Computational Optimization and Applications (2019) 74:729–746
https://doi.org/10.1007/s10589-019-00125-6

A delayed weighted gradient method for strictly convex quadratic minimization

Harry Fernando Oviedo Leon[1]

Received: 24 September 2018 / Published online: 28 August 2019

Abstract
In this paper is developed an accelerated version of the steepest descent method by a two-step iteration. The new algorithm uses information with delay to define the iterations. Specifically, in the first step, a prediction of the new test point is calculated by using the gradient method with the exact minimal gradient steplength and then, a correction is computed by a weighted sum between the prediction and the predecessor iterate of the current point. A convergence result is provided. In order to compare the efficiency and effectiveness of the proposal, with similar methods existing in the literature, numerical experiments are performed. The numerical comparison of the new algorithm with the classical conjugate gradient method shows that our method is a good alternative to solve large-scale problems.

Keywords Gradient methods · Convex quadratic optimization · Linear system of equations

Mathematics Subject Classification 90C20 · 90C25 · 90C52 · 65F10

1 Introduction

In this work, we consider the strictly convex quadratic minimization problem,

$$\min_{x\in\mathbb{R}^n} f(x) = \frac{1}{2}x^\top Ax - x^\top b \tag{1}$$

where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$ is a symmetric and positive definite (SPD) matrix. It is well known that the solution of (1) is equivalent to solve the following linear system

✉ Harry Fernando Oviedo Leon
harry.oviedo@cimat.mx

[1] Mathematics Research Center, CIMAT A.C., Guanajuato, Mexico

$$Ax = b. \tag{2}$$

Nowadays, solving (2) is an important subject in the numerical methods community, with interest not only on the theoretical properties of the methods, but also on solving instances with large values of n. Direct methods such as *Gaussian elimination* or (sparse) *Cholesky factorization* become impractical due to the amount of time and operations that these procedures must perform to obtain the solution. For this reason, when solving large-scale problems is needed, typically an iterative method is chosen. An iterative method uses an initial point to generate a sequence of improving approximated solutions until convergence.

Some iterative methods have emerged to deal with problem (2). Among the first ones, we can mention the *Richardson method*, *Jacobi method*, *Gauss–Seidel method*, and *Successive over-relaxation method* (for details about these methods see [1]). Another highlighted procedure is the so-called *steepest descent* or also known as the *gradient method*, which was proposed by Cauchy in 1854 [2]. However, it is well known that the steepest descent method has some drawbacks, mainly associated to slow convergence rate. Although all these methods have a well-established convergence analysis, sometimes these procedures have poor practical behaviour. These early methods have been replaced by use of the *conjugate gradient method* (CG) [3] or modified versions of it. So far, the CG method remains the method of choice to solve the problem (1) when n is large.

Some attempts to accelerate gradient-type methods focus on the introduction and development of new step-sizes [4–7]. Other approaches are based on introducing a momentum term into the line-search schemes, such as the CG method and the Nesterov's accelerated gradient method [8]. In 1988 Barzilai and Borwein [4] introduced two new accelerated versions of the gradient method (BB-methods) for solving problem (1), that use a different technique to select the step-size. Specifically, Barzilai and Borwein employ a delayed step-size which greatly accelerate the convergence rate of the method. Raydan [9] developed an ingenious convergence study of the BB-methods. In addition, Dai et al. [10] demonstrated that these methods enjoy R-linear convergence rate. To date, several formulations have been proposed to select step-size, which attempt to incorporate second-order information without increasing significantly the computational cost of the classical steepest descent method. For example, Yuan [5] introduced an ingenious step-size that was built by imposing finite termination for the bidimensional quadratic problem. In addition, Dai et al. [6,7] proposed two step-sizes that alternate the BB-methods and the exact step-size of the classic steepest descent, in the odd and even iterations.

The purpose of this paper is to develop a new accelerated version of the gradient method for (1) or (2). The new method updates each iterate through two steps. In the first step, an auxiliary point y_k is generated by performing a gradient step equipped with the minimal gradient steplength, then in the second step, the new test point is calculate by a weighted sum of y_k and the iterate predecessor to the previous point. Thus, the new method uses delay but in a different way as BB-methods do. In addition, we present several numerical experiments that show the efficiency of the new proposal, comparing it with diverse state-of-the-art methods. Numerical results show evidence that the new method performs almost similar to the

conjugate gradient method, and sometimes converges in fewer iterations than the CG method.

This paper is organized as follows. In the next section we briefly present the gradient method and some of its variants. Section 3 is dedicated to define our new method, and to present the convergence analysis. In Sect. 4, we run some numerical experiments to show the efficiency and effectiveness of our proposal. Finally, in Sect. 5 we offer the general conclusions of this work.

2 Steepest descent method and its variants

The gradient method is an iterative algorithm that updates the iterates using the following scheme, starting from a given point x_0:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \tag{3}$$

where $\alpha_k > 0$ is called the step-size. There are a lot of alternatives for selecting the step-size (see [6,7,11–15]). One of the most popular is to choose α_k as the positive real number that minimizes the objective function along the vector $-\nabla f(x_k)$, i.e.

$$\alpha_k = \arg\min_{\alpha>0} f(x_k - \alpha \nabla f(x_k)). \tag{4}$$

By a simple calculation, it can be verified that the solution to (4) is given by,

$$\alpha_k^{SD} = \frac{\nabla f(x_k)^\top \nabla f(x_k)}{\nabla f(x_k)^\top A \nabla f(x_k)}. \tag{5}$$

A detailed convergence analysis of the gradient method with exact step-size α_k^{SD} is found in [16]. A variant of this method is to take α_k as that positive scalar that minimizes the gradient norm, i.e.

$$\alpha_k = \arg\min_{\alpha>0} ||\nabla f(x_k - \alpha \nabla f(x_k))||_2. \tag{6}$$

It is not difficult to prove that the solution of (6) is

$$\alpha_k^{MG} = \frac{\nabla f(x_k)^\top A \nabla f(x_k)}{\nabla f(x_k)^\top A^2 \nabla f(x_k)}, \tag{7}$$

which is known as *minimal gradient steplength* [17].

It is well known that the gradient method presents poor performance if any of these two values (α_k^{SD} or α_k^{MG}) is used as the step-size. This issue led many researchers to take interest in designing better step-sizes. Two of the most popular step-sizes were introduced in 1988 by Barzilai and Borwein [4]. These authors propose to choose α_k by forcing a quasi-Newton property. More precisely, they proposed to select α_k as a one of the following two alternatives:

$$\alpha_k^{BB1} = \arg\min_{\alpha>0} ||B(\alpha)s_{k-1} - y_{k-1}||_2, \quad \text{or} \quad \alpha_k^{BB2} = \arg\min_{\alpha>0} ||s_{k-1} - B(\alpha)^{-1}y_{k-1}||_2, \tag{8}$$

where $s_{k-1} = x_k - x_{k-1}$, $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$, $B(\alpha) = \frac{1}{\alpha} I_n$ is considered as an approximation of the Hessian of f, and I_n denotes the identity matrix of order n. It is easy to prove that the solutions of the optimization problems (8) are given by:

$$\alpha_k^{BB1} = \frac{s_{k-1}^\top s_{k-1}}{s_{k-1}^\top y_{k-1}}, \quad \text{and} \quad \alpha_k^{BB2} = \frac{s_{k-1}^\top y_{k-1}}{y_{k-1}^\top y_{k-1}}, \tag{9}$$

respectively. Other equivalent expressions for step-sizes α_k^{BB1} and α_k^{BB2} are,

$$\alpha_k^{BB1} = \frac{\nabla f(x_{k-1})^\top \nabla f(x_{k-1})}{\nabla f(x_{k-1})^\top A \nabla f(x_{k-1})}, \quad \text{and} \quad \alpha_k^{BB2} = \frac{\nabla f(x_{k-1})^\top A \nabla f(x_{k-1})}{\nabla f(x_{k-1})^\top A^2 \nabla f(x_{k-1})}. \tag{10}$$

Note that the formulae (10) are identical to the step-sizes α_{k-1}^{SD} and α_{k-1}^{MG} respectively. The retard is the most emblematic feature of the BB-methods, and it is precisely what improves the convergence speed of the steepest descent method.

The seminal paper of Barzilai and Borwein opened a new topic of study and gave rise to many researches about how to choose better step-sizes to accelerate the convergence of the gradient method. Two detailed reviews of these methods are found in [11,18]. Friedlander et al. [19] introduced the gradient method with retards (GMR) to solve the problem (1). The practical version of the GMR family can be written as

$$x_{k+1} = x_k - \alpha_{\nu(k)} \nabla f(x_k), \tag{11}$$

where the step-size $\alpha_{\nu(k)}$ is given by

$$\alpha_{\nu(k)} = \frac{\nabla f(x_{\nu(k)})^\top \nabla f(x_{\nu(k)})}{\nabla f(x_{\nu(k)})^\top A \nabla f(x_{\nu(k)})}, \tag{12}$$

here, $\nu(k)$ is arbitrarily chosen in the set $\{k, k-1, \ldots, \max\{0, k-\bar{m}\}\}$, and $\bar{m}$ is a given positive integer. Note that if $\nu(k) = k$ then GMR reduces to the steepest descent method. Furthermore, when $\nu(k) = k-1$ the GMR becomes the gradient method with BB1 step-size given by (10).

On the other hand, in the literature there are also methods that try to improve the performance of the gradient method by running two steps, where the first step corresponds to the gradient method with some step-size and the second step corrects the point obtained in first step. Lamotte et al. [20] introduced a two step gradient method which is similar to our proposal. Their work uses a smoothing technique, and updates the iterates as follows: first they calculate

$$x_k = x_{k-1} - \alpha_{\nu(k-1)} \nabla f(x_{k-1}), \tag{13}$$

using the step-size with retard according to (12) and then they correct this point by a smoothing technique

$$y_k(\omega_k) = x_k + \omega_k(y_{k-1} - x_k) \quad \text{with} \quad y_0 = x_0, \tag{14}$$

where the parameter ω_k is calculated as

$$\omega_k = \arg\min_{\hat{\omega}\in\mathbb{R}} ||\nabla f(y_k(\hat{\omega}))||_2, \tag{15}$$

stopping the procedure when $||\nabla f(y_k(\omega_k))||_2$ is small enough. Other methods that employ two steps to correct the gradient method are found in [8,21,22]. A similar strategy is presented by Brezinski et al. [23], where the authors combine two methods to obtain a hybrid algorithm that converges faster than each of the methods separately. A drawback of these methods is that they require to perform more floating-point operations per iteration, and typically involve a greater number of inner products than the standard line-search methods such as steepest descent or BB-methods. However, these kind of methods tend to accelerate the convergence rate. In this work, we develop a new two-step method similar to those presented in [20,23], which is introduced in next section.

3 A new two-step method

In this section, we introduce a new two-step method which performs an optimal line-search based on the gradient method, with non-delayed step-size in the first step and using the smoothing technique (15) to impose the delay on the scheme. The resulting algorithm is related to the class of methods based on a three-term recursion formula, and it can also be rewritten in a similar fashion as the conjugate gradient method. In particular, we follow the ideas of the two-step methods, and calculate a prediction of the new test point using the gradient method, equipped with the minimal gradient steplength (7). The idea is to generate an auxiliary sequence $\{y_k\}$ as follows:

$$y_k = x_k - \alpha_k^{MG} \nabla f(x_k), \tag{16}$$

and then, in a second step, to correct this point y_k using a similar smoothing technique described in (14), that is,

$$x_{k+1} = x_{k-1} + \beta_k(\, y_k - x_{k-1}\,), \tag{17}$$

where β_k is calculated in the same way as in (15), i.e.

$$\beta_k = \arg\min_{\beta\in\mathbb{R}} ||\nabla f(x_{k-1} + \beta(\, y_k - x_{k-1}\,))||_2. \tag{18}$$

It is straightforward to prove that the solution of the optimization problem (18) is given by

$$\beta_k = \frac{\nabla f(x_{k-1})^\top (\nabla f(x_{k-1}) - \nabla f(y_k))}{||\nabla f(x_{k-1}) - \nabla f(y_k)||_2^2}. \tag{19}$$

Observe that the new iterate x_{k+1} is calculated by a weighted sum between the point x_{k-1} and y_k. In addition, note that this method also uses delay, because the information about x_{k-1} is merged in the definition of x_{k+1}. However, it does not use delay in the construction of the step-size α_k.

Now we analyze the similarity between the new proposal and the conjugate gradient method [3,16]. The conjugate gradient method, uses the following updating scheme

$$x_{k+1} = x_k + \tau_k p_k, \quad \text{with} \quad \tau_k = -\frac{p_k^\top \nabla f(x_k)}{p_k^\top A p_k}, \tag{20}$$

where the direction search p_{k+1} is updated by the following recurrence,

$$p_{k+1} = -\nabla f(x_{k+1}) + \gamma_{k+1} p_k, \quad \text{with} \quad \gamma_{k+1} = \frac{||\nabla f(x_{k+1})||_2^2}{||\nabla f(x_k)||_2^2}, \tag{21}$$

for more details about this method see [16]. It follows from (20) and (21) that the point x_{k+1} can be rewritten as

$$x_{k+1} = x_k - \tau_k \nabla f(x_k) + \frac{\tau_k \gamma_k}{\tau_{k-1}} s_{k-1}, \tag{22}$$

where $s_{k-1} = x_k - x_{k-1}$. Observe that the update scheme (17) also can be rewritten in a similar form to the conjugate gradient (22). By combining (16) and (17) we have

$$x_{k+1} = x_k - \beta_k \alpha_k^{MG} \nabla f(x_k) + (\beta_k - 1) s_{k-1}. \tag{23}$$

Thus, the new method updates the iterates using a linear combination of $\nabla f(x_k)$ and s_{k-1} analogously to the CG method. Now we state the detailed algorithm corresponding to the equations (16)–(17)–(19).

Algorithm 1 Delayed Weighted Gradient Method (DWGM)

Require: $A \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $x_{-1} = x_0$, $g_0 = \nabla f(x_0)$, $g_{-1} = g_0$, $k = 0$.
1: **while** $||g_k||_2 > \epsilon$ **do**
2: $w_k = A g_k$,
3: $\alpha_k = \frac{g_k^\top w_k}{w_k^\top w_k}$,
4: $y_k = x_k - \alpha_k g_k$,
5: $r_k = g_k - \alpha_k w_k$
6: $\beta_k = \frac{g_{k-1}^\top (g_{k-1} - r_k)}{||g_{k-1} - r_k||_2^2}$,
7: $x_{k+1} = x_{k-1} + \beta_k (y_k - x_{k-1})$
8: $g_{k+1} = g_{k-1} + \beta_k (r_k - g_{k-1})$
9: $k = k + 1$.
10: **end while**

The main computational efforts to be made at each step are the calculation of the matrix–vector product Ag_k, the calculation of four inner products, and the calculation

Table 1 Example 1: Comparison of errors $||\nabla f(x_k)||_2$ for some methods

k	BB1	BB2	CG	DWGM
1	2	2	2	2
2	21.047	21.047	1.8492	1.3578
3	27.138	6.6702	1.6332	1.0441
4	2.9949	1.6973	0.3926	0.3675
5	0.7415	0.9775	3.46e−15	2.57e−15
6	0.5735	0.5618		
7	0.3796	0.4322		
8	0.5505	0.2071		
9	0.6062	1.3160		
10	0.0720	0.0246		
⋮	⋮	⋮		
23	4.36e−08	2.92e−05		
24	2.18e−08	1.92e−07		
25	1.77e−10	9.61e−08		
26		2.21e−10		

of six vector sums. This suggests that the new method involves double number of inner products and sums to which the CG method performs per iteration. The vector sum and inner product can be carried out in a small multiple of n floating-point operations, but the cost of the matrix–vector product depends on the form of matrix A, obviously. An important key property of Algorithm 1 is that this procedure approaches the solution quickly, as we show in Sect. 4.

Example 1 To illustrate the behavior of the DWGM method, we compare it with the BB-methods and the CG method in a toy experiment. Specifically, we consider the problem (1) with

$$A = \begin{pmatrix} 20 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \text{and} \quad b = (1, 1, 1, 1)^\top.$$

This test problem was taken from [4]. All algorithms are started at $x_0 = (0, 0, 0, 0)^\top$, we terminate the process when $||\nabla f(x_k)||_2 < 1\text{e}{-8}$. In Table 1, we report the norm of the residual $\nabla f(x_k)$ for all methods to compare. We can observe that the CG and DWGM methods converge faster than the BB-methods. In addition, we see that the CG and the DWGM perform the same number of iterations and we also note that the sequence of residuals generated by the DWGM method decreases slightly faster than the sequence of residuals generated by the CG method. The results obtained in the numerical experiments (see Sect. 4) suggest that this indeed also occurs in more general linear systems.

Now we present some theoretical results concerning Algorithm 1. The following lemma establishes monotone decrement in the gradient norm for any sequence generated by Algorithm 1.

Lemma 1 *Let $\{x_k\}$ be a sequence generated by Algorithm 1. Then $\{||\nabla f(x_k)||_2\}$ is a monotonically decreasing sequence.*

Proof Let x_k be the point generated by Algorithm 1 in the k-th iteration, and suppose that $||\nabla f(x_k)||_2 \neq 0$. By construction of Algorithm 1 we have

$$\begin{aligned} ||r_k||_2^2 &= ||\nabla f(x_k)||_2^2 - 2\alpha_k \nabla f(x_k)^\top w_k + \alpha_k^2 ||w_k||_2^2 \\ &= ||\nabla f(x_k)||_2^2 - 2\alpha_k \nabla f(x_k)^\top w_k + \alpha_k \left(\frac{\nabla f(x_k)^\top w_k}{||w_k||_2^2} \right) ||w_k||_2^2 \\ &= ||\nabla f(x_k)||_2^2 - \alpha_k \nabla f(x_k)^\top w_k, \end{aligned} \tag{24}$$

where $r_k = \nabla f(y_k)$ and $w_k = A\nabla f(x_k)$. Since α_k is positive and A is a positive definite matrix we obtain $||r_k||_2 < ||\nabla f(x_k)||_2$. On the other hand, by the minimization property of β_k [see Eq. (18)], it is clear that $||\nabla f(x_{k+1})||_2 \leq ||r_k||_2$. Therefore we conclude that $||\nabla f(x_{k+1})||_2 < ||\nabla f(x_k)||_2$, that is, $\{||\nabla f(x_k)||_2\}$ decreases monotonously. □

The following lemma provides us some bounds for the parameter β_k defined in (18)–(19).

Lemma 2 *Let β_k be the parameter defined in (18). Then for each $k \in \mathbb{N}$,*

$$0 \leq \beta_k \leq \frac{1}{2} \left(1 + \frac{||\nabla f(x_{k-1})||_2^2}{||\nabla f(x_{k-1}) - r_k||_2^2} \right), \tag{25}$$

where $r_k = \nabla f(y_k)$.

Proof We first prove the non-negativity of β_k. Rewriting the Eq. (19), we have

$$\nabla f(x_{k-1})^\top r_k = ||\nabla f(x_{k-1})||_2^2 - \beta_k ||\nabla f(x_{k-1}) - r_k||_2^2. \tag{26}$$

It follows from Cauchy–Schwarz inequality and Lemma 1 that

$$\nabla f(x_{k-1})^\top r_k \leq ||\nabla f(x_{k-1})||_2 ||r_k||_2 \leq ||\nabla f(x_{k-1})||_2 ||\nabla f(x_k)||_2 \leq ||\nabla f(x_{k-1})||_2^2. \tag{27}$$

Therefore, in view of (26) and (27), we obtain

$$\beta_k \geq 0, \quad \forall k \in \mathbb{N}. \tag{28}$$

Finally, using the inequality $u^\top v \leq \frac{1}{2}(||u||_2^2 + ||v||_2^2)$ in (19), we arrive at

$$\beta_k \leq \frac{1}{2}\left(1 + \frac{||\nabla f(x_{k-1})||_2^2}{||\nabla f(x_{k-1}) - r_k||_2^2}\right), \quad \forall k \in \mathbb{N}, \tag{29}$$

which proves the lemma. □

It follows from Lemma 1 that the global convergence of Algorithm 1 is guaranteed, which is established in the following theorem.

Theorem 1 *Let $\{x_k\}$ be a sequence generated by Algorithm 1, and $\lambda_1 > \lambda_2 > \cdots > \lambda_n > 0$ be the eigenvalues of the matrix square root of A, i.e., the eigenvalues of $A^{1/2}$. Then the sequence $\{\nabla f(x_k)\}$ converges to zero Q-linearly with convergence factor $\frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}$.*

Proof From Lemma 1, we have that $\{||\nabla f(x_k)||_2\}$ is a monotonically decreasing and bounded below by zero sequence, therefore $\{||\nabla f(x_k)||\}$ is convergent. From the proof of Lemma 1, we know that

$$\begin{aligned} ||r_k||_2^2 &= ||\nabla f(x_k)||_2^2 - \alpha_k^{MG} \nabla f(x_k)^\top w_k \\ &= \left(1 - \frac{\alpha_k^{MG}}{\alpha_k^{SD}}\right) ||\nabla f(x_k)||_2^2. \end{aligned} \tag{30}$$

In view of (5) and (7), we obtain

$$\frac{\alpha_k^{MG}}{\alpha_k^{SD}} = \frac{(v_k^\top v_k)^2}{(v_k^\top A v_k)(v_k^\top A^{-1} v_k)}, \tag{31}$$

where $v_k = A^{1/2}\nabla f(x_k)$. Then applying the Kantorovich inequality to (31) and substituting the result in (30), we arrive at

$$||r_k||_2 \leq \left(\frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}\right) ||\nabla f(x_k)||_2. \tag{32}$$

On the other hand, by the equation (18) we obtain $||\nabla f(x_{k+1})||_2 \leq ||r_k||_2$. Then, by combining this last result with (32) we obtain

$$||\nabla f(x_{k+1})||_2 \leq \left(\frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}\right) ||\nabla f(x_k)||_2.$$

It follows immediately that $\{\nabla f(x_k)\}$ converges to zero Q-linearly with convergence factor $\frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}$ and hence, since A is positive definite, we also conclude that $\{x_k\}$ tends to the unique minimizer of f when k goes to infinity. □

Remark 1 Observe that the convergence rate factor obtained for Algorithm 1 is analogous to the one available for the gradient method with the minimal gradient steplength (7). This means that each step of the process is at least as good as the gradient method with the mentioned step-size.

Table 2 Computational cost per iteration for different methods

Method	Matrix–vector	InnerP	Vector sums	Flops
BB1	1	2	2	$2n^2 + 7n - 2$
BB2	1	3	2	$2n^2 + 9n - 3$
AM	1	2 or 3	2	$2n^2 + 7n - 2$ or $2n^2 + 9n - 3$
LDGM	1	8	4	$2n^2 + 23n - 8$
YuanGM	1	2	2	$2n^2 + 7n - 2$
ABB	1	3	2	$2n^2 + 9n - 3$
CG	1	2	3	$2n^2 + 9n - 2$
DWGM	1	4	6	$2n^2 + 17n - 4$

4 Numerical experiments

In this section, we demonstrate the effectiveness of Algorithm 1 on two different experiments. All methods were implemented in Matlab (R2015a). We compare the new method DGWM with BB1 [4], AM [6], ABB [13], ABB_{min1} with $(m, \tau) = (9, 0.8)$ and ABB_{min2} with $\tau = 0.9$ proposed in [14]. Comparisson with the Yuan gradient method [5], the conjugate gradient method (CG) [3] and also with a novel gradient method that was re- cently published by Liu and Dong [24], which we denote by LDGM, are provided. In the ABB method we set $\kappa = 0.5$. All experiments were performed on a intel(R) CORE(TM) i7-4770, CPU 3.40 GHz with 500 GB HD and 16 GB RAM.

Before presenting the numerical experiments, we compare the computational costs per iteration of several methods. To do so, we review the computational costs of some basic vector operations. Given two n-dimensional vectors $v, w \in \mathbb{R}^n$, a scalar λ and a dense matrix $A \in \mathbb{R}^{n\times n}$, computing an inner product $v^\top w$ needs $2n - 1$ flops (n multiplications and $n - 1$ sums), while computing a matrix–vector multiplication Av requires $2n^2 - n$ flops (this is equivalent to calculate n inner products). In addition, computing a vector sum of the form $v + \lambda w$ needs $2n$ flops while calculating an ordinary vector sum $v + w$ only requires n flops. In Table 2, we list the computational cost of the aforementioned methods giving more details. In this table, the columns "InnerP", "Vector Sums" and "Matrix–vector" give respectively the number of inner products, vector sums and matrix–vector multiplications per iteration performed by each method, and the column "Flops" gives the total number of floating-point operations per iterations for each method.

Table 2 tells us that the methods that have the lowest computational cost per iteration are the BB1 and the YuanGM gradient methods, in counterpart, the procedures that have the highest computational cost are the LDGM and DWGM respectively. However, all methods are of order $O(n^2)$. Note that DWGM requires the greatest number of vector sums per iteration, because it needs two computations of ∇f per iteration, while the others just need one. Also observe that the number of inner products and sums per iteration of the proposed method is double of those of the CG method. Nevertheless, in this work, we present enough numerical evidence to show that the new proposal

Table 3 Experiment 1: The number of iterations and inner products obtained by all methods

n	BB1		AM		LDGM		YuanGM		ABB		CG		DWGM	
	Nitr	Nip	Nitr	Nip	Nitr	Nip	Nitr	Nip	Nitr	Nip	Nitr	Nip	Nitr	Nip
100	99	297	107	374	108	972	191	573	106	424	**64**	192	**64**	320
500	231	693	267	934	287	2575	533	1599	228	912	149	447	**147**	735
1000	299	897	357	1249	321	2889	756	2268	343	1372	212	636	**209**	1045
2500	591	1773	741	2593	539	4851	1218	3654	588	2352	370	1110	**364**	1820
5000	1165	3495	948	3318	894	8046	1690	5070	750	3000	480	1440	**470**	2350
8000	1036	3108	1317	4609	1036	9324	2283	6849	1043	4172	609	1827	**595**	2975
10,000	1408	4224	1449	5071	1122	10,094	2356	7068	1097	4388	681	2043	**665**	3325
12,000	1383	4149	1486	5201	1144	10,296	2550	7650	1209	4836	747	2241	**729**	3645
15,000	1390	4170	1866	6531	1445	13,005	3045	9135	1375	5500	837	2511	**815**	4075
20,000	1676	5028	2272	7952	1521	13,689	3286	9858	1702	6808	968	2904	**941**	4705
50,000	2952	8856	5083	17,790	3177	28,593	5028	15,084	2609	10,436	1538	4614	**1488**	7440

Bold values show the best result obtained in terms of the number of iterations for each instance.

can converge in a smaller number of iterations than the rest of the methods, and in this way, it can save some matrix–vector products, which is the most expensive task to be done for each method.

In the first experiment, we solve the problem (1) with $A = diag(1, 2, \ldots, n)$ and $b = (1, 2, \ldots, n)^\top$, for each $n \in \{100, 500, 1000, 2500, 5000, 8000, 10{,}000, 12{,}000, 15{,}000, 20{,}000, 50{,}000\}$. Observe that n is the condition number of the matrix A. For all the tests, the initial point is the origin, that is, $x_0 = (0, 0, \ldots, 0)^\top$. All methods are stopped if the inequality $||\nabla f(x_k)||_2 \leq 1e{-}8$ is satisfied. Table 3 reports the number of iterations (Nitr) and the number of inner products (counting the inner product associated with the matrix–vector multiplication, keeping in mind that the matrix is diagonal) required by each method (Nip) for this experiment. From Table 3, we see that DGWM method performs much better than the five gradient methods and it converges slightly faster than the CG method. In addition, as the condition number $\kappa(A)$ becomes larger, the DGWM method needs less iterations than the other methods. However, the CG method performs fewer inner products, which indicates that the CG procedure was more efficient than the rest of the methods.

In the second experiment, we compare all the methods on three sets of problems for different dense matrices A. In particular, the methods are compared under three situations of the condition number $\kappa(A)$. Specifically, we consider $A = Q\Sigma Q^\top$ where $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix obtained from the QR factorization of a random matrix $\hat{M} \in \mathbb{R}^{n\times n}$, that is,

$$\hat{M} = \texttt{randn}(n, n) \quad \text{and} \quad [Q, R] = \texttt{qr}(\hat{M}),$$

using Matlab notation, whereas $\Sigma \in \mathbb{R}^{n\times n}$ is a diagonal matrix which is defined, for each set of problems, as described below.

Set 1 Each diagonal entry of Σ is generated as follows: $\sigma_{ii} = 1 + \frac{99(i-1)}{n+1} + 2u_i$, where u_i is uniformly distributed in the interval [0, 1].

Table 4 Experiment 2: Performance of the methods for well conditioned quadratic problems, (*Set 1*)

Method	Nitr	Time	NrmG	Nitr	Time	NrmG
	Test1: n = 500, $\kappa_{av}(A) = 69.3$			Test2: n = 1000, $\kappa_{av}(A) = 71.5$		
BB	114.	0.005	7.38e−9	122.1	0.021	6.82e−9
AM	120.5	0.006	6.79e−9	125.3	0.023	7.38e−9
LDGM	119.2	0.011	7.00e−9	120.4	0.029	7.16e−9
YuanGM	160.6	0.009	8.52e−9	167.7	0.032	8.74e−9
ABB	108.7	0.006	7.24e−9	112.5	0.023	6.95e−9
CG	82.6	**0.005**	8.50e−9	90.7	**0.019**	8.47e−9
DWGM	**81.5**	**0.005**	8.50e−9	**89.3**	0.021	8.51e−9
ABB_{min1}	110.6	0.007	6.54e−9	114.2	0.026	6.82e−9
ABB_{min2}	105.8	0.007	7.39e−9	107.9	0.026	7.97e−9
	Test3: n = 2500, $\kappa_{av}(A) = 75.4$			Test4: n = 5000, $\kappa_{av}(A) = 82.3$		
BB	122.9	0.353	6.81e−9	130	1.415	7.30e−9
AM	131	0.377	7.50e−9	137.5	1.497	6.76e−9
LDGM	122.7	0.362	6.55e−9	132.9	1.44	7.36e−9
YuanGM	174.6	0.501	8.24e−9	190.1	2.038	8.39e−9
ABB	120	0.344	6.00e−9	124.2	1.353	7.44e−9
CG	98	0.286	8.84e−9	104.2	1.134	8.69e−9
DWGM	**96.3**	**0.282**	8.97e−9	**102.1**	**1.105**	8.85e−9
ABB_{min1}	119.4	0.345	6.23e−9	127.1	1.379	6.68e−9
ABB_{min2}	112.3	0.328	7.39e−9	119.7	1.286	6.94e−9

Bold values show the best result obtained in terms of the number of iterations for each instance.

Set 2 The diagonal of Σ is given by $\sigma_{ii} = i + 2u_i$ with u_i uniformly distributed in the interval [0, 1].

Set 3 The diagonal elements σ_{ii} of Σ are given by $\sigma_{ii} = i^{1.5} + u_i$, where u_i are random numbers uniformly distributed in the interval [0, 1].

Additionally, the exact solution of the problem is created as $x^* = \texttt{randn}(n, 1)$ and $b = Ax^*$. As the starting point, we choose $x_0 = (0, 0, \ldots, 0)^\top$. Note that the sets of problems *Set 1* and *Set 2* are well conditioned and moderately ill-conditioned respectively, while the *Set 3* is ill-conditioned. For each set of problem, we consider $n \in \{500, 1000, 2500, 5000\}$ and we use a tolerance of $\epsilon = 1\text{e}{-8}$ and $N = 15000$ as the maximum number of iteration allowed for all methods. For each value of n, we repeat 100 independent runs of each method. The Tables 4, 5 and 6 collect the results in terms of average number of iterations (Nitr), average execution time in seconds (Time), average condition number of A ($\kappa_{av}(A)$) and the average gradient norm (NrmG), calculated according to $NrmG = ||A\hat{x} - b||_2$ where $\hat{x}$ denotes the solution estimated by each algorithm.

In Fig. 1, we illustrate the behavior of our proposed algorithm solving one instance of each set of problem. For all experiments reported in this figure, we use $n = 1000$ and $\epsilon = 1\text{e} - 4$. Figure 1 reports the residual norm for the four most efficient algorithms

Table 5 Experiment 2: Performance of the methods for moderately ill-conditioned quadratic problems, (*Set 2*)

Method	Nitr	Time	NrmG	Nitr	Time	NrmG
	Test5: n = 500, $\kappa_{av}(A) = 282.8$			Test6: n = 1000, $\kappa_{av}(A) = 531.6$		
BB	247.9	0.011	7.27e−9	345.1	0.057	7.11e−9
AM	258.8	0.012	7.45e−9	360.6	0.065	2.08e−9
LDGM	236.6	0.02	7.51e−9	332.3	0.082	7.07e−9
YuanGM	351.4	0.018	8.87e−9	513.4	0.098	8.84e−9
ABB	217	0.011	6.93e−9	295.5	0.056	7.92e−9
CG	132.8	**0.007**	8.43e−9	187.4	**0.038**	9.05e−9
DWGM	**131.1**	0.008	8.51e−9	**184.2**	0.04	9.01e−9
ABB_{min1}	216.9	0.012	6.76e−9	288.2	0.058	7.06e−9
ABB_{min2}	203.4	0.012	6.54e−9	278.9	0.059	6.37e−9
	Test7: n = 2500, $\kappa_{av}(A) = 1285.4$			Test8: n = 5000, $\kappa_{av}(A) = 2870.7$		
BB	561.3	1.604	7.33e−9	817.9	8.772	7.77e−9
AM	572.2	1.636	3.05e−9	922.1	9.873	2.44e−9
LDGM	543.7	1.603	7.00e−9	777.7	8.281	7.72e−9
YuanGM	849.3	2.423	8.94e−9	1265.2	13.208	9.18e−9
ABB	464.6	1.324	8.12e−9	725.6	7.73	7.93e−9
CG	300.5	0.861	9.38e−9	426.1	4.487	9.63e−9
DWGM	**294.5**	**0.851**	9.44e−9	**416**	**4.411**	1.07e−9
ABB_{min1}	447.7	1.275	7.94e−9	665.2	7.127	7.66e−9
ABB_{min2}	441.7	1.268	7.34e−9	639.4	6.748	7.79e−9

Bold values show the best result obtained in terms of the number of iterations for each instance.

on each problem. We can observe that our proposal converges faster than the rest of the methods and also our procedure shows a similar behavior to the CG method.

As shown in Table 4, the DWGM method and the CG method are superior to the other five gradient methods. In addition, all methods obtain a good estimate of the solution with excellent precision and show very competitive results for well-conditioned problems. From Table 5, we can see that the CG and DWGM methods converge in almost half the time and iterations that are necessary for the other methods. We also observe that the DWGM method converges slightly faster than the CG method in average. In addition, in Tables 4 and 5, we observe that although the DWGM method has a higher computational cost per iteration than the CG method, it performs comparably with the CG method in terms of CPU time, when n is large and A is well or moderately well conditioned.

The numerical results related to the set of ill-conditioned *Set 3* problems are listed in Table 6. From this table, we see that for this situation of bad conditioning of A, DWGM successfully solves all problems. Furthermore, we note that the gradient methods require ten times the iterations needed by CG and DWGM to satisfy the stopping criterion. In addition, we observe that the CG method and our proposal show similar and competitive results in terms of iterations and CPU time.

Table 6 Experiment 2: Performance of the methods for ill-conditioned quadratic problems, (*Set 3*)

Method	Nitr	Time	NrmG	Nitr	Time	NrmG
	Test9: n = 500, $\kappa_{av}(A) = 7623.6$			Test10: n = 1000, $\kappa_{av}(A) = 2.14e+4$		
BB	1502.9	0.067	8.31e−9	2763.5	0.472	8.49e−9
AM	1717.6	0.073	1.00e−9	3741.5	0.657	1.62e−9
LDGM	1238.5	0.094	8.83e−9	2072.2	0.487	8.50e−9
YuanGM	2092.3	0.099	9.33e−9	3732.5	0.688	9.15e−9
ABB	1126	0.056	7.53e−9	1904.7	0.344	7.45e−9
CG	398.1	**0.02**	8.70e−9	678.7	**0.125**	9.39e−9
DWGM	**395.3**	0.023	1.01e−9	**671.7**	0.135	4.22e−9
ABB_{min1}	1043	0.056	7.89e−9	1663	0.313	8.28e−9
ABB_{min2}	994.1	0.058	7.29e−9	1707.4	0.338	7.27e−9
	Test11: n = 2500, $\kappa_{av}(A) = 8.44e+4$			Test12: n = 5000, $\kappa_{av}(A) = 2.42e+5$		
BB	5851.5	16.498	1.68e−9	1.08e+4	116.378	8.18e−9
AM	1.16e+4	32.494	6.33e−9	27198	293.604	1.01e−9
LDGM	4185.8	12.083	1.50e−9	7165.9	77.176	6.55e−9
YuanGM	7793.3	21.633	1.75e−9	1.28e+4	135.717	7.90e−9
ABB	4399.4	12.123	1.47e−9	8281	89.465	7.56e−9
CG	1361.2	**3.736**	1.34e−9	2304.4	24.485	5.16e−9
DWGM	**1342.9**	**3.736**	7.24e−9	**2266.1**	**24.344**	6.41e−9
ABB_{min1}	3440.3	9.47	1.46e−9	5801.7	62.79	6.97e−9
ABB_{min2}	3599.3	9.984	1.39e−9	6592.4	70.56	6.93e−9

Bold values show the best result obtained in terms of the number of iterations for each instance.

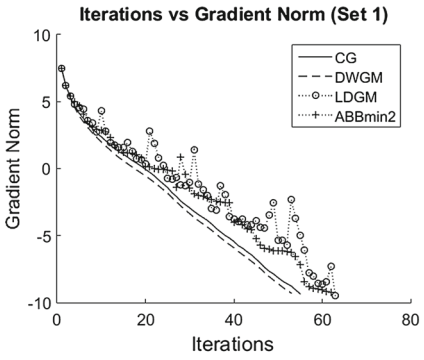
The previous considerations indicate that the proposed method outperforms, in efficiency, a wide collection of state-of-the-art gradient methods and is competitive with the CG method. However, when matrix A is ill-conditioned, the performance and efficiency of our algorithm goes down. Thus, it is necessary to use preconditioning techniques to improve the performance of the new proposal. In addition, we observe that the main competitor of DWGM is the CG method.

In order to perform a better comparison of our Algorithm 1 with the CG method, we carry out a third experiment to test both procedures on large-scale and ill-conditioned synthetic problems. For this numerical experiment, we consider $A = QDQ^{\top} \in \mathbb{R}^{n\times n}$ where

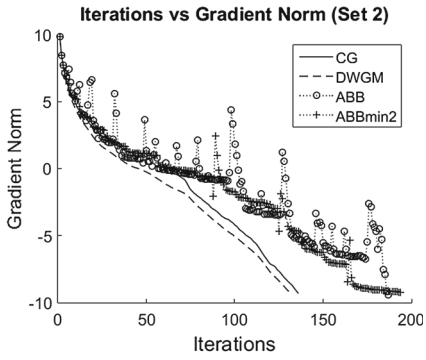
$$Q = (I - 2v_3v_3^{\top})(I - 2v_2v_2^{\top})(I - 2v_1v_1^{\top}),$$

and where I is the identity matrix, v_1, v_2, and v_3 are three unit vectors generated by the following Matlab commands, $v_i = \texttt{rand}(n, 1)$, $v_i = v_i/norm(v_i)$, for $i = 1, 2, 3$, and $D = diag(d_1, d_2, \ldots, d_n)$ is a diagonal matrix whose i-th component is defined by

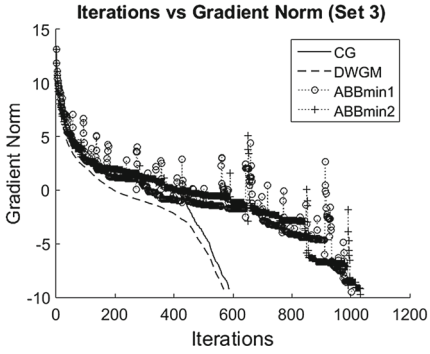
$$log(d_i) = \frac{i-1}{n-1} ncond, \quad i = 1, 2, \ldots, n.$$



(a) Set 1: Iterations vs Gradient Norm



(b) Set 2: Iterations vs Gradient Norm



(c) Set 3: Iterations vs Gradient Norm

Fig. 1 Behavior of the four best algorithms for $n = 1000$, $\epsilon = 1\text{e}{-4}$ and for each set of problem. The y-axis is on a logarithmic scale

This test problem was taken from [25].

Note that the parameter *ncond* is related to the condition number of A. The exact solution of the problem was generated by $x^* = 2 * \texttt{rand}(n, 1) - \texttt{ones}(n, 1)$ using Matlab notation and the starting point was $x_0 = (0, 0, \ldots, 0)^\top \in \mathbb{R}^n$. For this experiment, we use $\epsilon = 1\text{e}{-6}$ as the tolerance for the gradient norm stopping criterion and also we use $N = 50{,}000$ as the maximum number of iterations allowed for each algorithm. Additionally, we consider $n \in \{1000, 5000, 10{,}000, 15{,}000, 20{,}000\}$ and $ncond \in \{5, 10, 15\}$. For each pair $(n, ncond)$, we generate ten independent runs and we report the mean of the gradient norm (NrmG), the mean of the iterations (Nitr), as well as the average CPU-time in seconds (Time) obtained for each algorithm.

The detailed summary of computational results associated with the third experiment are reported in Table 7. In this table, we can see that both methods work well on ill-conditioned problems. In addition, we observe that our proposal converges slightly faster than the CG method in terms of iterations and CPU-time. Note that in the most difficult case ($n = 20{,}000$ and $ncond = 15$), the DWGM method needs, on average, 7 min less than the time required by the CG method to reach the optimum. This

Table 7 Experiment 3: Numerical results for random large-scale ill-conditioned quadratic test problems

n	ncond	CG			DWGM		
		Nitr	Time	NrmG	Nitr	Time	NrmG
1000	5	113	**0.02**	9.26e−7	**109**	**0.02**	9.43e−7
	10	1283	**0.22**	9.23e−7	**1192**	**0.22**	9.87e−7
	15	13,628	2.36	8.79e−7	**12466**	**2.34**	9.99e−7
5000	5	119	1.25	9.45e−7	**116**	**1.21**	9.01e−7
	10	1490	15.53	9.86e−7	**1361**	**14.22**	9.91e−7
	15	18,280	193.94	9.60e−7	**15936**	**170.17**	9.99e−7
10,000	5	122	5.11	8.87e−7	**118**	**4.89**	8.79e−7
	10	1532	63.35	9.79e−7	**1400**	**57.94**	9.92e−7
	15	19,182	798.52	9.61e−7	**16640**	**695.37**	9.99e−7
15,000	5	123	11.95	8.91e−7	**119**	**11.32**	9.11e−7
	10	1553	142.82	9.85e−7	**1420**	**130.24**	9.94e−7
	15	19,566	1799.7	9.66e−7	**16950**	**1558.4**	9.99e−7
20,000	5	124	21.25	8.74e−7	**120**	**20.12**	8.93e−7
	10	1566	259.4	9.91e−7	**1432**	**235.73**	9.92e−7
	15	19,789	3234.5	9.85e−7	**17138**	**2804.7**	9.99e−7

Bold values show the best result obtained in terms of the number of iterations for each instance.

numerical results show that the proposed method is competitive in terms of CPU-time and iterations with the conjugate gradient method and in some cases the DWGM method can outperform the CG method.

5 Concluding remarks

Since the classical steepest descent method was proposed by Cauchy [2], different techniques have been used to design accelerated versions of this method. Many of these techniques focus on the design of better step-sizes than the one proposed by Cauchy, while others are based on the principle of correcting the Cauchy's point by introducing a second step on each iteration. In this work, a new accelerated variant of the gradient method is introduced, which takes two optimal steps to obtain the new test point. The global convergence of DWGM is established and it is also proved that this method enjoys at least a Q-linear convergence rate. Numerical results show that DWGM is a very efficient alternative to deal with large-scale strictly convex quadratic minimization problems.

From the experiments performed, it is observed that DWGM is significantly better than several gradient methods existing in the literature. In addition, the proposed DWGM performs quite similar to the well-known conjugate gradient method in terms of iterations and CPU time. Nevertheless, the new proposal needs to carry out more floating-point operations per iteration.

On the other hand, the method presented in this paper is only designed to solve convex quadratic problems. A possible extension of this method consists in combining our proposal with the trust-region strategy, similar to the Steihaug's method [26], which can lead to an efficient new trust-region method to solve large-scale unconstrained minimization problems.

Acknowledgements I would like to thank Dr. Marcos Raydan for your helpful comments and suggestions on this work, and also for sending me pertinent information. The author also would like to thank Dr. Hugo Lara and two anonymous referees for their useful suggestions and comments.

References

1. Kincaid, D., Kincaid, D.R., Cheney, E.W.: Numerical analysis: mathematics of scientific computing, vol. 2. American Mathematical Society (2009)
2. Cauchy, A.: Méthode générale pour la résolution des systemes d'équations simultanées. Comptes Rendus Sci. Paris **25**(1847), 536–538 (1847)
3. Hestenes, M.R., Stiefel, E. (eds.): Methods of Conjugate Gradients for Solving Linear Systems, vol. 49. NBS, Washington (1952)
4. Barzilai, J., Borwein, J.M.: Two-point step size gradient methods. IMA J. Numer. Anal. **8**(1), 141–148 (1988)
5. Yuan, Y.: A new stepsize for the steepest descent method. J. Comput. Math. **24**(2), 149–156 (2006)
6. Dai, Y.-H., Yuan, Y.-X.: Alternate minimization gradient method. IMA J. Numer. Anal. **23**(3), 377–393 (2003)
7. Dai, Y.-H.: Alternate step gradient method. Optimization **52**(4–5), 395–415 (2003)
8. Nesterov, Y.E.: One class of methods of unconditional minimization of a convex function, having a high rate of convergence. USSR Comput. Math. Math. Phys. **24**(4), 80–82 (1984)
9. Raydan, M.: On the Barzilai and Borwein choice of steplength for the gradient method. IMA J. Numer. Anal. **13**(3), 321–326 (1993)
10. Dai, Y.-H., Liao, L.-Z.: R-linear convergence of the Barzilai and Borwein gradient method. IMA J. Numer. Anal. **22**(1), 1–10 (2002)
11. Di Serafino, D., Ruggiero, V., Toraldo, G., Zanni, L.: On the steplength selection in gradient methods for unconstrained optimization. Appl. Math. Comput. **318**, 176–195 (2018)
12. Dai, Y.-H., Fletcher, R.: On the asymptotic behaviour of some new gradient methods. Math. Program. **103**(3), 541–559 (2005)
13. Zhou, B., Gao, L., Dai, Y.-H.: Gradient methods with adaptive step-sizes. Comput. Optim. Appl. **35**(1), 69–86 (2006)
14. Frassoldati, G., Zanni, L., Zanghirati, G., et al.: New adaptive stepsize selections in gradient methods. J. Ind. Manag. Optim. **4**(2), 299–312 (2008)
15. De Asmundis, R., di Serafino, D., Riccio, F., Toraldo, G.: On spectral properties of steepest descent methods. IMA J. Numer. Anal. **33**(4), 1416–1435 (2013)
16. Luenberger, D.G., Ye, Y., et al.: Linear and Nonlinear Programming, vol. 2. Springer, New York (1984)
17. De Asmundis, R., Di Serafino, D., Hager, W.W., Toraldo, G., Zhang, H.: An efficient gradient method using the Yuan steplength. Comput. Optim. Appl. **59**(3), 541–563 (2014)
18. Birgin, E.G., Martínez, J.M., Raydan, M., et al.: Spectral projected gradient methods: review and perspectives. J. Stat. Softw. **60**(3), 1–21 (2014)
19. Friedlander, A., Martínez, J.M., Molina, B., Raydan, M.: Gradient method with retards and generalizations. SIAM J. Numer. Anal. **36**(1), 275–289 (1999)
20. Lamotte, J.-L., Molina, B., Raydan, M.: Smooth and adaptive gradient method with retards. Math. Comput. Model. **36**(9–10), 1161–1168 (2002)
21. Brezinski, C.: Variations on richardson's method and acceleration. Bull. Belg. Math. Soc. Simon Stevin **3**(5), 33–44 (1996)
22. Brezinski, C.: Multiparameter descent methods. Linear Algebra Appl. **296**(1–3), 113–141 (1999)
23. Brezinski, C., Redivo-Zaglia, M.: Hybrid procedures for solving linear systems. Numer. Math. **67**(1), 1–19 (1994)

24. Liu, Z., Liu, H., Dong, X.: An efficient gradient method with approximate optimal stepsize for the strictly convex quadratic minimization problem. Optimization **67**(3), 427–440 (2018)
25. Dai, Y.-H., Fletcher, R.: Projected Barzilai–Borwein methods for large-scale box-constrained quadratic programming. Numer. Math. **100**(1), 21–47 (2005)
26. Steihaug, T.: The conjugate gradient method and trust regions in large scale optimization. SIAM J. Numer. Anal. **20**(3), 626–637 (1983)