An Accelerated Minimal Gradient Method with Momentum for Convex Quadratic Optimization

HARRY OVIEDO
Fundação Getulio Vargas (FGV/EMAp)
Rio de Janeiro, Brazil.
harry.leon@fgv.br

OSCAR DALMAU
Centro de Investigación en Matemáticas (CIMAT)
Guanajuato, Mexico.
dalmau@cimat.mx

RAFAEL HERRERA
Centro de Investigación en Matemáticas (CIMAT)
Guanajuato, Mexico.
rherrera@cimat.mx

Date: May 5, 2021

Abstract

In this article we address the problem of minimizing a strictly convex quadratic function using a novel iterative method. The new algorithm is based on the well–known Nesterov's accelerated gradient method. At each iteration of our scheme, the new point is computed by performing a line–search scheme using a search direction given by a linear combination of three terms, whose parameters are chosen so that the residual norm is minimized at each step of the process. We establish the linear convergence of the proposed method and show that its convergence rate factor is analogous to the one available for other gradient methods. Finally, we present preliminary numerical results on some sets of synthetic and real strictly convex quadratic problems, showing that the proposed method outperforms in terms of efficiency, a wide collection of state–of–the art gradient methods, and that it is competitive against the conjugate gradient method in terms of CPU time and number of iterations.

Keywords: Gradient methods, Nesterov's accelerated gradient method, conjugate gradient methods.

1 Introduction

In this article, we are interested in designing an efficient algorithm to solve the following quadratic optimization 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 positive definite (SPD) matrix. Problem (1) has been the subject of intensive research since it gives a simple framework to study and analyse properties of iterative

methods, as well as being a problem so hard to solve for large n, in which exact methods such as the Cholesky decomposition and Gaussian elimination are impractical. It is precisely in high dimensions, that iterative methods are required.

One way to find a local minimum of a function is via the gradient descent method. This method starts with an initial point $x_0 \in \mathbb{R}^n$ and performs a line–search along the opposite direction of the gradient, continuing this process until convergence is achieved. The best known version of the gradient method was introduced by Cauchy [4] in 1847, which is also called the steepest descent method. It is well–known that the steepest descent method has some drawbacks, since it can have slow convergence rates as well as oscillatory behaviour, which make it an inefficient method. An option to solve (1) more efficiently is produced by introducing a momentum term in the gradient scheme [13]. This technique can be seen as a line–search iterative scheme, which uses the search direction given by a linear combination of all gradients encountered so far during the search. Such a linear combination is not computed explicitly but calculated incrementally. Therefore, gradient methods with momentum take advantage of the history of the search and incorporate more information than the classical gradient methods.

In the context of convex optimization, the simplest gradient method with momentum is the Heavy–Ball method proposed by Polyak [17]. A small modification of the Heavy–Ball method leads to the well–known Nesterov's accelerated gradient method [13, 14]. For the convex quadratic case (1), the conjugate gradient (CG) method [9], and the preconditioned conjugate gradient method (PCG) are the most efficient. It is well-known [3] that the CG method is equivalent to the following recursive scheme, starting with $x_0 \in \mathbb{R}^n$ and $s_{-1} = 0 \in \mathbb{R}^n$:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k s_{k-1}, \tag{2}$$

where $s_{k-1} = x_k - x_{k-1}$ is referred to as the *momentum term* and the parameters α_k and β_k are determined by

$$(\alpha_k, \beta_k) = \arg \min_{\alpha,\beta \in \mathbb{R}^2} f(x_k - \alpha \nabla f(x_k) + \beta s_{k-1}). \tag{3}$$

Consequently, the conjugate gradient method is an optimal method because the parameters α_k, β_k are selected satisfying an optimality criterion. Other gradient methods with momentum have been developed in [5, 10, 16]

In this paper, we develop an optimal accelerated gradient method with momentum to compute the solution of the optimization problem (1). The new method is based on the Nesterov's accelerated method, and it uses a different optimization model to choose the parameters of the linear combination (2), in a similar way to the CG method (2)-(3). In particular, the proposal generates the new iterate x_{k+1} as the argument that minimizes the residual norm over the linear space $V_k = x_k + span\{\nabla f(x_k), x_k - x_{k-1}, \nabla f(x_k) - \nabla f(x_{k-1})\}$. We show that the residual sequence $\{||\nabla f(x_k)||_2\}$ converges Q–linearly to zero. In addition, we present

some numerical tests on dense matrices randomly generated to compare the performance of the proposal with the CG method. Preliminary results show that the proposed method can converge faster than the CG method both in terms of number of iterations and in CPU time.

This article is organized as follows. In the next section, we review some gradient methods with momentum in the context of the solution of linear systems of equations. In Section 3 we introduce the new method and describe some theoretical properties. Some preliminary results are presented in Section 4. Finally, conclusions are presented in the last section.

2 Gradient methods with momentum and memory

Gradient methods with momentum refer to a particular class of line-search methods for minimizing smooth functions, which involve a linear combination of the negative of the gradient $-g_k$ with a momentum term $x_k - x_{k-1}$. More specifically, these methods construct a sequence $\{x_k\}$ using the following recurrence formula

$$x_{k+1} = x_k - \alpha_k g_k + \beta_k s_{k-1}, \tag{4}$$

where $\alpha_k \in \mathbb{R}$ is called the step–size and $\beta_k \in \mathbb{R}$ is referred to as the *momentum parameter*. Different choices of parameter pairs (α_k, β_k) correspond to different gradient methods with momentum. The simplest gradient method with momentum is the Heavy-Ball method (HBM) of Polyak [17], which considers keeping $\alpha_k = \alpha$ and $\beta_k = \beta$ constant throughout the process. It is well–known [Ang] that the Heavy–Ball Method with parameters

$$\alpha = \frac{4}{(\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}})^2} \quad \text{and} \quad \beta = \frac{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}},$$

where $\lambda_{\min}$ and $\lambda_{\max}$ denote the minimum and maximum eigenvalue of A respectively, converges linearly to the unique solution of problem (1). Another classical iterative algorithm belonging to this type of procedure is the Nesterov's accelerated gradient method [13, 14]. This method updates the iterates using the following two–step scheme: starting with $z_1 = x_0$ and $\gamma_0 = 1$,

$$z_{k+1} = x_k + \beta_k s_{k-1}, \tag{5}$$

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

where $\alpha_k = \frac{1}{L}$ is the inverse of the strong smoothness constant of f and

$$\gamma_k = \frac{1}{2}\left((4\gamma_{k-1}^2 + \gamma_{k-1}^4)^{\frac{1}{2}} - \gamma_{k-1}^2\right), \tag{7}$$

$$\beta_k = \gamma_k(1 - \gamma_{k-1}^{-1}). \tag{8}$$

Note that the iterative scheme (5)–(6) can be rewritten as

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k + \beta_k s_{k-1}) + \beta_k s_{k-1}. \tag{9}$$

Thus, Nesterov's method corresponds to a gradient method with momentum very similar to HBM, except that the gradient is evaluated at an intermediate point.

On the other hand, the most efficient gradient method with momentum to solve (1) is the conjugate gradient method, which updates the iterates by

$$x_{k+1} = x_k + \alpha_k p_k, \tag{10}$$

starting at $p_0 = -g_0$, where α_k is the argument that minimizes f along the line $x_k + \alpha p_k$ and the direction p_k is computed as follows

$$p_{k+1} = -g_{k+1} + \beta_{k+1} p_k, \tag{11}$$

where $\beta_{k+1} = \frac{||g_{k+1}||_2^2}{||g_k||_2^2}$. It follows from (10) and (11) that the point x_{k+1} can be rewritten as

$$x_{k+1} = x_k - \alpha_k g_k + \frac{\alpha_k}{\alpha_{k-1}} \beta_k s_{k-1}. \tag{12}$$

Therefore, the CG method is part of the family of gradient methods with momentum. An important feature of the CG method is that it is equivalent to the iterative scheme (2)–(3), so that it is an optimal method. In the next section, we will introduce a new accelerated gradient method with momentum which is based on this special feature.

3 The algorithm and its convergence analysis

In this section, we introduce a new variant of the accelerated gradient method proposed by Nesterov [13, 14] to address the numerical solution of the problem (1). Motivated by the optimal properties of the CG method and the idea of the Nesterov method, we propose the following iterative scheme to deal with problem (1)

$$x_{k+1} = x_k - \alpha_k \nabla f\left(x_k + \frac{\mu_k}{\alpha_k} s_{k-1}\right) - \beta_k s_{k-1}. \tag{13}$$

Observe that if $\mu_k = \alpha_k \beta_k$ then we recover the iterative scheme proposed by Nesterov (9). In addition, it is not difficult to prove that (13) is equivalent to

$$x_{k+1} = x_k - \alpha_k g_k - \mu_k y_{k-1} - \beta_k s_{k-1}, \tag{14}$$

where $g_k = \nabla f(x_k)$ and $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$. Unlike the pair of parameters (α, β) proposed by Nesterov, we choose the triplet of parameters that minimizes the residual norm at x_k, that is,

$$(\alpha_k, \beta_k, \mu_k) = \arg \min_{(\alpha,\beta,\mu)\in\mathbb{R}^3} \phi(\alpha, \beta, \mu) = ||\nabla f(x_k - \alpha g_k - \mu y_{k-1} - \beta s_{k-1})||_2. \tag{15}$$

From the optimality conditions of the optimization problem (15), we have that the solution of (15) must solve the following linear system of equations,

$$\begin{bmatrix} ||w_k||_2^2 & w_k^\top y_{k-1} & w_k^\top v_{k-1} \\ w_k^\top y_{k-1} & ||y_{k-1}||_2^2 & y_{k-1}^\top v_{k-1} \\ w_k^\top v_{k-1} & y_{k-1}^\top v_{k-1} & ||v_{k-1}||_2^2 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \mu \end{bmatrix} = \begin{bmatrix} g_k^\top w_k \\ g_k^\top y_{k-1} \\ g_k^\top v_{k-1} \end{bmatrix}, \tag{16}$$

where $w_k = Ag_k$ and $v_{k-1} = Ay_{k-1} = w_k - w_{k-1}$. Note that there always exists a triplet of parameters $(\alpha_k, \beta_k, \mu_k)$ such that a decrease of the residual rule is guaranteed. In fact, if we select $(\alpha_k^{MG}, 0, 0)$ then we obtain a decrease in the gradient norm in each iteration, where α_k^{MG} is the so–called *minimal gradient step–length* [7],

$$\alpha_k^{MG} = \arg\min_{\alpha>0} ||\nabla f(x_k - \alpha g_k)||_2 = \frac{g_k^\top A g_k}{g_k^\top A^2 g_k}. \tag{17}$$

This means that our proposal is at least as good as the gradient method equipped with the step–size given by (17), and therefore inherits all its convergence properties.

Proposition 3.1. *Let $\{x_k\}$ be a sequence generated by Algorithm 1. Then for all positive integer k, the following properties are satisfied*

1. $g_{k+1}^\top A g_k = y_{k-1}^\top g_{k+1} = v_{k-1}^\top g_{k+1} = 0$,
2. $y_{k-1}^\top w_k = g_k^\top A g_k$,
3. $y_{k-1}^\top v_{k-1} = g_k^\top A g_k + g_{k-1}^\top A g_{k-1}$.

Proof. The proof of this proposition is a direct consequence of the optimality conditions (16) associated with the optimization problem (15). □ □

Now we present the proposed algorithm and, afterwards, we analyze its global convergence and establish its rate of convergence.

Algorithm 1 Accelerated Minimal Gradient Method (AMGM)

Require: $x_0, b \in \mathbb{R}^n$ and a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$.

Ensure: x^* the unique solution of (1).

Compute $g_0 = Ax_0 - b$, $w_0 = Ag_0$, $\alpha_0 = \frac{g_0^\top w_0}{||w_0||_2^2}$, $s_0 = -\alpha_0 g_0$, $x_1 = x_0 + s_0$, $y_0 = -\alpha_0 w_0$ and $g_1 = g_0 + y_0$, $k = 1$.
while $||g_k||_2 \neq 0$ **do**
  $w_k = Ag_k$.
  Compute $(\alpha_k, \beta_k, \mu_k)$ as the solution of the linear system of equations (16).
  $s_k = -\alpha_k g_k - \mu_k y_{k-1} - \beta_k s_{k-1}$.
  $x_{k+1} = x_k + s_k$.
  $y_k = -\alpha_k w_k - \mu_k(w_k - w_{k-1}) - \beta_k y_{k-1}$.
  $g_{k+1} = g_k + y_k$.
  $k \leftarrow k + 1$.
end while

Similar to the conjugate gradient method, the operation that dominates and requires the greatest computational effort in Algorithm 1 is the matrix–vector product Ag_k. Moreover, Algorithm 1 needs to compute

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

	HBM	**BB1**	**Nesterov**	**AMGM**
Nitr	202	71	300	43
Fval	-1.4878	-1.4878	-1.4878	-1.4878
NrmG	9.71e-9	6.89e-9	9.51e-9	1.69e-9
$\|\mathbf{f}(\hat{\mathbf{x}}) - \mathbf{f}(\mathbf{x}^*)\|$	2.22e-16	2.22e-16	2.22e-16	2.22e-16

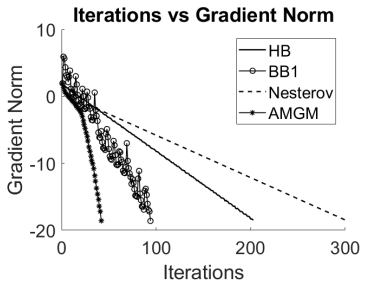
six inner products per iteration, while the CG method only requires two inner products. In addition, the CG method performs three vector sums per iteration, while our Algorithm 1 needs to calculate six vector sums. Clearly, each iteration of the CG method is computationally less expensive than our proposal. Nevertheless, in Section 4, we present enough numerical evidence to show that Algorithm 1 can converge to the unique minimizer of f in fewer iterations and shorter CPU time than the CG method in many situations.

Now we present a toy experiment in order to compare the behavior of Algorithm 1 against the Heavy-Ball method, the Barzilai–Borwein gradient method [2], and the accelerated Nesterov's gradient method. Specifically, we consider the problem (1) where $A = \texttt{diag}(\lambda_{50}, \lambda_{49}, \ldots, \lambda_1) \in \mathbb{R}^{50\times 50}$ with $\lambda_1 > \lambda_2 > \ldots > \lambda_{50}$ randomly generated as follows: $\lambda_1 = 100$, $\lambda_{50} = 1$ and $\lambda_i = 49 * \texttt{rand} + 1$ for all $i \in \{2, 3, \ldots, 49\}$, using Matlab notation. The vectors b and x_0 were taken as $b = (1, 1, \ldots, 1)^\top \in \mathbb{R}^{50}$ and $x_0 = (0, 0, \ldots, 0)^\top \in \mathbb{R}^{50}$ respectively. The tolerance for the gradient norm stopping criterion was $\epsilon =$ 1e-8 and we used $N = 1000$ as the maximum number of iterations for each method. Table 3 presents the numerical results obtained by each method. Figure 1 shows the behaviour of the gradient norm and the absolute error in the objective function for all the methods.

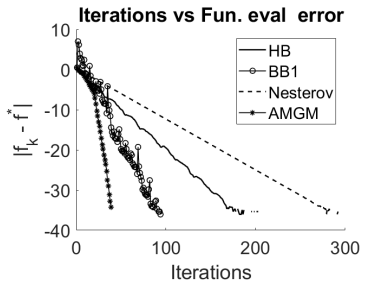
From Table 3, we can see that all the algorithms reach the same objective function value even when the gradient norm criterion is not satisfied for the SD method with the tolerance required. Furthermore, we can observe that the BB and AMGM methods converge faster than the other methods. In addition, we see that our proposal is significantly faster than the other gradient methods with momentum (HBM and Nesterov methods). This suggests that our combination of parameters $(\alpha_k, \beta_k, \mu_k)$ works better than the pair of parameters proposed by Polyak and Nesterov in [13, 17].

Now, we present some theoretical results concerning Algorithm 1. The following theorem establishes the global convergence of our procedure.

Theorem 3.2. *Let $\{x_k\}$ be a sequence generated by Algorithm 1 and x^* the unique minimizer of the convex quadratic function f given in (1). Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$ be the eigenvalues of A. Then either $x_j = x^*$ for some finite j, or the sequence $\{||g_k||_2\}$ converges Q–linearly to zero with convergence factor*



(a) Set 1: Iterations vs Gradient Norm



(b) Set 2: Iterations vs Abs. Err. in f

Figure 1: Behavior of the algorithms for $n = 50$, $\epsilon =$ 1e-8. The y–axis is on a logarithmic scale.

$\frac{\lambda_1-\lambda_n}{\lambda_1+\lambda_n}$.

Proof. By the minimization property (15), it is clear that

$$\phi(\alpha_k, \beta_k, \mu_k)^2 \leq \phi(\alpha_k^{MG}, 0, 0)^2, \tag{18}$$

for all positive integer k, where α_k^{MG} is the minimal gradient step–length. In view of (18), we arrive at

$$\begin{aligned}
||g_{k+1}||_2^2 &\leq ||\nabla f(x_k - \alpha_k^{MG} g_k)||_2^2 \\
&= ||g_k||_2^2 - 2\alpha_k^{MG}(\, g_k^\top A g_k\,) + (\alpha_k^{MG})^2(\, g_k^\top A^2 g_k\,) \\
&= ||g_k||_2^2 - 2\alpha_k^{MG}(\, g_k^\top A g_k\,) + (\alpha_k^{MG})^2 ||w_k||_2^2 \\
&= ||g_k||_2^2 - 2\alpha_k^{MG}(\, g_k^\top A g_k\,) + \alpha_k^{MG}\left(\frac{g_k^\top A g_k}{||w_k||_2^2}\right)||w_k||_2^2 \\
&= ||g_k||_2^2 - \alpha_k^{MG}(\, g_k^\top A g_k\,) \\
&= ||g_k||_2^2 - \alpha_k^{MG}||g_k||_2^2\left(\frac{g_k^\top A g_k}{||g_k||_2^2}\right) \\
&= (1-\hat{c})||g_k||_2^2,
\end{aligned} \tag{19}$$

where $\hat{c} = \frac{(g_k^\top A g_k)^2}{(g_k^\top A^2 g_k)||g_k||_2^2}$. Now, by introducing the vector $p_k = A^{1/2} g_k$, the real number $\hat{c}$ can be rewritten as

$$\hat{c} = \frac{(p_k^\top p_k)^2}{(p_k^\top A p_k)(p_k^\top A^{-1} p_k)}.$$

Then applying the Kantorovich inequality in this last equation we have

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

It follows from (19) and the inequality (20) that

$$||g_{k+1}||_2 \leq \left(\frac{\lambda_1-\lambda_n}{\lambda_1+\lambda_n}\right)||g_k||_2. \tag{21}$$

Since $\frac{\lambda_1-\lambda_n}{\lambda_1+\lambda_n} < 1$, we conclude that the sequence $\{||g_k||_2\}$ converges to zero Q–linearly with convergence factor $\frac{\lambda_1-\lambda_n}{\lambda_1+\lambda_n}$. □ □

Remark. *Note that the convergence rate factor obtained for our Algorithm 1 is similar to the one available for the gradient method proposed by Cauchy (see [4, 12]). Furthermore, observe that if $\lambda_1 = \lambda_n$ then $\{x_k\}$*

Table 2: Iteration counts of diagonal randomly generated test problems.

n	$\kappa(A)$	BB1	AMGM	LDGM	CG	ABB	ABB$_{min1}$	ABB$_{min2}$
Spectral distribution according to Test I								
100	10^3	178	**68**	168	**68**	140	193	176
	10^4	201	**74**	199	**74**	168	273	193
	10^5	436	**87**	404	**87**	272	550	370
1000	10^3	386	**208**	357	211	325	373	356
	10^4	830	**252**	697	254	558	804	689
	10^5	755	**265**	723	266	606	1071	772
10000	10^3	529	**360**	494	375	449	447	443
	10^4	1401	**699**	1185	718	1136	1201	1225
	10^5	2554	**751**	2049	764	1694	2387	1937
Spectral distribution according to Test II								
100	10^3	501	185	472	**182**	454	423	430
	10^4	1635	362	1450	**353**	1542	1298	1356
	10^5	6164	685	4554	**646**	5355	4017	4427
1000	10^3	538	**368**	532	382	488	463	454
	10^4	1836	**1090**	1559	1145	1644	1436	1483
	10^5	7129	**3211**	5576	3398	5833	4569	4767
10000	10^3	538	**406**	533	423	511	476	463
	10^4	1931	**1272**	1635	1351	1727	1515	1548
	10^5	6772	**3979**	5475	4304	6576	4753	4971

converges to x^* in only one iteration.

4 Numerical Experiments

In this section, we test a variety of randomly generated numerical experiments to illustrate the efficiency of our four-term line search algorithm AMGM. We mainly compare Algorithm 1 against the Barzilai and Borwein gradient method (BB1) [2], the ABB method [18] with $\kappa = 0.5$, the ABB$_{min1}$ method with $(m, \tau) = (9, 0.8)$ and the ABB$_{min2}$ method with $\tau = 0.9$ proposed in [8], the CG method [9] and also with the gradient method LDGM proposed by Liu and Dong [11]. All codes are written in MATLAB. All the runs were carried out on a intel(R) CORE(TM) i7–4770, CPU 3.40 GHz with 500GB HD and 16GB RAM.

In the following tables, the column "Nitr" represents the number of iterations, while the columns "NrmG" and "Time" denote the gradient norm, and the CPU time (in seconds) that the algorithms spent to reach the stopping criteria, respectively.

4.1 Diagonal random problems

In this subsection, we present a numerical comparison of our proposal against other state–of–the-art methods available in the literature. In particular, we test the algorithms on problems of the form (1) where $A = \texttt{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix generated randomly. We select a few typical test problems as follows,

- **Test I**: $\lambda_1 = 1$, $\lambda_n = \kappa(A)$ and λ_i is randomly sampled from the uniform probability distribution in $(1, \kappa(A))$, for all $i \in \{2, \ldots, n-1\}$.

- **Test II**: $\lambda_1 = 1, \lambda_n = \kappa(A)$ and $\lambda_i = 10^{p_i}$, where p_i is randomly sampled from the uniform probability distribution in $(0, \log_{10}(\kappa(A)))$, for all $i \in \{2, \ldots, n-1\}$.

This experiment was taken from [8]. In Example 1 and 2, $\kappa(A)$ denotes the condition number of the matrix A. For this examples, we consider $\kappa(A)$ varying $\{10^3, 10^4, 10^5\}$ and with the three different sizes $n = 10^2, 10^3, 10^4$. The entries of the initial points x_0 are randomly sampled in the interval $(-5, 5)$, the stopping condition is $||\nabla f(x_k)||_2 \leq$ 1e-8 and each entry of the vector $b \in \mathbb{R}^n$ is randomly generated from the uniform probability distribution in $(-10, 10)$. For each example and also for each pair $(n, \kappa(A))$ we run ten independent problems and we report the average of each value **Nitr**, **Time** and **NrmG** defined at the beginning of this Section.

The numerical results concerning these two tests are reported in Table 2. We can see that the most efficient methods are the AMGM and the CG. We note that our procedure was as good as the CG method for small problems ($n = 10^2$), being even slightly better than the CG method for the large-scale ones ($n = 10^3$ and $n = 10^4$). Meanwhile, the proposed scheme clearly outperformed all the other gradient–type methods.

4.2 Randomly generated dense systems of linear equations.

In the second set of problems, we test the algorithms on dense systems of linear equations, generated as follows: the matrix $A = V \Lambda V^\top \in \mathbb{R}^{n \times n}$ where $V \in \mathbb{R}^{n \times n}$ is a randomly generated orthogonal matrix, and $\Lambda = \texttt{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix defined by the following three distributions:

- **Distribution I**: $\lambda_1 = \kappa(A)$, $\lambda_n = 1$ and the rest of $\lambda_j' s$ are determined in such a way that $\lambda_i / \lambda_{i-1}$ is constant.
- **Distribution II**: $\lambda_1 = 1$, $\lambda_n = \kappa(A)$, $\lambda_i = \lambda_1 + (\lambda_n - \lambda_1) u_i$, 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 $i = n/2 + 1, \ldots, n$.
- **Distribution III**: $\lambda_1 = 1$, $\lambda_n = \exp(\text{ncond})$, $\lambda_i = \exp\left(\frac{i-1}{n-1}\text{ncond}\right)$, for $i = 2, \ldots, n-1$.

These distributions of eigenvalues were also considered in [10, 15]. For this experiment, we use $\epsilon =$1e-8 as the tolerance. In addition, the vector $b \in \mathbb{R}^n$ and the starting point $x_0 \in \mathbb{R}^n$ were generated by using the

Table 3: Numerical results for linear systems with eigenvalues determined by Distribution I.

		CG			AMGM		
n	$\kappa(A)$	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
1000	1e3	313.4	0.056	9.25e-9	**298.4**	0.073	9.66e-9
	1e4	956.0	0.146	9.48e-9	**894.6**	0.177	9.83e-9
	1e5	2863.4	0.553	9.52e-9	**2639.2**	0.558	9.93e-9
	1e6	8485.4	1.389	8.85e-9	**7718.4**	1.483	9.97e-9
5000	1e3	327.2	2.890	9.70e-9	**310.6**	2.710	9.69e-9
	1e4	1046.2	9.211	9.71e-9	**966.8**	8.924	9.92e-9
	1e5	3327.6	28.672	9.68e-9	**3008.0**	25.109	9.97e-9
	1e6	10531.0	87.730	9.37e-9	**9327.2**	78.225	9.99e-9
10000	1e3	328.4	10.525	9.66e-9	**311.6**	9.973	9.69e-9
	1e4	1053.4	33.430	9.85e-9	**973.4**	30.958	9.89e-9
	1e5	3383.4	107.874	9.80e-9	**3046.0**	97.383	9.96e-9
	1e6	10808.0	347.995	9.83e-9	**9514.2**	309.682	9.99e-9

Table 4: Numerical results for linear systems with eigenvalues determined by Distribution II.

		CG			AMGM		
n	$\kappa(A)$	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
1000	1e3	192.2	0.023	9.00e-9	**186.0**	0.035	9.36e-9
	1e4	270.6	0.037	7.92e-9	**263.4**	0.048	8.70e-9
	1e5	292.8	0.061	7.85e-9	**288.6**	0.067	8.82e-9
	1e6	372.4	0.056	7.61e-9	**370.2**	0.074	8.46e-9
5000	1e3	240.4	2.141	8.68e-9	**229.2**	1.976	9.60e-9
	1e4	517.0	4.375	9.13e-9	**499.8**	4.195	9.71e-9
	1e5	618.8	5.145	9.52e-9	**606.0**	5.153	9.60e-9
	1e6	706.6	5.952	8.73e-9	**695.8**	5.965	9.59e-9
10000	1e3	241.2	7.904	9.35e-9	**230.2**	7.460	9.76e-9
	1e4	580.2	18.935	9.63e-9	**553.6**	18.619	9.72e-9
	1e5	844.4	27.933	9.36e-9	**826.0**	27.424	9.74e-9
	1e6	960.6	32.048	9.57e-9	**944.4**	31.427	9.79e-9

Table 5: Numerical results for linear systems with eigenvalues determined by Distribution III.

		CG			AMGM		
n	**ncond**	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
	5	122.6	0.019	8.87e-9	**118.8**	0.032	9.10e-9
1000	10	1374.2	0.208	9.40e-9	**1279.8**	0.222	9.89e-9
	15	14505.0	1.663	8.98e-9	**13068.0**	1.929	9.99e-9
	5	124.3	1.063	9.34e-9	**121.0**	1.029	8.99e-9
5000	10	1549.4	13.024	9.81e-9	**1423.6**	12.013	9.93e-9
	15	18928.0	159.825	9.71e-9	**16552.0**	140.627	9.99e-9
	5	125.0	4.116	8.93e-9	**121.0**	3.990	9.21e-9
10000	10	1571.8	56.862	9.86e-9	**1440.0**	52.374	9.94e-9
	15	19583.0	603.723	9.29e-9	**17025.0**	522.104	9.99e-9

following Matlab commands

$$y = \texttt{randn}(n, 1), \quad b = A * y, \quad x_0 = \texttt{zeros}(n, 1).$$

For these three distributions, and for each pair $(n, \kappa(A))$ (or (n, ncond) in case of **Distribution III**), we ran the two algorithms (CG and AMGM) ten times from different randomly generated instances of linear systems of equations, and report the average of the following values: number of iterations, execution time (CPU–time) and the gradient norm evaluated in the solution found by the algorithms, for each of these three distributions. A summary of the numerical results on these three different distributions is reported in Tables 3, 4 and 5. From these tables, we can observe that both methods achieve the same level of accuracy on all problems. In addition, our solver converges faster than CG on all eigenvalue distributions analyzed.

4.3 Randomly generated tridiagonal linear systems of equations.

In the third set of test problems, we analyze the behavior of the methods by solving randomly generated tridiagonal linear systems of equations. In particular, the tridiagonal matrix $A \equiv [a_{ij}] \in \mathbb{R}^{n\times n}$ is constructed as follows, the diagonal entries are given by

$$a_{11} = |e_1| + p_1, \quad a_{nn} = |e_{n-1}| + p_n, \quad a_{ii} = |e_i| + |e_{i-1}| + p_i, \text{ for } i = 2, \ldots, n-1,$$

where $e \in \mathbb{R}^{n-1}$ is a randomly generated vector, whose elements follow a standard normal distribution, and $p \in \mathbb{R}^n$ is a vector whose elements are defined by

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

In addition, the sub–diagonal and the super–diagonal of A are defined precisely by the elements of e. In order to form the matrix A, we use the following Matlab commands

$$D = \texttt{sparse}(1 : n, 1 : n, d, n, n); \quad E = \texttt{sparse}(2 : n, 1 : n-1, e, n, n);$$

Table 6: Numerical results on tridiagonal random problems.

		CG			AMGM		
n	**ncond**	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
5000	5	161.8	0.006	9.26e-9	**157.8**	0.012	9.47e-9
	10	1967.3	0.056	9.70e-9	**1843.0**	0.103	9.94e-9
	15	23835.0	0.670	9.50e-9	**21580.0**	1.227	9.99e-9
10000	5	164.6	0.012	9.08e-9	**160.6**	0.022	9.25e-9
	10	2027.7	0.150	9.78e-9	**1897.5**	0.255	9.91e-9
	15	25029.0	1.861	9.85e-9	**22535.0**	2.933	9.99e-9
100000	5	173.2	0.103	9.44e-9	**169.3**	0.149	9.31e-9
	10	2138.3	1.295	9.94e-9	**2003.9**	1.706	9.94e-9
	15	26906.0	20.219	9.95e-9	**24172.0**	25.933	9.99e-9
500000	5	179.0	1.046	9.42e-9	**175.1**	1.390	9.28e-9
	10	2199.0	13.527	9.94e-9	**2064.5**	17.515	9.94e-9
	15	27725.0	172.409	9.97e-9	**24962.0**	215.182	9.99e-9
1000000	5	181.2	2.178	9.63e-9	**177.2**	3.139	9.62e-9
	10	2225.5	27.150	9.93e-9	**2091.0**	38.597	9.93e-9
	15	28029.0	361.054	9.99e-9	**25265.0**	495.573	9.99e-9

and

$$A = E + D + E',$$

where $d = [a_{11}, a_{22}, \ldots, a_{nn}]$, that is, $d \in \mathbb{R}^n$ is the vector whose elements are the diagonal entries of A. Additionally, the vector $b \in \mathbb{R}^n$ and the initial point $x_0 \in \mathbb{R}^n$ were created by using the following Matlab commands

$$y = -10 + 20 * \texttt{rand}(n, 1); \quad b = A * y; \quad x_0 = -10 + 20 * \texttt{rand}(n, 1).$$

For this experiment we use $\epsilon = 1e$-9 as the pre–established tolerance for the stop criterion. The numerical results associated to this experiment are shown in Table 6. In total, we generate ten independent instances of this tridiagonal problems and report the average of number of iterations (Nitr), the average of CPU time in seconds (Time), the average of the gradient norm $||\nabla f(\hat{x})||_2$ evaluated at the estimated solution $\hat{x}$ (NrmG). We can observe from Table 6 that AMGM is considerably competitive with respect to CG. In particular, AMGM converges in fewer iterations than CG. However, CG achieves the required precision in less computational time on all instances.

4.4 Sparse large–scale linear systems of equations.

In this subsection, we evaluate the numerical performance of the methods on large–scale sparse linear systems of equations using real data. In particular, we consider 60 matices taken from the UF Sparse Matrix Collection [6][1]. The vector $b \in \mathbb{R}^n$ and the initial point $x_0 \in \mathbb{R}^n$ were created as follows: $x^* = (1, 2, 3, \ldots, n)^\top$ then $b = Ax^*$, and the initial points was fixed at $x_0 = (1, 1, 1, \ldots, 1)^\top$. For this numerical test, we let the algorithms run up to $N = 150000$ iterations and stop them at iteration $k < N$ if $||\nabla f(x_k)||_2 < 1e-9||\nabla f(x_0)||_2$.

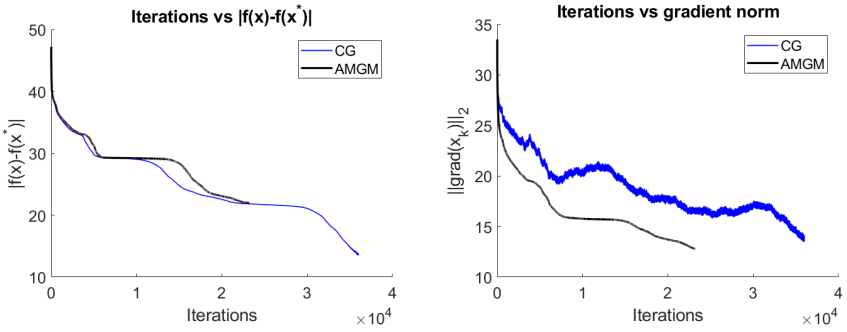
A detailed summary of the computational results is reported in Tables 7 and 8. From these tables, we can see that AGMGM method is considerably faster than CG, since it can always take less CPU time than the CG method and also takes fewer iterations to reach the required tolerance. In particular, our procedure achieves a winning percentage of 93.3% with three ties on the 60 instances, in terms of iterations. Specifically, our proposal solves all instances in 13758.75 iterations on average, and takes 115.085 seconds on average, while the CG method needs on average 31073.88 iterations and 147.387 seconds to find the solutions of all instances. In addition, we note that the CG method fails to converge on *bcsstk24*, *bcsstk25*, *bcsstk36* and *msc23052*, in which it uses the maximum number of iterations allowed without obtaining the required accuracy on the gradient norm.

We further illustrate the residuals $|f(x_k) - f(x^*)|$ and $||\nabla f(x_k)||_2$, where $x^* = A^{-1}b$, throughout the iterations for the instances *nasasrb* and *s3rmq4m1* in Figures 2 and 3, respectively. From these figures, we observe that the CG method shows a non–monotone decrease in terms of the gradient norm $||\nabla f(x_k)||_2$, while this one does achieve a smooth descent in terms of the f–residual $|f(x_k) - f(x^*)|$. In contrast, these two residuals are reduced monotonically in the AMGM method.

5 Conclusions

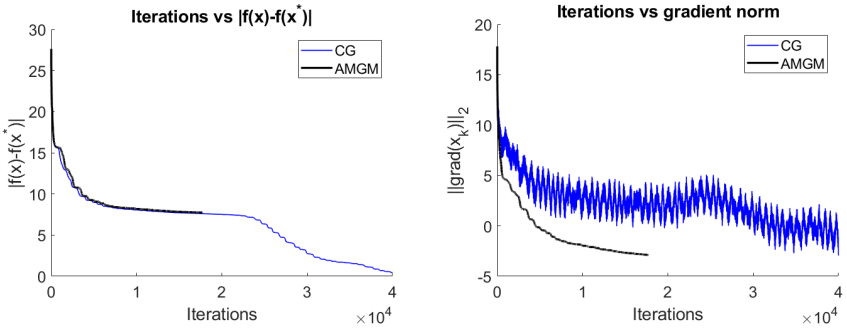
In this article, we have proposed a new accelerated gradient method with momentum for solving symmetric and positive definite linear systems of equations. The update formula of our iterative scheme can be seen as an extension of the well–known Nesterov's accelerated gradient method. In each step of the process, our algorithm guarantees a decrease in the residual norm. In addition, our proposal converges linearly to the unique solution of (1) and we also prove that it is at least as efficient as the gradient method equipped with the minimal gradient step–length. A numerical study, with diverse syntectic problems taken from the literature, corroborates that our accelerated minimal gradient method has a performance and efficiency superior to some state–of–the–art gradient methods and also is competitive against the well–known conjugate gradient

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



(a) Iterations vs f–residual (b) Iterations vs gradient norm

Figure 2: Convergence history of AMGM and CG, from the same initial point, for the minimization of a strictly convex quadratic function defined by the instance nasasrb ($n = 5489$). On the left, we plot the distance between $f(x_k)$ and the optimal value $f(x^*)$ of the cost function, throughout the iterations. On the right, we plot the sequence $\{||\nabla f(x_k)||_2\}_{k\in\mathbb{N}}$. In both plots, the y–axis is on a logarithmic scale.



(a) Iterations vs f–residual (b) Iterations vs gradient norm

Figure 3: Convergence history of AMGM and CG, from the same initial point, for the minimization of a strictly convex quadratic function defined by the instance s3rmq4m1 ($n = 54870$). On the left, we plot the distance between $f(x_k)$ and the optimal value $f(x^*)$ of the cost function, throughout the iterations. On the right, we compare the convergence behaviour in terms of the gradient norm $||\nabla f(x_k)||_2$. In both plots, the y–axis is on a logarithmic scale.

Table 7: Numerical results for linear systems from the UF sparse matrix collection, part I.

		CG			AMGM		
Name	**n**	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
1138_bus	1138	2412	0.02	8.04e-10	**2285**	0.10	9.91e-10
2cubes_sphere	101492	85004	208.92	9.71e-10	**34674**	147.58	9.99e-10
af_0_k101	503625	43174	889.07	9.99e-10	**25802**	841.40	9.99e-10
af_1_k101	503625	44530	879.55	9.99e-10	**25740**	562.77	9.99e-10
af_2_k101	503625	46345	922.68	9.98e-10	**24799**	1041.60	9.99e-10
af_3_k101	503625	36574	733.98	9.99e-10	**18816**	597.48	9.99e-10
af_4_k101	503625	47396	943.21	9.98e-10	**21555**	704.23	9.99e-10
af_5_k101	503625	48333	971.11	9.99e-10	**22087**	723.06	9.99e-10
af_shell3	504855	4161	84.10	9.33e-11	**3711**	83.31	9.99e-10
f_shell7	504855	4170	86.46	9.76e-10	**3707**	84.46	9.97e-10
apache1	80800	2610	2.23	9.06e-10	**2489**	2.50	9.82e-10
apache2	715176	4909	53.35	9.89e-10	**4453**	64.88	9.95e-10
bcsstk08	1074	4765	0.08	9.40e-10	**4184**	0.13	9.99e-10
bcsstk09	1083	283	0.01	9.15e-10	**280**	0.01	8.69e-10
bcsstk10	1086	3619	0.08	9.94e-10	**3170**	0.12	9.99e-10
bcsstk11	1473	10833	0.36	9.99e-10	**8593**	0.38	9.99e-10
bcsstk13	2003	126707	8.30	8.55e-10	**45440**	3.76	9.99e-10
bcsstk14	1806	12130	0.59	9.45e-10	**5732**	0.42	9.99e-10
bcsstk15	3948	18490	1.65	9.05e-10	**8876**	1.17	9.99e-10
bcsstk16	4884	509	0.11	9.92e-10	**348**	0.10	9.89e-10
bcsstk17	10974	20991	9.78	9.96e-10	**19459**	10.92	9.99e-10
bcsstk18	11948	150000	42.97	1.75e-09	**20654**	7.10	9.99e-10
bcsstk21	3600	9357	0.33	9.90e-10	**7941**	0.50	9.99e-10
bcsstk23	3134	48668	2.58	9.95e-10	**8280**	0.79	9.99e-10
bcsstk24	3562	150000	17.09	2.06e-08	**47170**	6.92	9.99e-10
bcsstk25	15439	150000	53.12	5.88e-08	**69466**	31.12	9.99e-10
bcsstk26	1922	20417	0.56	9.92e-10	**13293**	0.61	9.99e-10
bcsstm23	3134	4285	0.08	9.54e-10	**2935**	0.13	9.98e-10
bcsstm24	3562	2489	0.05	9.51e-10	**1986**	0.10	1.00e-10
Flan_1565	1564794	18980	2305.60	9.96e-10	**13389**	1664.70	9.99e-10

Table 8: Numerical results for linear systems from the UF sparse matrix collection, part II

		CG			AMGM		
Name	**n**	**Nitr**	**Time**	**NrmG**	**Nitr**	**Time**	**NrmG**
bcsstk27	1224	1001	0.04	8.91e-10	**906**	0.06	9.98e-10
bcsstk28	4410	14000	2.10	8.57e-10	**13704**	2.63	9.96e-10
bcsstk36	23052	150000	220.83	2.68e-08	**69277**	119.12	9.99e-10
bcsstk38	8032	23382	7.56	9.48e-10	**5822**	2.37	9.99e-10
crystm01	4875	102	0.01	7.83e-10	**99**	0.02	9.03e-10
crystm02	13965	104	0.05	8.90e-10	**100**	0.05	9.65e-10
crystm03	24696	103	0.08	9.80e-10	**99**	0.10	9.69e-10
ex15	6867	2104	0.18	9.75e-10	**1382**	0.18	9.99e-10
fv1	9604	**31**	0.01	8.76e-10	**31**	0.01	7.61e-10
fv2	9801	**31**	0.01	8.76e-10	**31**	0.01	7.61e-10
fv3	9801	208	0.03	9.69e-10	**204**	0.04	9.09e-10
Kuu	7102	741	0.21	9.63e-10	**689**	0.21	9.98e-10
mhd4800b	4800	52840	2.00	9.98e-10	**28675**	1.87	9.99e-10
msc04515	4515	5226	0.39	9.33e-10	**5092**	0.57	9.97e-10
msc23052	23052	150000	231.00	1.37e-08	**52143**	92.04	9.99e-10
Muu	7102	71	0.01	9.63e-10	**70**	0.02	8.04e-10
nasa4704	4704	21621	1.71	9.30e-10	**15923**	1.93	9.99e-10
nasasrb	54870	37643	116.34	9.48e-10	**23200**	75.11	9.99e-10
s1rmq4m1	5489	5615	1.22	8.42e-10	**4703**	1.23	9.99e-10
s2rmq4m1	5489	17820	3.47	9.45e-10	**15283**	3.55	9.99e-10
s1rmt3m1	5489	6153	0.96	9.82e-10	**5564**	1.12	9.98e-10
s2rmt3m1	5489	24844	3.82	9.62e-10	**21170**	4.04	9.87e-11
s3rmq4m1	5489	39984	7.81	9.73e-10	**17720**	4.52	9.99e-10
s3rmt3m1	5489	63635	9.45	9.70e-10	**24429**	4.81	9.99e-10
s3rmt3m3	5357	96268	14.19	9.72e-10	**26334**	5.19	9.99e-10
sts4098	4098	20690	1.52	9.74e-10	**16933**	1.80	9.99e-10
t2dal_e	4257	5474	0.14	9.45e-10	**2374**	0.14	9.99e-10
bcsstm08	1074	**120**	0.00	4.37e-10	130	0.01	6.51e-10
bcsstm11	1473	**29**	0.00	1.22e-10	**29**	0.00	1.22e-10
bcsstm12	1473	2452	0.05	7.74e-10	**2095**	0.08	9.97e-10

method. As future work, it is necessary to study the possible extensions of the proposed method both in the field of unconstrained convex optimization and in the general unconstrained non–linear case.

Acknowledgements

This work was supported in part by CONACYT (Mexico), Grants 258033 and 256126.

References

[Ang] Ang, A. Heavy ball method on convex quadratic problem.

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

[3] Bertsekas, D. P. (1999). *Nonlinear programming*. Athena scientific Belmont.

[4] Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538.

[5] Cyrus, S., Hu, B., Van Scoy, B., and Lessard, L. (2018). A robust accelerated optimization algorithm for strongly convex functions. In *2018 Annual American Control Conference (ACC)*, pages 1376–1381. IEEE.

[6] Davis, T. A. and Hu, Y. (2011). The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1.

[7] De Asmundis, R., Di Serafino, D., Hager, W. W., Toraldo, G., and Zhang, H. (2014). An efficient gradient method using the yuan steplength. *Computational Optimization and Applications*, 59(3):541–563.

[8] Frassoldati, G., Zanni, L., and Zanghirati, G. (2008). New adaptive stepsize selections in gradient methods. *Journal of industrial and management optimization*, 4(2):299–312.

[9] Hestenes, M. R. and Stiefel, E. (1952). *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC.

[10] Leon, H. F. O. (2019). A delayed weighted gradient method for strictly convex quadratic minimization. *Computational Optimization and Applications*, 74(3):729–746.

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

[12] Luenberger, D. G., Ye, Y., et al. (1984). *Linear and nonlinear programming*, volume 2. Springer.

[13] Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate o(1/k$\hat{2}$). *Soviet Mathematics Doklady*, 27:372–376.

[14] Nesterov, Y. (2013). *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media.

[15] Oveido, H. F., Dalmau, O. S., and Herrera, R. (2019). Two novel gradient methods with optimal step sizes. *Submitted to Journal of Computational Mathematics*.

[16] Oviedo, H., Dalmau, O., and Herrera, R. (2020). A hybrid gradient method for strictly convex quadratic programming. *Optimization-online*. http://www.optimization-online.org/DB_HTML/2020/02/7633.html.

[17] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[18] Zhou, B., Gao, L., and Dai, Y.-H. (2006). Gradient methods with adaptive step-sizes. *Computational Optimization and Applications*, 35(1):69–86.