A Hybrid Gradient Method for Strictly Convex Quadratic Programming

Harry Oviedo[a], Oscar Dalmau[b] and Rafael Herrera[c]

[a,b,c]Mathematics Research Center, CIMAT A.C. Guanajuato, Mexico

ARTICLE HISTORY
Compiled December 31, 2020

ABSTRACT
In this paper, we present a reliable hybrid algorithm for solving convex quadratic minimization problems. At the k–th iteration, two points are computed: first, an auxiliary point $\dot{x}_k$ is generated by performing a gradient step using an optimal steplength, and secondly, the next iterate x_{k+1} is obtained by means of weighted sum of $\dot{x}_k$ with the penultimate iterate x_{k-1}. The coefficient of the linear combination is computed by minimizing the residual norm along the line determined by the previous points. In particular, we adopt an optimal, non–delayed steplength in the first step and then use a smoothing technique to impose a delay on the scheme. Under a modest assumption, we show that our algorithm is Q–linearly convergent to the unique solution of the problem. Finally, we report numerical experiments on strictly convex quadratic problems, showing that the proposed method is competitive in terms of CPU–time and iterations with the conjugate gradient method.

KEYWORDS
Gradient methods, convex quadratic optimization and linear system of equations.

1. Introduction

The aim of this work is to propose and analyze a new iterative method for solving quadratic minimization problems

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

where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$ is a is symmetric positive definite (SPD) matrix. This problem has been the subject of intensive research since it provides a simple framework for investigating the behavior of gradient methods and because it has many applications, especially with the addition of box–constraints. This kind of problems arises in many applications, e.g., in compressed sensing and image processing [1, 2], machine learning and data mining [3], as well as many others [4, 5].

The classical version of the gradient method was introduced by Cauchy [6] in 1847, which is known as the steepest descent method. The steepest descent algorithm is an intuitive and memoryless method. It is well known that this method presents several

Emails: harry.oviedo@cimat.mx, dalmau@cimat.mx and rherrera@cimat.mx

weaknesses suffering from a slow convergence rate and a zig–zag behavior throughout the iterations. Many techniques have been developed with the purpose of accelerating its convergence. At first, it was believed that the slow convergence was due to the search direction used by the method. After the works of Yudin and Nemirowsky [7], Nesterov [8] and Barzilai and Borwein [9] a renewed interest arose in studying this method. Nowadays, it has become very useful again, due to the complexity bounds and accelerated versions.

The work presented by Barzilai and Borwein proved how an appropriate selection of the step-size parameter can considerably accelerate Cauchy's method. Since their seminal paper appeared in [9], different step–sizes have been designed in order to improve the efficiency of the gradient method, while maintaining its simplicity and low memory demand, (see [10–12] for more details). On the other hand, accelerated versions of the gradient method can be obtained considering a different paradigm from those based on the design of better step sizes. In [13], Brezinski introduced hybrid methods as a proposal to accelerate the convergence of different iterative methods for solving systems of linear equations. These methods have also been extended to solve systems of nonlinear equations [14]. This kind of strategy is based on the combination of two iterative schemes to obtain a more efficient one. These techniques can significantly increase the convergence speed of fixed–point methods in both the linear and non–linear cases.

In this paper, we introduce a hybrid iterative method to compute the solution of the optimization problem (1). Under a modest assumption, we show that the residual sequence $\{\nabla f(x_k)\}$ converges to zero Q–linearly. In addition, we present numerical experiments to compare the proposal with the conjugate gradient method to demonstrate the effectiveness and efficiency of the new method.

This article is organized as follows. In the next two sections, we review the conjugate gradient method, the classical gradient method and hybrid methods for solving systems of linear equations. In Section 4 we introduce a new hybrid method and give some theoretical properties of the proposal. Some preliminary numerical results are presented in Section 5. Conclusions and discussions are made in the last section.

2. The conjugate gradient method

The conjugate gradient method was originally developed by Hestenes and Stiefel [15], as a direct method for efficiently solving symmetric positive definite linear systems of equations. The CG method is based on the idea of iteratively building an orthogonal base of $\mathbb{R}^n$ according to the inner product $\langle u, v\rangle := u^\top Av$, with the additional property that the gradients sequence $\{g_k\}$ forms an orthogonal set under the usual inner product of $\mathbb{R}^n$, that is, $g_i^\top g_j = 0$, for all $i \neq j$. Specifically, the conjugate gradient method updates the iterates as follows

$$x_{k+1} = x_k + \alpha_k p_k, \tag{2}$$

where $\alpha_k > 0$ is the step–size determined by

$$\alpha_k \equiv \arg\min_{\alpha>0} f(x_k + \alpha p_k) = -\frac{g_k^\top p_k}{p_k^\top A p_k}. \tag{3}$$

The directions sequence $\{p_k\}$ is updated recursively by

$$p_{k+1} = -g_{k+1} + \beta_{k+1} p_k, \tag{4}$$

where $\beta_k \geq 0$ is selected in such a way that $p_{k+1}^\top A p_k = 0$, which only leaves one option for this scalar

$$\beta_{k+1} = \frac{g_{k+1}^\top A p_k}{p_k^\top A p_k}. \tag{5}$$

Following [15, 16], the CG algorithm is now presented.

Algorithm 1 Conjugate Gradient Method (CG)

Require: $A \in \mathbb{R}^{n\times n}$, b, $x_0 \in \mathbb{R}^n$, $g_0 = Ax_0 - b$, $p_0 = -g_0$, $k = 0$.
Ensure: x^* a stationary point.
  while $||g_k||_2 > \epsilon$ **do**
    $w_k = Ap_k$,
    $\alpha_k = \left(||r_k||_2^2\right) / \left(p_k^\top w_k\right)$,
    $x_{k+1} = x_k + \alpha_k p_k$,
    $g_{k+1} = g_k + \alpha_k w_k$,
    $\beta_k = \left(||g_{k+1}||_2^2\right) / \left(||g_k||_2^2\right)$,
    $p_{k+1} = -g_{k+1} + \beta_k p_k$,
    $k = k + 1$.
  end while

This Algorithm corresponds to the practical version of the usual conjugate gradient method presented in Algorithm 5.2 of Nocedal's book [16], which uses some important equivalences for the step–size and for β_k, in order to save some arithmetic operations per iteration. The CG method has finite termination theoretically. However, in practice this theoretical property may fail due to numerical rounding errors. Additionally, it is well–known that CG is an optimal method which is established in the following theorem.

Theorem 2.1. *In Algorithm 1, for all $k \geq 1$, the iterate x_k is such that $f(x_k)$ is the minimum possible value of $f(\cdot)$ on the set*

$$V_k = \{x \in \mathbb{R}^n : x = x_0 + \sum_{i=1}^{k} c_i(x_i - x_{i-1}),\ \textit{and}\ c_i \in \mathbb{R},\ \textit{for}\ 0 \leq i \leq k\}.$$

Theorem 2.1 is a theoretical justification of the nice numerical performance that the CG method typically shows in practice.

3. Steepest descent method and the hybrid methods

The steepest descent algorithm computes the new point x_{k+1} using the following recursive formula, starting from a given vector x_0:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \tag{6}$$

where $\alpha_k > 0$ is called the step–size. Different step-sizes correspond to different gradient methods and appropriate selection of this parameter can lead to accelerated versions. A brief review and comparison among different step–sizes is found in [10]. The original version of the gradient method was introduced by Cauchy [6] in 1847. He studied the problem (1), considering in each iteration the "Cauchy step"

$$\alpha_k^{SD} = \arg\min_{\alpha>0} f(x_k - \alpha \nabla f(x_k)) = \frac{\nabla f(x_k)^\top \nabla f(x_k)}{\nabla f(x_k)^\top A \nabla f(x_k)}. \tag{7}$$

A rigorous study of convergence of this method is found in [17, 18]. A variant of this method is to choose α_k as follows

$$\alpha_k^{MG} = \arg\min_{\alpha>0} ||\nabla f(x_k - \alpha \nabla f(x_k))||_2 = \frac{\nabla f(x_k)^\top A \nabla f(x_k)}{\nabla f(x_k)^\top A^2 \nabla f(x_k)}, \tag{8}$$

which is known as *minimal gradient steplength* [10].

Unfortunately, despite the intuitive appeal and low memory requirement of the gradient method, with either of the step-sizes (7)–(8), this method can suffer from very slow convergence. For this reason, researchers are interested in accelerating this method. One way to achieve this is through the design of a new and good step-size. Another way is to design a hybrid method. In this work we consider the second approach.

Hybrid methods involve the mixing of the iterates produced by two methods. This class of methods was introduced by Brezinski in [13]. More specifically, starting from two sequences $\{\dot{x}_k\}$ and $\{\ddot{x}_k\}$ generated by two different iterative methods, the hybrid methods construct a new sequence $\{x_k\}$ by using the following over-relaxation scheme

$$x_{k+1} = \omega_k \dot{x}_k + (1 - \omega_k)\ddot{x}_k, \tag{9}$$

where the scalar ω_k is selected as the argument that minimizes the gradient norm at x_{k+1}, that is,

$$\omega_k = \arg\min_{\omega>0} ||\nabla f(\omega \dot{x}_k + (1-\omega)\ddot{x}_k)||_2. \tag{10}$$

For more details about this type of methods see [13, 14].

4. A new hybrid method

In this section, we introduce a new hybrid method to address the numerical solution of problem (1). There are several alternatives for selecting the two iterative methods used

in the general hybrid scheme (9). In fact, in [13] a total of seven cases are listed. In this paper we consider the case that computes $\dot{x}_k$ by some method and takes $\ddot{x}_k = x_{k-1}$. Specifically, we consider the gradient method with a new step-size to obtain $\dot{x}_k$ and continue with the hybrid scheme as explained in Section 3. In particular, starting at an arbitrary point x_0, we compute $\dot{x}_k$ by

$$\dot{x}_k = x_k - \alpha_k \nabla f(x_k), \tag{11}$$

where $\alpha_k > 0$ is defined by

$$\begin{aligned} \alpha_k &= \arg\min_{\alpha>0} (1-\theta) f(x_k - \alpha \nabla f(x_k)) + \theta ||\nabla f(x_k - \alpha \nabla f(x_k))||_2^2, \\ &= \alpha_k^{MG} \left(\frac{(1-\theta)\alpha_k^{SD} + 2\theta}{(1-\theta)\alpha_k^{MG} + 2\theta} \right). \end{aligned} \tag{12}$$

where θ is some constant value in the interval (0,1]. We compute the new test point by the following recurrence

$$x_{k+1} = \omega_k \dot{x}_k + (1-\omega_k) x_{k-1}, \tag{13}$$

where the parameter ω_k is updated according to (10),

$$\omega_k = \frac{\nabla f(x_{k-1})^\top (\nabla f(x_{k-1}) - \nabla f(\dot{x}_k))}{||\nabla f(x_{k-1}) - \nabla f(\dot{x}_k)||_2^2}. \tag{14}$$

Note that from the previous iterate x_k the new test point x_{k+1} is calculated in two steps. First, a standard gradient method is used to obtain a prediction $\dot{x}_k$ of x_{k+1}, and secondly, the new point is taken to be the point that minimizes the residual norm on the line connecting x_{k-1} and $\dot{x}_k$.

From (14), it is guaranteed that $\nabla f(x_{k+1})$ is A–orthogonal to the vector $\dot{x}_k - x_{k-1}$, and

$$||\nabla f(x_{k+1})||_2 \leq \min(\, ||\nabla f(x_{k-1})||_2, ||\nabla f(\dot{x}_k)||_2 \,),$$

which will be important to establish the convergence of the method. Finally, we propose the following algorithm

In Algorithm 2, the computational effort at each iteration consists in the calculation of five inner products, the estimation of six vector sums and the matrix–vector product Ag_k. 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 the matrix A. Compared with the CG method, our HGM method requires the calculation of a larger number of arithmetic operations per iteration, since the CG only needs to compute two inner products, three vector sums and one matrix–vector multiplication, per iteration. However, our procedure can approach the solution quickly, as we shown in the Section 5.

Now we present some theoretical results concerning Algorithm 2. In the rest of this section, we define by $\nu(\theta) = \frac{1-\theta}{2\theta}$ for any $\theta \in (0,1]$, and we assume that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$ are the eigenvalues of the matrix A. The following lemma establishes the

Algorithm 2 Hybrid Gradient Method (HGM)

Require: $A \in \mathbb{R}^{n\times n}$, $b, x_0 \in \mathbb{R}^n$, $\theta \in (0,1]$, $x_{-1} = x_0$, $g_0 = \nabla f(x_0)$, $g_{-1} = g_0$, $k = 0$.
Ensure: x^* a stationary point.
  while $||g_k||_2 > \epsilon$ **do**
    $w_k = Ag_k$,
    $\dot{x}_k = x_k - \alpha_k g_k$, with α_k as in (12)
    $r_k = g_k - \alpha_k w_k$
    $\omega_k = \frac{g_{k-1}^\top (g_{k-1} - r_k)}{||g_{k-1} - r_k||_2^2}$,
    $x_{k+1} = x_{k-1} + \omega_k(\,\dot{x}_k - x_{k-1}\,)$
    $g_{k+1} = g_{k-1} + \omega_k(\,r_k - g_{k-1}\,)$
    $k = k + 1$,
  end while

monotone decrement in the gradient norm for any sequence generated by Algorithm 2.

Consider the step–size α_k defined in (12). It's not difficult to prove that this step–size is a convex combination of the step-sizes α_k^{MG} and α_k^{SD}, which implies that $\alpha_k \in [\alpha_k^{MG}, \alpha_k^{SD}]$ for all $k \in \mathbb{N}$. Now we proceed to demonstrate the global convergence of our iterative process.

Lemma 4.1 establishes a relation between the gradient norm at x_{k+1} and the gradient norm of the point y_k obtained in the first phases of the Algorithm 2.

Lemma 4.1. *Let $\{x_k\}$ be a sequence generated by Algorithm 2. Then*

$$||\nabla f(x_{k+1})||_2^2 = \left(1 - \frac{(r_k^\top z_k)^2}{||r_k||_2^2 ||z_k||_2^2}\right) ||r_k||_2^2, \tag{15}$$

where $z_k := \nabla f(x_{k-1}) - r_k$.

Proof. From the step 7 of Algorithm 2 we have

$$\nabla f(x_{k+1}) = r_k + (1 - \omega_k) z_k. \tag{16}$$

This implies that

$$||\nabla f(x_{k+1})||_2^2 = ||r_k||_2^2 + 2(1-\omega_k) r_k^\top z_k + (1-\omega_k)^2 ||z_k||_2^2. \tag{17}$$

Using (14) we obtain

$$(1 - \omega_k) = 1 - \frac{\nabla f(x_{k-1})^\top z_k}{||z_k||_2^2} = -\frac{r_k^\top z_k}{||z_k||_2^2},$$

and substituting this result in (17) we arrive at

$$||\nabla f(x_{k+1})||_2^2 = \left(1 - \frac{(r_k^\top z_k)^2}{||r_k||_2^2 ||z_k||_2^2}\right) ||r_k||_2^2,$$

which completes the proof. □

Lemma 4.2. *Let $\{x_k\}$ be a sequence generated by Algorithm 2. If $\lambda_n \geq \nu(\theta)$ then $\{||\nabla f(x_k)||_2\}$ and $\{||r_k||_2\}$ are monotonic decreasing sequences. Moreover, both sequences $\{||\nabla f(x_k)||_2\}$ and $\{||r_k||_2\}$ converge to zero.*

Proof. Let x_k be the point generated by Algorithm 2 and suppose that $||\nabla f(x_k)||_2 \neq 0$. From the construction of the update scheme (13) and the definition of α_k (see (12)) we obtain

$$\begin{aligned} ||r_k||_2^2 &= ||\nabla f(x_k)||_2^2 - 2\alpha_k \nabla f(x_k)^\top w_k + \alpha_k^2 ||w_k||_2^2, \\ &= \left(\frac{\alpha_k}{\alpha_k^{SD}}(\hat{c}_k - 1) + 1\right) ||\nabla f(x_k)||_2^2, \end{aligned} \tag{18}$$

where $w_k = A\nabla f(x_k)$, and $\hat{c}_k = (1-\theta)\frac{\alpha_k^{SD} - \alpha_k^{MG}}{(1-\theta)\alpha_k^{MG} + 2\theta}$.

Since $\alpha_k \in [\alpha_k^{MG}, \alpha_k^{SD}]$ for all $k \in \mathbb{N}$ we have,

$$\begin{aligned} \frac{\alpha_k}{\alpha_k^{SD}} &\geq \frac{\alpha_k^{MG}}{\alpha_k^{SD}}, \\ &= \frac{(v_k^\top v_k)^2}{(v_k^\top A v_k)(v_k^\top A^{-1} v_k)}, \end{aligned} \tag{19}$$

where $v_k = A^{1/2}\nabla f(x_k)$. Applying the Kantorovich inequality to (19) we arrive at,

$$\frac{\alpha_k}{\alpha_k^{SD}} \geq \frac{4\lambda_1 \lambda_n}{(\lambda_1 + \lambda_n)^2}. \tag{20}$$

On the other hand, in view of $\lambda_n \geq \nu(\theta)$ we obtain

$$0 \leq \hat{c}_k = \nu(\theta)\left(\frac{\alpha_k^{SD} - \alpha_k^{MG}}{\nu(\theta)\alpha_k^{MG} + 1}\right) \leq \frac{\lambda_1}{\lambda_1 + \nu(\theta)}, \tag{21}$$

or equivalently,

$$-1 \leq \hat{c}_k - 1 \leq -\frac{\nu(\theta)}{\lambda_1 + \nu(\theta)}. \tag{22}$$

Merging the inequalities (20), (22) and the equation (18), we obtain

$$||r_k||_2^2 \leq \left(\frac{(\lambda_1 + \lambda_n)^2 - 4\lambda_1\lambda_n\omega(\theta)}{(\lambda_1 + \lambda_n)^2}\right) ||\nabla f(x_k)||_2^2, \tag{23}$$

where $\omega(\theta) = \frac{\nu(\theta)}{\lambda_1 + \nu(\theta)} \in (0,1)$. Thus

$$||r_k||_2 < ||\nabla f(x_k)||_2. \tag{24}$$

On the other hand, by the minimization property of ω_k, it is clear that $||\nabla f(x_{k+1})||_2 \leq ||r_k||_2$. Therefore we conclude that $||\nabla f(x_{k+1})||_2 < ||\nabla f(x_k)||_2$ and also that $||r_k||_2 < ||r_{k-1}||_2$, that is, the sequences $\{||r_k||_2\}$ and $\{||\nabla f(x_k)||_2\}$ decrease

monotonically. Furthermore, in view of (18), (23) and the inequality $||\nabla f(x_{k+1})||_2 \leq ||r_k||_2$, we also conclude that the sequences $\{||\nabla f(x_k)||_2\}$ and $\{||r_k||_2\}$ converge to zero. □

It follows from Lemma 4.2 that the global convergence of Algorithm 2 is guaranteed. This is established in the following theorem.

Theorem 4.3. *Let $\{x_k\}$ be a sequence generated by Algorithm 2 and suppose that the smallest eigenvalue of A satisfies $\lambda_n \geq \nu(\theta)$. Then the sequences $\{\nabla f(x_k)\}$ and $\{r_k\}$ converge to zero Q-linearly with convergence factor $\sqrt{\frac{(\lambda_1+\lambda_n)^2-4\lambda_1\lambda_n\omega(\theta)}{(\lambda_1+\lambda_n)^2}}$, where $\omega(\theta) = \frac{\nu(\theta)}{\lambda_1+\nu(\theta)}$.*

Proof. Let us assume that $\lambda_n \geq \frac{1-\theta}{2\theta}$. From Lemma 4.2, we have that $\{||\nabla f(x_k)||_2\}$ and $\{||r_k||_2\}$ are monotone decreasing and bounded below by zero, thus both are convergent. Furthermore, it follows from (18) and (23) that

$$||r_k||_2 \leq \sqrt{\frac{(\lambda_1+\lambda_n)^2 - 4\lambda_1\lambda_n\omega(\theta)}{(\lambda_1+\lambda_n)^2}}||\nabla f(x_k)||_2. \tag{25}$$

On the other hand, by the equation (10), we obtain $||\nabla f(x_{k+1})||_2 \leq ||r_k||_2$, for all k. Then, by combining this last result with (25) we have

$$||\nabla f(x_{k+1})||_2 \leq \sqrt{\frac{(\lambda_1+\lambda_n)^2 - 4\lambda_1\lambda_n\omega(\theta)}{(\lambda_1+\lambda_n)^2}}||\nabla f(x_k)||_2$$

and also

$$||r_k||_2 \leq \sqrt{\frac{(\lambda_1+\lambda_n)^2 - 4\lambda_1\lambda_n\omega(\theta)}{(\lambda_1+\lambda_n)^2}}||\nabla f(x_k)||_2 \leq \sqrt{\frac{(\lambda_1+\lambda_n)^2 - 4\lambda_1\lambda_n\omega(\theta)}{(\lambda_1+\lambda_n)^2}}||r_{k-1}||_2.$$

It follows immediately that $\{\nabla f(x_k)\}$ and $\{r_k\}$ converge to zero Q–linearly with convergence factor $\sqrt{\frac{(\lambda_1+\lambda_n)^2-4\lambda_1\lambda_n\omega(\theta)}{(\lambda_1+\lambda_n)^2}}$ and 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. Note that if we select $\theta = 1$ throughout the iterations, then the hypothesis $\lambda_n \geq \nu(\theta)$ is always verified. So from Theorem 1, we have that Algorithm 1 converges to the unique solution of (1). This particular case ($\theta = 1$) has been studied in [19].

5. Numerical results

In this section, we provide two numerical tests to demonstrate the convergence performance of the proposed HGM algorithm with $\theta = 0.5$ (in order to give the same importance to the objective function and the gradient norm of f in the optimization model (12)). We compare our proposal with the classical conjugate gradient method (CG). We implement all methods in Matlab. All the experiments were carried out on a intel(R) CORE(TM) i7-4770, CPU 3.40 GHz with 500GB HD and 16GB

Table 1. Average number of iterations required by the seven methods with 7 different starting points for the first experiment.

Tol	CondA	CG	HGM
1e-1	10^3	5	**4.9**
	10^4	5.1	**5**
	10^5	**5**	**5**
	10^6	5.2	**4.8**
1e-3	10^3	56.4	**42.1**
	10^4	89.1	**47.8**
	10^5	91.9	**47.1**
	10^6	87.9	**47.6**
1e-6	10^3	161.9	**153.5**
	10^4	407.4	**371.1**
	10^5	950.3	**851.7**
	10^6	2020.9	**1732.2**

RAM. The implementation of our algorithm and the CG method are available in `http://www.optimization-online.org/DB_HTML/2020/02/7633.html`.

5.1. Random Problems

In the first experiment, we consider sparse quadratic test problems which were taken from [20]. Each problem is generated by using the following Matlab commands, $n = 1000$, $A = \texttt{sprandsym}(n, 0.8, 1/condA, 1)$, $x^* = -10 + 20 * \texttt{rand}(n, 1)$ and $b = Ax$, where $condA$ denotes the condition number of the matrix A. The initial point was randomly generated by $x_0 = -10 + 20 * \texttt{rand}(n, 1)$. For this specific experiment we stop the algorithms if the inequality $||\nabla f(x_k)||_2 < \epsilon ||\nabla f(x_0)||_2$ is satisfied where $\epsilon =$ 1e-6, or the number of iterations exceeds $K = 10000$.

Table 1 reports the average number of iterations required by the seven methods with five different starting points for the first set of problems. For each pair of tolerance and condition number, the winner method is marked in bold. As shown in Table 1, our method wins in all cases. In particular, if only an approximation of the solution is required, with low accuracy and the matrix A is ill–conditioned, then HGM obtains the estimate solution in almost half of the iterations of the CG, (see Table 1, with $tol =$ 1e-3 and $condA = 10^4, 10^5, 10^6$). From these results we see that our proposal is a good alternative to find an estimation of the solution of sparse linear system of equations with moderate accuracy.

In the second group of experiments we evaluate the algorithms in solving large scale problems proposed in [21]. In particular, the matrix A in (1) is randomly generated as follows: $A = QDQ^\top \in \mathbb{R}^{n\times n}$ with

$$Q = (I - 2v_1v_1^\top)(I - 2v_2v_2^\top)(I - 2v_3v_3^\top), \tag{26}$$

where I denotes the identity matrix, v_1, v_2, and v_3 are normalized random vectors,

Table 2. Numerical results for random large–scale ill–conditioned quadratic test problems.

n	**ncond**	**CG**			**HGM**		
		Nitr	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
1000	5	113	0.02	9.20e-7	**111**	0.03	9.14e-7
	10	**1285**	0.23	9.09e-7	1303	0.26	9.92e-7
	15	**13648**	2.29	9.21e-7	15167	2.95	9.90e-7
5000	5	119	1.29	9.46e-7	**116**	1.25	8.99e-7
	10	1491	15.75	9.79e-7	**1381**	14.74	9.94e-7
	15	18290	192.57	9.38e-7	**16542**	175.78	9.99e-7
10000	5	122	5.15	8.59e-7	**118**	4.93	8.81e-7
	10	1533	63.67	9.84e-7	**1410**	58.78	9.93e-7
	15	19172	794.23	9.62e-7	**16922**	703.08	9.80e-7
15000	5	123	11.84	8.98e-7	**119**	11.35	9.20e-7
	10	1553	142.94	9.89e-7	**1426**	131.06	9.95e-7
	15	19554	1796.60	9.74e-7	**17139**	1576.60	9.99e-7
20000	5	124	22.33	8.76e-7	**120**	21.31	8.97e-7
	10	1566	261.15	9.88e-7	**1436**	237.87	9.94e-7
	15	19781	3258.00	9.81e-7	**17283**	2848.40	9.82e-7

$D = \text{diag}(d_1, d_2, \ldots, d_n)$ is a diagonal matrix whose i–th component is defined by

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

Here *ncond* is related to the condition number of A. Additionally, the vector $b \in \mathbb{R}^n$ is generated by $b = Ax^*$, where $x^* = 2 * \texttt{rand}(n, 1) - \texttt{ones}(n, 1)$ using Matlab notation and the starting point is the null vector.

For this experiment, we test both procedures on large–scale and ill–conditioned dense problems. We let all algorithms run up to $K = 50000$ iterations and stop them at iteration $k < K$ if $||\nabla f(x_k)||_2 <$ 1e-8. For this end, we consider the experiment given by (26), but varying the value of n in the set $\{1000, 5000, 10000, 15000, 20000\}$ and for three different values of $ncond \in \{5, 10, 15\}$. For each value of *ncond*, we run the algorithms on ten independent problems and we report the mean of the iterations **Nitr**, the average CPU–time in seconds **Time** and the average gradient norm **NrmG**, for all algorithms. The numerical results concerning this test are summarized in Table 2. From Table 2 we can see that our proposal obtains the solution of each linear system of equations faster than the CG method in both CPU–time and number of iterations in almost all the instances considered.

For the third experiment, we compare both methods considering some randomly generated symmetric and positive definite linear system of equations constructed as follows: the matrix $A \in \mathbb{R}^{n\times n}$ is a dense matrix factorized by $A = Q^\top DQ$, where $Q \in \mathbb{R}^{n\times n}$ is a random orthogonal matrix generated by the following Maltab's command $[Q, \sim] = \texttt{qr}(\texttt{randn}(n, n))$ and $D = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix defined by the following two structures:

- **structure 1**: $\lambda_1 = \kappa(A)$, $\lambda_n = 1$ and the rest of $\lambda_j's$ are determined in such a way that λ_i/λ_{i-1} is constant.

Table 3. Numerical results for linear systems with eigenvalues determined by structure 1.

		CG			HGM		
n	$\kappa(A)$	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
	10^3	312.8	0.042	9.46e-9	**309.2**	0.046	9.56e-9
1000	10^4	**950.4**	0.092	9.27e-9	958.6	0.098	9.93e-9
	10^5	**2878.2**	0.272	8.54e-9	2991.2	0.296	9.96e-9
	10^6	**8611.0**	0.808	9.35e-9	9270.2	1.001	9.98e-9
	10^3	326.8	2.589	9.69e-9	**311.8**	2.447	9.63e-9
5000	10^4	1043.4	9.059	9.64e-9	**977.2**	8.203	9.91e-9
	10^5	3322.4	26.988	9.52e-9	**3063.2**	25.214	9.98e-9
	10^6	10538.0	84.362	9.74e-9	**9615.8**	77.599	9.99e-9
	10^3	328.2	10.067	9.73e-9	**311.8**	9.588	9.79e-9
10000	10^4	1054.0	32.355	9.78e-9	**978.4**	29.937	9.91e-9
	10^5	3382.4	104.973	9.91e-9	**3075.0**	95.587	9.95e-9
	10^6	10808.0	351.982	9.85e-9	**9677.8**	316.978	9.99e-9

- **structure 2**: $\lambda_{\min} = 1$, $\lambda_{\max} = \kappa(A)$, $\lambda_i = \lambda_{\min} + (\lambda_{\max}\lambda_{\min})u_i$, where u_i is a random number from a uniform distribution in (0,0.2) for $i = 1, 2, \ldots, n/2$ and in (0.8,1) for $i = n/2 + 1, \ldots, n$.

These particular structures of eigenvalues were taken from [10, 22]. For this experiment, we use ϵ =1e-8 as the tolerance, and consider the following values of $\kappa(A) \in \{10^3, 10^4, 10^5, 10^6\}$ and $n \in \{1000, 5000, 10000\}$. For each structure, and for each pair $(\kappa(A), n)$ in the sets stated above, we generate ten different instances of these problems and report the average values of the number of iterations, execution time (CPU–time) and the gradient norm evaluated in the solution found by the algorithms. Tables 3 and 4 present the results achieved for each structure.

From tables 3 and 4, we can see that our HGM method, on average, solves the problems in a smaller number of iterations and even in CPU–time than the CG method for linear systems of equations, whose eigenvalues of the matrix A have the structure 1. In contrast, the CG method obtains the solution faster than our HGM when the eigenvalues of A follow the structure 2. However, both methods show very similar performance.

5.2. Two point boundary value problems

In the fourth experiment, we compare our proposal with the CG method under tolerance ϵ =1e-4 through a "real" problem [5]. In particular, for this experiment, $A \equiv (a_{i,j}) \in \mathbb{R}^{n\times n}$ is a tridiagonal matrix constructed as follows

$$a_{i,i} = 2/h^2, \quad a_{i,i-1} = -1/h^2 \text{ if } i \neq 1 \quad a_{i,i+1} = -1/h^2 \text{ if } i \neq n,$$

for all $i \in \{1, 2, \ldots, n\}$, where $h = 11/n$ and n varying in $\{100, 500, 750, 1000\}$, and the initial point x_0 is the null vector of $\mathbb{R}^n$ and the vector b is generated by $b = -1 + 2 * \texttt{rand}(n, 1)$ using Matlab notation. This kind of linear system of equations appears frequently in the numerical solution of two–point boundary–value problems. We consider the cases $n = 500, 1000, 2500, 5000$, and set $K = 50000$ as the maximum number of iterations. In Table 5 we present the average number of

Table 4. Numerical results for linear systems with eigenvalues determined by structure 2.

		CG			HGM		
n	$\kappa(A)$	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
1000	10^3	**201.8**	0.025	8.69e-9	204.8	0.028	9.19e-9
	10^4	**306.6**	0.037	9.19e-9	360.4	0.043	9.36e-9
	10^5	**290.4**	0.038	8.21e-9	396.8	0.054	9.57e-9
	10^6	**348.0**	0.050	8.83e-9	516.8	0.085	9.68e-9
5000	10^3	232.8	2.059	3.26e-10	**223.2**	1.968	9.52e-9
	10^4	**548.8**	4.623	9.71e-9	549.0	4.599	9.84e-9
	10^5	**615.2**	4.759	9.41e-9	734.6	5.666	9.77e-9
	10^6	**672.4**	5.146	9.19e-9	913.2	7.004	9.67e-9
10000	10^3	240.8	7.284	9.74e-9	**229.6**	6.833	9.81e-9
	10^4	614.2	18.312	9.63e-9	**584.8**	17.377	9.88e-9
	10^5	**864.8**	25.743	9.34e-9	969.8	28.828	9.84e-9
	10^6	**934.4**	27.940	9.50e-9	1187.8	35.505	9.90e-9

Table 5. Performance of the seven methods solving two point boundary value problems.

Method	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
	n = 500			n = 1000		
CG	**500**	0.02	4.16e-6	**1000**	0.21	1.14e-5
HGM	720	0.05	9.95e-5	1208	0.33	9.92e-5
	n = 2500			n = 5000		
CG	**2500**	7.38	1.73e-5	**5000**	53.29	3.49e-5
HGM	2634	7.70	9.25e-5	**5000**	53.59	8.18e-5

iterations, the average CPU–time in seconds and the average gradient norm required by all methods with different choices of n on ten independent instances. As shown in Table 5, the HGM method obtains a very similar performance to the conjugate gradient method. However, the CG method was the most efficient method in all instances.

5.3. Convex quadratic test function with real data

The fifth experiment set contains 54 strictly convex quadratic test functions using some large–scale sparse matrices $A \in \mathbb{R}^{n\times n}$, with $n \geq 1000$ taken from the UF Sparse Matrix Collection [23][1]. For generating the vector b, we assumed that the solution $x^* = (1, 2, 3, \ldots, n)^\top$, i.e., $b = Ax^*$, and the initial points was fixed at $x_0 = (1, 1, 1, \ldots, 1)^\top$. For this experiment, we let both algorithms run up to $N = 150000$ iterations and stop them at iteration $k < K$ if $||\nabla f(x_k)||_2 < \epsilon||\nabla f(x_0)||_2$, where ϵ=1e-9. The comparison results are summarized in Tables 6 and 7, where **NrmG** denotes the evaluation of stopping criteria at the solution $\hat{x}$ estimated by each algorithm, that is, $\mathbf{NrmG} = ||\nabla f(\hat{x})||_2/||\nabla f(x_0)||_2$.

As shown in tables 6 and 7, HGM is slightly superior to the conjugate gradient method, solving these particular large–scale and sparse linear system of equations. In terms of the number of iterations, our proposal wins in 51.85% of the problems (out of a total of 54 instances), while the CG does it 46.29% of the time,

[1]The SuiteSparse Matrix Collection tool–box is available in `https://sparse.tamu.edu/`

Table 6. Numerical results for linear systems from the UF Sparse Matrix Collection, Part I.

		CG			HGM		
Name	**n**	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
1138_bus	1138	**2412**	0.023	8.04e-10	15419	0.189	9.99e-10
2cubes_sphere	101492	85004	208.921	9.71e-10	**36363**	95.870	9.99e-10
af_0_k101	503625	43174	889.072	9.99e-10	**25863**	610.408	9.99e-10
af_1_k101	503625	44530	879.553	9.99e-10	**25803**	602.747	9.99e-10
af_2_k101	503625	46345	922.680	9.98e-10	**24858**	585.065	9.99e-10
af_3_k101	503625	36574	733.982	9.99e-10	**18860**	442.162	9.99e-10
af_4_k101	503625	47396	943.210	9.98e-10	**21604**	502.144	9.99e-10
af_5_k101	503625	48333	971.107	9.99e-10	**22137**	521.220	9.99e-10
af_shell3	504855	**4161**	84.103	9.33e-11	4323	101.940	9.98e-10
f_shell7	504855	**4170**	86.460	9.76e-10	4324	105.433	9.99e-10
apache1	80800	**2610**	2.225	9.06e-10	11079	11.935	9.99e-10
apache2	715176	**4909**	53.354	9.89e-10	14856	242.159	9.99e-10
bcsstk08	1074	**4765**	0.082	9.40e-10	7672	0.148	9.99e-10
bcsstk09	1083	**283**	0.008	9.15e-10	**283**	0.010	8.62e-10
bcsstk10	1086	3619	0.082	9.94e-10	**3487**	0.086	9.98e-10
bcsstk11	1473	**10833**	0.365	9.99e-10	12444	0.326	9.34e-10
bcsstk13	2003	126707	8.302	8.55e-10	**58419**	4.201	9.99e-10
bcsstk14	1806	12130	0.593	9.45e-10	**6342**	0.362	9.96e-10
bcsstk15	3948	18490	1.646	9.05e-10	**9767**	1.020	9.98e-10
bcsstk16	4884	509	0.114	9.92e-10	**361**	0.094	9.99e-10
bcsstk17	10974	**20991**	9.780	9.96e-10	23639	12.377	9.99e-10
bcsstk18	11948	150000	42.970	1.75e-9	**22993**	8.260	9.99e-10
bcsstk21	3600	9357	0.330	9.90e-10	**8631**	0.392	9.99e-10
bcsstk23	3134	48668	2.584	9.95e-10	**9411**	0.566	9.99e-10
bcsstk24	3562	150000	17.093	2.06e-8	**63009**	9.654	9.99e-10
bcsstk25	15439	150000	53.117	5.88e-8	**106666**	44.919	9.99e-10
bcsstk26	1922	20417	0.562	9.92e-10	**16890**	0.580	9.99e-10

Table 7. Numerical results for linear systems from the UF Sparse Matrix Collection, Part II.

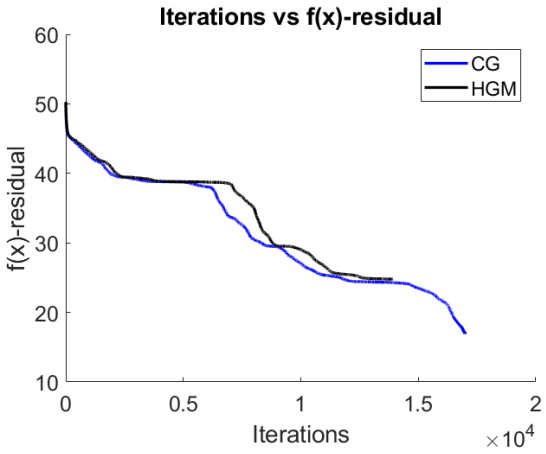
		CG			HGM		
Mame	**n**	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
bcsstk27	1224	1001	0.042	8.91e-10	**996**	0.048	9.91e-10
bcsstk28	4410	**14000**	2.103	8.57e-10	63135	10.969	9.99e-10
bcsstk36	23052	150000	220.831	2.68e-8	**86660**	133.172	9.99e-10
bcsstk38	8032	23382	7.559	9.48e-10	**9115**	3.457	9.99e-10
crystm01	4875	**102**	0.012	7.83e-10	119	0.016	8.17e-10
crystm02	13965	**104**	0.054	8.90e-10	118	0.048	9.24e-10
crystm03	24696	**103**	0.081	9.80e-10	117	0.098	9.04e-10
ex15	6867	2104	0.184	9.75e-10	**1475**	0.159	9.97e-10
fv1	9604	**31**	0.009	8.76e-10	32	0.009	5.56e-10
fv2	9801	**31**	0.009	8.76e-10	32	0.009	5.57e-10
fv3	9801	**208**	0.028	9.69e-10	338	0.055	9.78e-10
Kuu	7102	**741**	0.206	9.63e-10	1102	0.317	9.97e-10
mhd4800b	4800	**52840**	2.003	9.98e-10	150000	7.414	9.83e-2
msc04515	4515	5226	0.394	9.33e-10	**5215**	0.461	9.94e-10
msc23052	23052	150000	230.997	1.37e-8	**62704**	113.566	9.99e-10
Muu	7102	**71**	0.013	9.63e-10	75	0.016	9.26e-10
nasa4704	4704	21621	1.711	9.30e-10	**21617**	2.085	9.99e-10
nasasrb	54870	37643	116.338	9.48e-10	**25444**	81.529	9.99e-10
s1rmq4m1	5489	5615	1.223	8.42e-10	**5412**	1.235	9.99e-10
s2rmq4m1	5489	**17820**	3.468	9.45e-10	29280	6.234	9.99e-10
s1rmt3m1	5489	**6153**	0.956	9.82e-10	6200	1.124	9.98e-10
s2rmt3m1	5489	**24844**	3.820	9.62e-10	27765	4.874	9.99e-10
s3rmq4m1	5489	**39984**	7.811	9.73e-10	150000	31.779	1.94e-9
s3rmt3m1	5489	**63635**	9.447	9.70e-10	86160	14.846	9.99e-10
s3rmt3m3	5357	96268	14.188	9.72e-10	**67990**	12.164	9.99e-10
sts4098	4098	**20690**	1.525	9.74e-10	44790	3.978	9.99e-10
t2dal_e	4257	**5474**	0.137	9.45e-10	5964	0.210	9.99e-10

the remaining percentage is due to a tie between the methods on the instance *bcsstk09*.

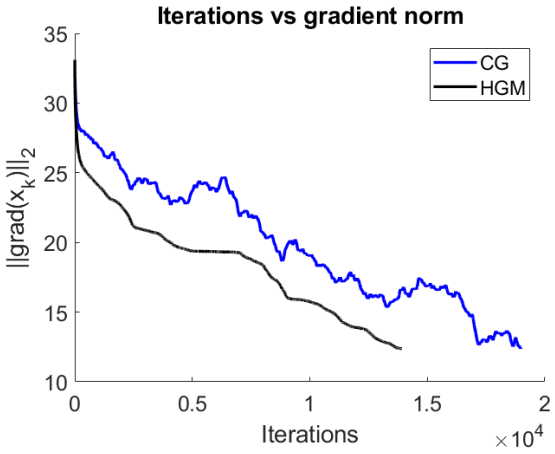
From these percentages, we could conclude that both methods show a very similar performance. However, the HGM method takes a total of 26432.5 iterations, on average, to obtain the solution of the 54 instances, while the CG method takes 34001.4 iterations on average. Furthermore, comparing the average execution times over all 54 instances, HGM method takes 80.261 seconds on average to find the solution, while the CG takes 121.064 seconds on average.

In addition, in Figure 1, we show the behaviour of the HGM and CG method on the positive linear system of equation $Ax = b$, where A is the matrix "Flan_1565" with ID 2544, taken form University of Florida Sparse Matrix Collection [23], which leads to the solution of a large–scale system with $n = 1564794$ and condition number $\kappa(A) = 1.225 \times 10^8$. The vector $b \in \mathbb{R}^n$ and the initial point $x_0 \in \mathbb{R}^n$ were generated as we explained at the beginning of this subsection. In Figure 1 "f–residual" denotes the quantity $f(x_k) + \frac{1}{2}b^\top x^*$, where $x^* = A^{-1}b$ is the solution of the linear system. Note that this quantity can also be computed by $\frac{1}{2}(x_k - x^*)^\top A(x_k - x^*)$. Here it is important to note that the available theory of the CG method states that the CG

algorithm is optimal in terms of this quantity.



(a) Iterations vs f–residual



(b) Iterations vs $||\nabla f(x_k)||_2$

Figure 1. Comparison of the new HGM method with CG using the large–scale ($n = 1564794$) sparse matrix "Flan_1565". The y–axis is on a logarithmic scale.

From Figure 1 (a), we observe that the *f–residual* curve associated to our HGM method is above the corresponding *f–residual* curve of CG procedure, which shows that, although HGM converges in fewer iterations than the CG method, it does not contradict the optimal property of CG method, since it decreases the objective function (and therefore the *f–residual*) more slowly than the CG method. However, in Figure 1 (b), we can see that the HGM algorithm tends to decrease the norm of the gradient of f faster than the CG method, which makes the method stop faster than the CG procedure, because the stopping criteria for both algorithms is just the norm of the gradient. Additionally, we observe that both the *f–residual* curve and the gradient norm associated with the HGM method, show a very smooth decrease, while the gradient norm curve for the CG method shows a non–monotone pattern.

Finally, as a conclusion to this section, we remark that from the numerical experiments carried out, the HGM method shows a tendency to converge in fewer iterations

than the CG method, and this may be due to the monotonic decrease of the gradient norm, which is the usual stop criterion for standard optimization methods. This can explain why on several tests our HGM method converges faster than the CG method. However, this property is observed only experimentally, we do not show a theoretical property that supports this claim, which will be open to investigation in future works.

6. Conclusions

In this paper, we introduce a new hybrid method for unconstrained convex quadratic programming. The proposal can be seen as a two–step method, where in the first step a prediction of the new test point is computed and, in the second step, an acceleration technique is used. This method can be considered as an accelerated version of the classical gradient method, because in the first step the gradient method with a new step–size is used to obtain an estimate of the new iterate. The global convergence and Q–linear convergence of the new procedure are established under a modest assumption. Numerical results show that HGM is very efficient and exhibits a fast performance that competes with the well–known conjugate gradient method, which makes its use interesting to solve large scale linear systems of equations, as well as, to study its possible extensions to the case of general nonlinear unconstrained optimization.

Acknowledgments

This work was supported in part by CONACYT (Mexico), Grants 258033 and 256126. The first author thanks PhD. Marcos Raydan for providing pertinent information and also for their constructive suggestions.

References

[1] Mário AT Figueiredo, Robert D Nowak, and Stephen J Wright. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE Journal of selected topics in signal processing*, 1(4):586–597, 2007.

[2] Hui Huang. Efficient reconstruction of 2D images and 3D surfaces, 2008.

[3] Gaëlle Loosli and Stéphane Canu. Quadratic Programming and Machine Learning-Large Scale Problems and Sparsity. *Optimization in Signal and Image Processing*, pages 111–135, 2009.

[4] Jorge J Moré and Gerardo Toraldo. Algorithms for bound constrained quadratic programming problems. *Numerische Mathematik*, 55(4):377–400, 1989.

[5] Ana Friedlander, José Mario Martínez, Brigida Molina, and Marcus Raydan. Gradient method with retards and generalizations. *SIAM Journal on Numerical Analysis*, 36(1):275–289, 1998.

[6] Augustin Cauchy. Méthode générale pour la résolution des systemes déquations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[7] DB Iudin and Arkadi S Nemirovskii. Informational complexity and efficient methods for solving complex extremal problems. *Matekon*, 13(3):25–45, 1977.

[8] Yu E Nesterov. One class of methods of unconditional minimization of a convex function, having a high rate of convergence. *USSR Computational Mathematics and Mathematical Physics*, 24(4):80–82, 1984.

[9] Jonathan Barzilai and Jonathan M Borwein. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988.

[10] Daniela Di Serafino, Valeria Ruggiero, Gerardo Toraldo, and Luca Zanni. On the steplength selection in gradient methods for unconstrained optimization. *Applied Mathematics and Computation*, 318:176–195, 2018.

[11] Ernesto G Birgin, Jose Mario Martínez, Marcos Raydan, et al. Spectral projected gradient methods: review and perspectives. *J. Stat. Softw*, 60(3):1–21, 2014.

[12] Harry F. Oveido, Oscar S. Dalmau, and Rafael Herrera. Two novel gradient methods with optimal step sizes. *Submitted to Journal of Computational Mathematics*, 2019.

[13] C Brezinski. Hybrid methods for solving systems of equations. *NATO ASI Series C Mathematical and Physical Sciences-Advanced Study Institute*, 508:271–290, 1998.

[14] C Brezinski and J-P Chehab. Nonlinear hybrid procedures and fixed point iterations. *Numerical functional analysis and optimization*, 19(5-6):465–487, 1998.

[15] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

[16] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[17] David G Luenberger, Yinyu Ye, et al. *Linear and nonlinear programming*, volume 2. Springer, 1984.

[18] Jorge Nocedal, Annick Sartenaer, and Ciyou Zhu. On the behavior of the gradient norm in the steepest descent method. *Computational Optimization and Applications*, 22(1):5–35, 2002.

[19] Harry Fernando Oviedo Leon. A delayed weighted gradient method for strictly convex quadratic minimization. *Computational Optimization and Applications*, 74(3):729–746, 2019.

[20] Zexian Liu, Hongwei Liu, and Xiaoliang Dong. An efficient gradient method with approximate optimal stepsize for the strictly convex quadratic minimization problem. *Optimization*, 67(3):427–440, 2018.

[21] Yu-Hong Dai and Roger Fletcher. Projected Barzilai-Borwein methods for large-scale box-constrained quadratic programming. *Numerische Mathematik*, 100(1):21–47, 2005.

[22] Harry F Oveido, Oscar S Dalmau, and Rafael Herrera. Two novel gradient methods with optimal step sizes. *Submitted to Journal of Computational Mathematics*, 2019.

[23] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.