\), where \(f\) is the function as defined in Section 1. Since \(s_k = x_{k+1} - x_k = -\alpha_k g_k\) and \(y_k = A s_k\), it directly follows from (2.6) and (2.7) that
\[\alpha_{BB1}^{(k+1)} = \frac{g_k^T g_k}{g_k^T A g_k} \quad \text{and} \quad \alpha_{BB2}^{(k+1)} = \frac{g_k^T A g_k}{g_k^T A^2 g_k}.
\]

By Lemma 3.2,
\[g_k = \sum_{i=1}^{n} g_i^{(k)} v_i,
where \(g_i^{(k)} = (1 - \alpha_{k-1} \lambda_i) g_i^{(k-1)}\). The orthonormality of \(\{v_1, v_2, \ldots, v_n\}\) yields
\[g_k^T g_k = \sum_{i=1}^{n} (g_i^{(k)})^2, \quad A g_k = \sum_{i=1}^{n} \lambda_i g_i^{(k)} v_i.
\]

Consequently,
\[\alpha_{BB1}^{(k+1)} = \frac{\sum_{i=1}^{n} (g_i^{(k)})^2}{\sum_{i=1}^{n} (g_i^{(k)})^2 \lambda_i} \quad \text{and} \quad \alpha_{BB2}^{(k+1)} = \frac{\sum_{i=1}^{n} (g_i^{(k)})^2 \lambda_i}{\sum_{i=1}^{n} (g_i^{(k)})^2 \lambda_i^2}.
\]

Since \(g_k = A x_k - b = -A e_k\), by Lemma 3.2,
\[g_k = \sum_{i=1}^{n} g_i^{(k)} v_i = -\sum_{i=1}^{n} c_i^{(k)} \lambda_i v_i.
Thus, \(g_i^{(k)} = -c_i^{(k)} \lambda_i\).
From the proof of Lemma 3.5, we know that for any integer \( l, l = 1, 2, \ldots, n - 1, \) and for a real number \( \epsilon > 0, \)

\[
\lambda_n \geq (a_k^{CBB})^{-1} \geq \frac{2}{3} \lambda_{l+1}
\]

in the case that

\[
\begin{align*}
\sum_{i=1}^{l} (c_i^{(k)})^2 \lambda_i^2 & = \sum_{i=1}^{l} (g_i^{(k)})^2 < (1/2)\epsilon < \epsilon, \\
(c_{l+1}^{(k)})^2 \lambda_{l+1}^2 & = (g_{l+1}^{(k)})^2 > \epsilon.
\end{align*}
\]

Therefore, \( a_k^{CBB} \) satisfies Dai’s property [7, Property A] and, similar to the proof of a theorem there [7, Theorem 4.1], we get the R-linear convergence of Algorithm 1. \( \Box \)

**Remark 3.8.** As in [21, Lemma 2], under the sufficient condition \( \lambda_n < 2\lambda_1 \) for \( A \), we can also prove that the sequence \( \{x_k\} \) Q-linearly converges to \( x_\star \). The R-linear or Q-linear convergence is defined by Sun and Yuan [23, Section 1.5.4].

Note that \( a_k^{BB_1}, a_k^{BB_2}, a_k^{ABB}, a_k^{CBB} \) and \( a_k^{CABB} \) can all be rewritten in a unified form:

\[
a_k = \mu_k \frac{g_{k-1}^T A \rho(k) g_{k-1}}{g_{k-1}^T A (\rho(k)+1) g_{k-1}} + (1 - \mu_k) \frac{g_{k-1}^T A \gamma(k) g_{k-1}}{g_{k-1}^T A (\gamma(k)+1) g_{k-1}},
\]

(3.16)

where \( \rho(k), \gamma(k) \in \{0, 1\}, \mu_k \in [0, 1] \) and \( \rho(k) \neq \gamma(k) \). At the end of this section, we will give the global and local convergence analyses for the unified step size (3.16).

**Corollary 3.9.** Let \( \{x_k\} \) be a sequence generated when Algorithm 1 is used to solve model (2.5), where the step size \( a_k \) is computed by (3.16). Let \( x_\star \) be the unique minimizer of (2.5). Then either there exists an integer \( j \) such that \( x_j = x_\star \), or the sequence \( \{x_k\} \) converges to \( x_\star \).

**Proof.** Since \( a_k^{BB_1}, a_k^{BB_2}, a_k^{ABB}, a_k^{CBB} \) and \( a_k^{CABB} \) all have the property of Proposition 2.3, if \( a_k \) is defined by (3.16), then

\[
0 < \lambda_n^{-1} \leq a_k \leq \lambda_1^{-1} \quad \text{for all } k.
\]

For any integer \( l, l = 1, 2, \ldots, n - 1, \) and any real number \( \epsilon > 0, \) if

\[
\begin{align*}
\sum_{i=1}^{l} (g_i^{(k)})^2 & < 1/2\epsilon, \\
(g_{l+1}^{(k)})^2 & > \epsilon
\end{align*}
\]

Then...
hold, then it follows from (3.16) that

$$\begin{align*}
\alpha_k &= \mu_k \frac{\sum_{i=1}^{n} (g_i^{(k)})^2 \lambda_i^{(k)}}{\sum_{i=1}^{n} (g_i^{(k)})^2 \lambda_i^{(k)+1}} + (1 - \mu_k) \frac{\sum_{i=1}^{n} (g_i^{(k)})^2 \lambda_i^{(k)+1}}{\sum_{i=1}^{n} (g_i^{(k)})^2 \lambda_i^{(k)+1}} \\
&\leq \mu_k \frac{\sum_{i=1}^{l} (g_i^{(k)})^2 \lambda_i^{(k)} + \sum_{i=l+1}^{n} (g_i^{(k)})^2 \lambda_i^{(k)+1}}{\sum_{i=1}^{n} (g_i^{(k)})^2 \lambda_i^{(k)+1}} \\
&\quad + (1 - \mu_k) \frac{\sum_{i=1}^{l} (g_i^{(k)})^2 \lambda_i^{(k)} + \sum_{i=l+1}^{n} (g_i^{(k)})^2 \lambda_i^{(k)+1}}{\sum_{i=1}^{n} (g_i^{(k)})^2 \lambda_i^{(k)+1}} \\
&\leq \mu_k \frac{(2k) \lambda_i^{(k)} + \epsilon \lambda_i^{(k)+1}}{\epsilon \lambda_i^{(k)+1}} + (1 - \mu_k) \frac{(2k) \lambda_i^{(k)} + \epsilon \lambda_i^{(k)+1}}{\epsilon \lambda_i^{(k)+1}} \\
&= \mu_k \frac{3}{\lambda_i^{(k)+1}} + (1 - \mu_k) \frac{3}{\lambda_i^{(k)+1}} = \frac{3}{2} (\lambda_i^{(k)+1})^{-1}.
\end{align*}$$

Thus, by recalling the proofs of Lemma 3.5 and Theorem 3.6, we complete the proof of Corollary 3.9. □

**Corollary 3.10.** Let model (2.5) be solved by Algorithm 1, where the step size $\alpha_k$ is computed by (3.16). Then either $g_k = 0$ for some finite $k$, or the sequence $\{\|g_k\|\}$ R-linearly converges to zero.

**Proof.** From the proof of Corollary 3.9, we have that [7, Property (A)] holds for the step size defined by (3.16). By [7, Theorem 4.1], the result in this corollary follows. □

### 4. Numerical experiments

In this section, we will test the numerical efficiency of Algorithm 1 in virtue of the adaptive and composite BB step size. Especially, we will compare our methods with the other ones available in the literature.

Specifically, we call Algorithm 1 the CBB method. If the step size in Algorithm 1 is computed by (1.2), (1.3), (1.4), (1.5) or (2.4), we call the corresponding algorithms the BB1, BB2, ABB, NBB and CABB methods, respectively. Our first aim is to investigate whether or not the CABB method outperforms the ABB method proposed in [30]. In order to reveal what are the advantages of the adaptive weight $\alpha_k$, we compare our method with a modified version of Algorithm 1, where $\mu_k$ is a fixed constant $\mu$. 

---

**Downloaded from https://www.cambridge.org/core. IP address: 35.160.27.221, on 30 Apr 2022 at 12:19:42, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/S1446181118000263**
All of the computer codes are written in MATLAB R2014a and numerical experiments are conducted on a PC with a 2.60 GHz CPU and a 4.00 GB memory-based operation system of Windows 7.

For simplicity of the statement, we introduce the following notation:

- **CBB(µ)**: the gradient method with \( \alpha_k = \alpha_{CBB}^k \) and \( \mu_k = \mu \);
- **ABB**: the gradient method with \( \alpha_k = \alpha_{ABB}^k \) and \( \kappa = 0.5 \);
- **C(0.8)**: the gradient method with \( \alpha_k = \alpha_{CABB}^k \), \( \mu_k = 0.8 \), and \( \kappa = 0.5 \);
- **CABB**: the gradient method with \( \alpha_k = \alpha_{CABB}^k \) and \( \kappa = 0.5 \);
- **P**: the serial number of the test problems of Andrei [1];
- **n**: the size of the problem;
- **Cd**: the condition number of the Hessian matrix of \( f \);
- **NI**: the number of iterations;
- **f_{\text{min}}**: the minimal value of the objective function;
- **T**: the CPU time (s) being accurate to the third decimal place;
- **F**: the number of iterations exceeds 10000 or \( f_{\text{min}} \) is not the minimal value.

### 4.1. Test by ill-posed quadratic problems

We first investigate the performance of all the algorithms as they are applied to solve the ill-posed quadratic minimization problems.

Consider the following test problem, suggested by Yuan [27]:

\[
\begin{align*}
f(x) &= (x - x_*)^T \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)(x - x_*) , \quad x \in \mathbb{R}^n, \\
\end{align*}
\]

where \( \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \) creates a square diagonal matrix with the elements of the vector \( (\lambda_1, \lambda_2, \ldots, \lambda_n)^T \) on the main diagonal.

We solve problem (4.1) for the cases \( n = 10, 10^2, 10^3, 10^4 \). We randomly choose \( x_* \) in the interval \((-5, 5)\), \( i = 1, 2, \ldots, n \), to randomly generate different test problems. Let \( \lambda_1 = 1 \). Denote by

\[\lambda_n = Cd = (10, 10^2, 10^3, 10^4, 10^5)\]

the so-called condition number of the Hessian matrix of the objective function \( f \). Let \( \lambda_i \) be randomly generated in the interval \((1, \lambda_n)\) in order to generate different test problems, \( i = 2, 3, \ldots, n - 1 \). For all the randomly generated test problems, we start with

\[
x_0 = \text{zeros}(n, 1), \quad \alpha_0 = \alpha_{SD}^0 = \frac{g_0^T g_0}{g_0^T \nabla^2 f(x_0) g_0}.
\]

The termination condition of all the algorithms is

\[
\|g_k\|_2 \leq 10^{-5}\|g_0\|_2.
\]

First, for each case with different values of \( n \) and \( \lambda_n \), 10 test problems are generated. Then we evaluate the numerical performance of each algorithm by the average number of iterations required by the algorithms. In Table 1, we report the average number of iterations (NI) required by the different algorithms. The underlined result, the smaller one in each row, indicates that the corresponding algorithm outperforms the others.

From the numerical results in Table 1, we observe the following results.
<table>
<thead>
<tr>
<th>$n$</th>
<th>Cd</th>
<th>BB1</th>
<th>BB2</th>
<th>NBB</th>
<th>CBB</th>
<th>ABB</th>
<th>CABB</th>
<th>C(0.8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>18.4</td>
<td>18.8</td>
<td>19</td>
<td>19</td>
<td>18.3</td>
<td>19.2</td>
<td>19.4</td>
</tr>
<tr>
<td>10</td>
<td>10$^2$</td>
<td>50.4</td>
<td>55.2</td>
<td>65</td>
<td>48.1</td>
<td>40.6</td>
<td>40.1</td>
<td>39.6</td>
</tr>
<tr>
<td>10</td>
<td>10$^3$</td>
<td>87</td>
<td>96.8</td>
<td>142.2</td>
<td>82.3</td>
<td>77.5</td>
<td>79.3</td>
<td>92.1</td>
</tr>
<tr>
<td>10</td>
<td>10$^4$</td>
<td>141.9</td>
<td>180.7</td>
<td>380.5</td>
<td>139.9</td>
<td>151.4</td>
<td>134.9</td>
<td>155.6</td>
</tr>
<tr>
<td>10</td>
<td>10$^5$</td>
<td>26.8</td>
<td>27</td>
<td>46.1</td>
<td>26.8</td>
<td>24.7</td>
<td>24.7</td>
<td>26.3</td>
</tr>
<tr>
<td>10$^2$</td>
<td>10</td>
<td>19.6</td>
<td>19.9</td>
<td>19.3</td>
<td>19.5</td>
<td>19.6</td>
<td>19.1</td>
<td>19.4</td>
</tr>
<tr>
<td>10$^2$</td>
<td>10$^2$</td>
<td>53.9</td>
<td>54.8</td>
<td>63.2</td>
<td>53.9</td>
<td>46.9</td>
<td>46.7</td>
<td>48.1</td>
</tr>
<tr>
<td>10$^2$</td>
<td>10$^3$</td>
<td>103</td>
<td>114.1</td>
<td>158.9</td>
<td>103.3</td>
<td>117</td>
<td>106.5</td>
<td>100.7</td>
</tr>
<tr>
<td>10$^2$</td>
<td>10$^4$</td>
<td>159.7</td>
<td>180.5</td>
<td>262.5</td>
<td>130.5</td>
<td>147.1</td>
<td>130.8</td>
<td>137.5</td>
</tr>
<tr>
<td>10$^2$</td>
<td>10$^5$</td>
<td>59.9</td>
<td>70.7</td>
<td>89.6</td>
<td>59.9</td>
<td>59.3</td>
<td>59.1</td>
<td>53.9</td>
</tr>
<tr>
<td>10$^3$</td>
<td>10</td>
<td>19</td>
<td>19.9</td>
<td>18.9</td>
<td>18.9</td>
<td>19</td>
<td>18.9</td>
<td>19</td>
</tr>
<tr>
<td>10$^3$</td>
<td>10$^2$</td>
<td>50.7</td>
<td>56.2</td>
<td>57.1</td>
<td>51.5</td>
<td>49</td>
<td>48.4</td>
<td>54</td>
</tr>
<tr>
<td>10$^3$</td>
<td>10$^3$</td>
<td>114.3</td>
<td>114.6</td>
<td>150.3</td>
<td>115.7</td>
<td>115</td>
<td>104.7</td>
<td>108</td>
</tr>
<tr>
<td>10$^3$</td>
<td>10$^4$</td>
<td>112.9</td>
<td>115.2</td>
<td>158.8</td>
<td>115.4</td>
<td>100.2</td>
<td>102.4</td>
<td>104.5</td>
</tr>
<tr>
<td>10$^3$</td>
<td>10$^5$</td>
<td>119.4</td>
<td>122.7</td>
<td>163.3</td>
<td>100.4</td>
<td>118.1</td>
<td>116.5</td>
<td>117.2</td>
</tr>
<tr>
<td>10$^4$</td>
<td>10</td>
<td>19</td>
<td>20</td>
<td>19.1</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>10$^4$</td>
<td>10$^2$</td>
<td>50.7</td>
<td>59</td>
<td>62.3</td>
<td>52.8</td>
<td>53.4</td>
<td>52.8</td>
<td>49.2</td>
</tr>
<tr>
<td>10$^4$</td>
<td>10$^3$</td>
<td>109.4</td>
<td>119.9</td>
<td>131.1</td>
<td>113.5</td>
<td>98.8</td>
<td>106.1</td>
<td>104.5</td>
</tr>
<tr>
<td>10$^4$</td>
<td>10$^4$</td>
<td>114.8</td>
<td>123.8</td>
<td>153.7</td>
<td>114.8</td>
<td>113.6</td>
<td>112.8</td>
<td>117.9</td>
</tr>
<tr>
<td>10$^4$</td>
<td>10$^5$</td>
<td>117.1</td>
<td>129.6</td>
<td>138.5</td>
<td>118.8</td>
<td>106.2</td>
<td>113.1</td>
<td>113.4</td>
</tr>
</tbody>
</table>

(1) The CBB, ABB, C(0.8) and CABB methods are better than the BB1, BB2 and NBB methods, and the BB1 method is better than the BB2 and NBB methods. Out of 20 cases, there are 12, 16, 15 and 16 cases where the CBB, ABB, C(0.8) and CABB methods perform better than the BB1 method, respectively.

(2) The ABB, C(0.8) and CABB methods perform better than the CBB method. In addition, the CBB, C(0.8) and CABB methods outperform the ABB method in eight, 10 and 14 out of 20 cases, respectively.

In Table 2, we further report the average number of iterations required by the CBB($\mu$) methods with different fixed values of $\mu$, in order to investigate the influence of the parameter $\mu$ on the performance of the CBB method. We select the value of $\mu$ from $\{0.1, 0.2, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$.

The underlined numerical results in Table 2 indicate that the optimal value of $\mu$ is different for the test problems with different dimensions or condition numbers. The best choice of $\mu$ seems to be chosen in the interval $[0.7, 0.9]$.

To further evaluate the numerical performance of these algorithms, we observe the performance profiles of iteration and CPU time, which are measured by the function $\rho_s(\tau)$ proposed by Dolan and Moré [13]. In Figures 1 and 2, we show the numerical performance of the CBB, CBB(0.8), BB1, CABB, C(0.8) and ABB methods with
Table 2. Average NI of CBB(\(\mu\)) with different \(\mu\).

<table>
<thead>
<tr>
<th>(\mu)</th>
<th>CBB((\mu))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td></td>
</tr>
</tbody>
</table>

respect to the number of iterations by using the average number of iterations reported in Tables 1 and 2. In Figures 3 and 4, we show the numerical performance of the CBB, CBB(0.8), BB1, CABB, C(0.8) and ABB methods with respect to the CPU time. For Figures 3 and 4, we test problem (4.1) in the cases \(n = 1000, 10000\) and \(Cd = 10, 10^2, 10^3, 10^4\) and \(10^5\). And, for each case, we randomly generated 10 test problems.

In Figures 1–4, the vertical axis gives the probability \(P\) of problems for that any given method is within a factor \(\tau\) of the best possible ratio. The left-hand axis of the plot gives the percentage of the test problems for which a method needs the fewest iterations or least CPU time. The right-hand side of the plot gives the percentage of the test problems that are successfully solved by each of the methods. Dolan and Moré [13] pointed out that if the set of test problems is suitably large and representative of problems that are likely to occur in applications, the methods with large probability \(P(\tau)\) are to be preferred.

From Figures 1 and 3, it is clear that the CBB method outperforms the BB1 and CBB(\(\mu\)) methods. Therefore, we conclude that the new step size \(\alpha_k^{\text{CBB}}\) did improve the numerical efficiency of the BB methods.
From Figures 2 and 4, it is easy to see that the numerical performances of the CABB, C(0.8) and ABB methods are much better than the CBB method. In general, compared with the ABB and C(0.8) methods, the CABB method performs better.

4.2. Test by generic nonlinear problems

At the end of this section, we test numerical performance of the BB1, CBB, CBB(0.8), ABB, CABB and C(0.8) methods as they are used to solve the generic nonlinear benchmark test problems (nonquadratic minimization problems) of Andrei [1]. For each problem, the initial point has been fixed as in [1]. The initial step size is generated by the Armijo linear search [17, 18].
and the termination condition for all the algorithms is $\|g_k\|_2 \leq 10^{-6}$. All of the 14 generic nonlinear benchmark test problems of Andrei [1] are solved by the BB1, CBB, CBB(0.8), ABB, CABB and C(0.8) methods. In Table 3, we report the number of iterations, the minimal value of the objective function and the CPU time, respectively.

The numerical results in Table 3 demonstrate that all the six BB methods can successfully solve generic large-scale nonlinear (nonquadratic) optimization problems. Since these algorithms are used to solve problems 5, 6, 18, 29, 41, 49 and 53, it follows from the numerical performance that each method has its respective advantages. The underlined numerical results in Table 3 indicate that the CBB(0.8) and CBB (or C(0.8) and CABB) methods outperform the BB1 (or ABB) method.
Table 3. Numerical results for the test problems in [1].

<table>
<thead>
<tr>
<th>P</th>
<th>Algorithm</th>
<th>n</th>
<th>NI</th>
<th>$f_{\text{min}}$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>BB1</td>
<td>100000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>CBB</td>
<td>100000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>CBB(0.8)</td>
<td>100000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>ABB</td>
<td>100000</td>
<td>109</td>
<td>8.461e-19</td>
<td>3.301</td>
</tr>
<tr>
<td></td>
<td>CABB</td>
<td>100000</td>
<td>101</td>
<td>7.233e-14</td>
<td>3.219</td>
</tr>
<tr>
<td></td>
<td>C(0.8)</td>
<td>100000</td>
<td>81</td>
<td>5.694e-18</td>
<td>2.640</td>
</tr>
<tr>
<td>6</td>
<td>BB1</td>
<td>100000</td>
<td>55</td>
<td>1.855e-15</td>
<td>3.693</td>
</tr>
<tr>
<td></td>
<td>CBB</td>
<td>100000</td>
<td>65</td>
<td>7.722e-16</td>
<td>3.351</td>
</tr>
<tr>
<td></td>
<td>CBB(0.8)</td>
<td>100000</td>
<td>70</td>
<td>7.026e-17</td>
<td>2.846</td>
</tr>
<tr>
<td></td>
<td>ABB</td>
<td>100000</td>
<td>41</td>
<td>1.288e-11</td>
<td>1.802</td>
</tr>
<tr>
<td></td>
<td>CABB</td>
<td>100000</td>
<td>47</td>
<td>2.851e-16</td>
<td>2.024</td>
</tr>
<tr>
<td></td>
<td>C(0.8)</td>
<td>100000</td>
<td>52</td>
<td>2.796e-13</td>
<td>2.214</td>
</tr>
<tr>
<td>18</td>
<td>BB1</td>
<td>100</td>
<td>78</td>
<td>2.357e-15</td>
<td>0.017</td>
</tr>
<tr>
<td></td>
<td>CBB</td>
<td>100</td>
<td>60</td>
<td>8.879e-16</td>
<td>0.009</td>
</tr>
<tr>
<td></td>
<td>CBB(0.8)</td>
<td>100</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>ABB</td>
<td>100</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>CABB</td>
<td>100</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>C(0.8)</td>
<td>100</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>19</td>
<td>BB1</td>
<td>100000</td>
<td>7</td>
<td>2.174e-20</td>
<td>0.671</td>
</tr>
<tr>
<td></td>
<td>CBB</td>
<td>100000</td>
<td>7</td>
<td>1.238e-13</td>
<td>0.495</td>
</tr>
<tr>
<td></td>
<td>CBB(0.8)</td>
<td>100000</td>
<td>7</td>
<td>2.815e-16</td>
<td>0.594</td>
</tr>
<tr>
<td></td>
<td>ABB</td>
<td>100000</td>
<td>7</td>
<td>2.174e-20</td>
<td>0.542</td>
</tr>
<tr>
<td></td>
<td>CABB</td>
<td>100000</td>
<td>7</td>
<td>2.815e-16</td>
<td>0.498</td>
</tr>
<tr>
<td></td>
<td>C(0.8)</td>
<td>100000</td>
<td>7</td>
<td>1.385e-13</td>
<td>0.428</td>
</tr>
<tr>
<td>21</td>
<td>BB1</td>
<td>100000</td>
<td>14</td>
<td>1.429e-17</td>
<td>0.710</td>
</tr>
<tr>
<td></td>
<td>CBB</td>
<td>100000</td>
<td>14</td>
<td>8.728e-18</td>
<td>0.572</td>
</tr>
<tr>
<td></td>
<td>CBB(0.8)</td>
<td>100000</td>
<td>14</td>
<td>1.258e-21</td>
<td>0.696</td>
</tr>
<tr>
<td></td>
<td>ABB</td>
<td>100000</td>
<td>14</td>
<td>1.429e-17</td>
<td>0.641</td>
</tr>
<tr>
<td></td>
<td>CABB</td>
<td>100000</td>
<td>14</td>
<td>1.258e-21</td>
<td>0.578</td>
</tr>
<tr>
<td></td>
<td>C(0.8)</td>
<td>100000</td>
<td>14</td>
<td>8.728e-18</td>
<td>0.530</td>
</tr>
</tbody>
</table>
New adaptive BB step size and applications in optimization

<table>
<thead>
<tr>
<th></th>
<th>BB1</th>
<th>CBB</th>
<th>CBB(0.8)</th>
<th>ABB</th>
<th>CABB</th>
<th>C(0.8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>100k</td>
<td>310</td>
<td>1.821e-8</td>
<td>6.317</td>
<td>100k</td>
<td>811</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1000</td>
<td>210</td>
<td>482.5</td>
<td>0.067</td>
<td>1000</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1000</td>
<td>927</td>
<td>3.610e-08</td>
<td>1.027</td>
<td>1000</td>
<td>870</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>1000</td>
<td>1431</td>
<td>-1</td>
<td>1.335</td>
<td>1000</td>
<td>1305</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>100</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>100</td>
<td>2684</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100k</td>
<td>191</td>
<td>1.391e-8</td>
<td>3.072</td>
<td>100k</td>
<td>278</td>
</tr>
<tr>
<td></td>
<td></td>
<td>250</td>
<td>3.810e-8</td>
<td>4.254</td>
<td>239</td>
<td>2.458e-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>140</td>
<td>482.5</td>
<td>0.052</td>
<td>1000</td>
<td>213</td>
</tr>
<tr>
<td></td>
<td></td>
<td>205</td>
<td>-123.9</td>
<td>0.066</td>
<td>1000</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>700</td>
<td>3.506e-08</td>
<td>0.730</td>
<td>1000</td>
<td>1145</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1277</td>
<td>4.012e-08</td>
<td>1.315</td>
<td>1000</td>
<td>1271</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>991</td>
<td>0</td>
<td>0.681</td>
<td>1000</td>
<td>1146</td>
</tr>
<tr>
<td></td>
<td></td>
<td>991</td>
<td>0</td>
<td>0.681</td>
<td>1000</td>
<td>940</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>1413</td>
<td>0</td>
<td>0.681</td>
<td>1000</td>
<td>1413</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1431</td>
<td>-1</td>
<td>1.335</td>
<td>1000</td>
<td>1305</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>100</td>
<td>1213</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>100</td>
<td>2684</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>100</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>100</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>100</td>
<td>F</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Iterations</td>
<td>CPU Time</td>
<td>Objective Value</td>
<td>Convergence Order</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>----------</td>
<td>-----------------</td>
<td>-------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB1</td>
<td>1000</td>
<td>2108</td>
<td>1.931e-13</td>
<td>0.360</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBB</td>
<td>1000</td>
<td>1729</td>
<td>3.355e-15</td>
<td>0.349</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BB1(0.8)</td>
<td>1000</td>
<td>1546</td>
<td>2.914e-14</td>
<td>0.251</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABB</td>
<td>1000</td>
<td>1140</td>
<td>8.831e-18</td>
<td>0.199</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CABB</td>
<td>1000</td>
<td>1130</td>
<td>3.008e-13</td>
<td>0.221</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C(0.8)</td>
<td>1000</td>
<td>1058</td>
<td>2.544e-15</td>
<td>0.185</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iterations</th>
<th>CPU Time</th>
<th>Objective Value</th>
<th>Convergence Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB1</td>
<td>1000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>CBB</td>
<td>1000</td>
<td>1551</td>
<td>-781.9</td>
<td>0.428</td>
</tr>
<tr>
<td>BB1(0.8)</td>
<td>1000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>ABB</td>
<td>1000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>CABB</td>
<td>1000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C(0.8)</td>
<td>1000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iterations</th>
<th>CPU Time</th>
<th>Objective Value</th>
<th>Convergence Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB1</td>
<td>1000</td>
<td>61</td>
<td>1.070e-14</td>
<td>0.026</td>
</tr>
<tr>
<td>CBB</td>
<td>1000</td>
<td>62</td>
<td>4.818e-15</td>
<td>0.025</td>
</tr>
<tr>
<td>BB1(0.8)</td>
<td>1000</td>
<td>46</td>
<td>4.611e-15</td>
<td>0.021</td>
</tr>
<tr>
<td>ABB</td>
<td>1000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>CABB</td>
<td>1000</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C(0.8)</td>
<td>1000</td>
<td>47</td>
<td>2.729e-14</td>
<td>0.019</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iterations</th>
<th>CPU Time</th>
<th>Objective Value</th>
<th>Convergence Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB1</td>
<td>1000</td>
<td>447</td>
<td>1121e-13</td>
<td>0.111</td>
</tr>
<tr>
<td>CBB</td>
<td>1000</td>
<td>382</td>
<td>1.629e-13</td>
<td>0.099</td>
</tr>
<tr>
<td>BB1(0.8)</td>
<td>1000</td>
<td>519</td>
<td>2.033e-13</td>
<td>0.141</td>
</tr>
<tr>
<td>ABB</td>
<td>1000</td>
<td>327</td>
<td>1.013e-13</td>
<td>0.087</td>
</tr>
<tr>
<td>CABB</td>
<td>1000</td>
<td>321</td>
<td>1.381e-13</td>
<td>0.087</td>
</tr>
<tr>
<td>C(0.8)</td>
<td>1000</td>
<td>365</td>
<td>1.001e-13</td>
<td>0.090</td>
</tr>
</tbody>
</table>

### 5. Conclusions

In this paper, we have proposed a new adaptive and composite BB step size, which is an optimal weighted mean of the classical BB step sizes. Combining the steepest descent direction and this new step size, the so-called CBB algorithm has been developed.

We have established the global convergence of the CBB and CABB methods, as they are applied to solve the quadratic minimization problems. The R-linear convergence theorem has been established for the CBB and CABB methods.

Numerical experiments demonstrate that the developed algorithm in this paper outperforms the similar ones available in the literature, since it is used to solve...
ill-posed large-scale quadratic minimization problems or generic nonlinear test problems. In summary, the new step size is useful to improve the numerical efficiency of algorithms, especially for solving ill-posed or large-scale optimization problems.

Acknowledgements

The authors would like to express their thanks to the anonymous referees for their constructive comments on the paper, which have greatly improved its presentation. This research was supported by the National Natural Science Foundation of China (Grant No. 71671190).

References


