

Full Terms & Conditions of access and use can be found at
<https://www.tandfonline.com/action/journalInformation?journalCode=goms20>



Accelerated multiple step-size methods for solving unconstrained optimization problems

Branislav Ivanov^a, Predrag S. Stanimirović^b, Gradimir V. Milovanović^c, Snežana Djordjević^d and Ivona Brajević^b

^aTechnical Faculty in Bor, University of Belgrade, Bor, Serbia; ^bFaculty of Sciences and Mathematics, University of Niš, Niš, Serbia; ^cSerbian Academy of Sciences and Arts, Belgrade, Serbia; ^dFaculty of Technology, University of Niš, Niš, Serbia

ABSTRACT

Two transformations of gradient-descent iterative methods for solving unconstrained optimization are proposed. The first transformation is called modification and it is defined using a small enlargement of the step size in various gradient-descent methods. The second transformation is termed as hybridization and it is defined as a composition of gradient-descent methods with the Picard–Mann hybrid iterative process. As a result, several accelerated gradient-descent methods for solving unconstrained optimization problems are presented, investigated theoretically and numerically compared. The proposed methods are globally convergent for uniformly convex functions satisfying certain condition under the assumption that the step size is determined by the backtracking line search. In addition, the convergence on strictly convex quadratic functions is discussed. Numerical comparisons show better behaviour of the proposed methods with respect to some existing methods in view of the Dolan and Moré's performance profile with respect to all analysed characteristics: number of iterations, the CPU time, and the number of function evaluations.

ARTICLE HISTORY

Received 27 August 2018
Accepted 1 August 2019

KEYWORDS

Unconstrained optimization;
gradient-descent methods;
convergence; line search

2010 MATHEMATICS SUBJECT

CLASSIFICATIONS
65K05; 90C30

1. Introduction and overview of related results

The objective of this paper is to study the convergence properties and practical computational performance of four new methods, created with the aim to solve the following unconstrained optimization problem

$$\min f(x), \quad x \in \mathbb{R}^n. \quad (1)$$

It is assumed that the function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is uniformly convex and twice continuously differentiable.

The most frequently used general iterative scheme aimed to solve the multivariable unconstrained minimization problem (1) is the most general iterative scheme

$$x_{k+1} = x_k + t_k d_k, \quad (2)$$

where x_{k+1} is a new iterative point, x_k is the previous iterative point, $t_k > 0$ is a step length, and d_k is a search direction. The key problem is to find the descent direction vector d_k and a suitable step size t_k . The search direction d_k must satisfy the descent condition $g_k^T d_k < 0$. The most frequent descent direction is $d_k = -g_k$, which produces the gradient descent (GD) iterative scheme

$$x_{k+1} = x_k - t_k g_k, \quad (3)$$

where t_k is defined by the inexact or exact line search. The most important method for computing t_k is the backtracking line search. The backtracking line search procedure from [1] starts from $t = 1$ and it reduces the objective function sufficiently in each iteration. The following Algorithm 1.1 from [33] will be used in order to implement the inexact line search which determines the step size t_k .

Algorithm 1.1 The backtracking line search.

Require: Objective function $f(x)$, the direction d_k of the search at the point x_k and numbers $0 < \sigma < 0.5$ and $\beta \in (0, 1)$.

- 1: $t = 1$.
 - 2: While $f(x_k + t d_k) > f(x_k) + \sigma t g_k^T d_k$, take $t := t\beta$.
 - 3: Return $t_k = t$.
-

The Newton method with line search is defined by

$$x_{k+1} = x_k - t_k G_k^{-1} g_k, \quad (4)$$

wherein G_k^{-1} denotes the inverse of the Hessian matrix G_k defined by $G_k(x) = \nabla^2 f(x_k)$ and $g_k = \nabla f(x_k)$ is the gradient vector, and the step size t_k is computed using an inexact line search. The general iterative scheme of quasi-Newton type with line search

$$x_{k+1} = x_k - t_k H_k g_k \quad (5)$$

assumes that B_k is appropriately generated symmetric positive definite approximation of G_k and $H_k = B_k^{-1}$ [35]. The update B_{k+1} of B_k is defined upon the quasi-Newton property (secant equation)

$$B_{k+1} s_k = y_k, \quad \text{where } s_k = x_{k+1} - x_k, \quad y_k = g_{k+1} - g_k. \quad (6)$$

Brezinski in [5] classified known methods for updating the matrix B_k used in (5). Three common approaches in defining B_k are: scalar matrix $B_k = \lambda_k I$, diagonal matrix $B_k = \text{diag}(\lambda_1, \dots, \lambda_n)$ and an appropriate full matrix. In accordance with the necessity to modify and extend these methods to make them suitable for large problems [19], we exploit the simplest scalar approximation to the Hessian:

$$B_k = \gamma_k I \approx G_k, \quad \gamma_k > 0, \quad (7)$$

where I is appropriate identity matrix. Our central interest in the present paper leads to algorithms which are defined by the iterative rule

$$x_{k+1} = x_k - \gamma_k^{-1} t_k g_k, \quad (8)$$

where $g_k = \nabla f(x_k)$ is the gradient vector and $\gamma_k > 0$ is a parameter aimed to improve the behaviour of the gradient-descent algorithm and t_k denotes the basic step size. The

iterations (8) will be termed as *improved gradient-descent (IGD)* methods. Usually, the parameter t_k is defined using the inexact line search procedure, and γ_k is defined according to the Taylor's expansion of the objective function $f(x)$.

Andrei in [1,3] defined iterations in the form

$$x_{k+1} = x_k - \theta_k t_k g_k. \quad (9)$$

Approach based on random values of θ_k which are uniformly distributed inside $(0, 1]$ was proposed in [3]. Later, Andrei in [1] proposed Algorithm 1.2 for finding an appropriate value for θ_k in (9).

Algorithm 1.2 Determine the scalar θ_k from (9) as in [1].

Require: Objective function $f(x)$,

- 1: (Backtracking) Find the step size $t_k \in (0, 1]$ using Algorithm 1.1.
 - 2: Compute $z = x_k - t_k g_k$, $g_z = \nabla f(z)$ and $y_k = g_z - g_k$.
 - 3: Compute $a_k = t_k (g_k)^T g_k$, $b_k = -t_k (y_k)^T g_k$.
 - 4: Return $\theta_k = a_k / b_k$.
-

The iterative rule (9) within which θ_k is defined using Algorithm 1.2 was termed as *Accelerated Gradient Descent (AGD)* in [1]. Following this notation, these iterations will be denoted by

$$x_{k+1}^{AGD} = x_k^{AGD} - \theta_k^{AGD} t_k g_k^{AGD}. \quad (10)$$

Several variants and modifications of the *IGD* iterative scheme (8) were proposed in [22,23,25,33,34]. Now we are going to make a survey of *IGD* methods in order to reach our iteration schemes. The accelerated gradient methods defined in [33] are of the type (8), in which the approximation of the Hessian is defined by the scalar matrix $\gamma_k I$, where $\gamma_k = \gamma(x_k, x_{k-1})$ is the matching acceleration parameter. The *SM* method originated in [33] was defined by the iterative process

$$x_{k+1}^{SM} = x_k^{SM} - t_k (\gamma_k^{SM})^{-1} g_k^{SM}, \quad (11)$$

where g_k^{SM} is the gradient vector, $t_k > 0$ is the step length defined by the backtrack inexact line search and $\gamma_k^{SM} > 0$ is the acceleration parameter defined on the basis of the Taylor's approximation of the function f at the point x_{k+1}^{SM} , as follows:

$$\gamma_{k+1}^{SM} = 2\gamma_k^{SM} \frac{\gamma_k^{SM} [f(x_{k+1}^{SM}) - f(x_k^{SM})] + t_k \|g_k^{SM}\|^2}{t_k^2 \|g_k^{SM}\|^2}.$$

The *Double direction* and *double step-size* accelerated methods, termed as *ADSS* and *ADD* methods, respectively, were presented in [22,23]. The *ADD* method is based on the usage of two search directions

$$x_{k+1}^{ADD} = x_k^{ADD} - t_k (\gamma_k^{ADD})^{-1} g_k^{ADD} + t_k^2 d_k^{ADD}, \quad (12)$$

where d_k is an appropriate second search direction.

The next scheme was proposed as the *Accelerated double step-size* (ADSS) model in [22]:

$$x_{k+1}^{ADSS} = x_k^{ADSS} - \left(t_k (\gamma_k^{ADSS})^{-1} + l_k \right) g_k^{ADSS}, \quad (13)$$

where t_k and l_k are step sizes, derived by two independent backtracking procedures: the first backtracking calculates t_k , while the second backtracking gives the value of l_k . The TADSS method from [34] is defined upon the assumption $t_k + l_k = 1$, which implies the iterative rule

$$x_{k+1}^{TADSS} = x_k^{TADSS} - \psi_k g_k^{TADSS},$$

where $\psi_k = t_k((\gamma_k^{TADSS})^{-1} - 1) + 1$. The acceleration parameters of ADD, ADSS and TADSS methods are defined, respectively, as follows:

$$\gamma_{k+1}^{ADD} = 2 \frac{f(x_{k+1}^{ADD}) - f(x_k^{ADD}) - t_k (g_k^{ADD})^T (t_k d_k^{ADD} - (\gamma_k^{ADD})^{-1} g_k^{ADD})}{(t_k d_k^{ADD} - (\gamma_k^{ADD})^{-1} g_k^{ADD})^T}, \quad (ADD \text{ method [25]})$$

$$\gamma_{k+1}^{ADSS} = 2 \frac{f(x_{k+1}^{ADSS}) - f(x_k^{ADSS}) + (t_k (\gamma_k^{ADSS})^{-1} + l_k) \|g_k^{ADSS}\|^2}{(t_k (\gamma_k^{ADSS})^{-1} + l_k)^2 \|g_k^{ADSS}\|^2}, \quad (ADSS \text{ method [24]})$$

$$\gamma_{k+1}^{TADSS} = 2 \frac{f(x_{k+1}^{TADSS}) - f(x_k^{TADSS}) + \psi_k \|g_k^{TADSS}\|^2}{\psi_k^2 \|g_k^{TADSS}\|^2}, \quad (TADSS \text{ method [36]}).$$

$$\psi_k = t_k ((\gamma_k^{TADSS})^{-1} - 1) + 1$$

The efficiency of IGD methods with accelerated parameters defined using the Taylor expansion was numerically tested in [26].

The author in [12] considered two *Relaxed Gradient Descent Quasi Newton* (RGDQN and RGDQN1) iterative schemes of the form

$$x_{k+1} = x_k - \theta_k t_k \gamma_k^{-1} g_k, \quad (14)$$

where θ_k is the relaxation parameter. The RGDQN iterations are defined with random values θ_k from the interval $(0, 1)$, while the RGDQN1 algorithm uses the relaxation θ_k which is defined by

$$\theta_k = \frac{\gamma_k}{t_k \gamma_{k+1}}.$$

The RGDQN and RGDQN1 iterative schemes confirm that AGD iterative schemes with more than two acceleration parameters could be usable.

The author of [16] applied the TADSS iterative scheme in minimizing underage costs for spare assemblies and subassemblies in aviation industry. In spite of the fact that these costs are hard for quantification, a combination of the TADSS iterations with the method for spare parts inventory forecasting based on Rayleigh's model and the Newsvendor model is able to determine the underage cost in predefined time intervals.

Moreover, we will exploit the Picard–Mann hybrid iterative process which was defined in [15]. It is assumed that the mapping $T : \mathbb{C} \rightarrow \mathbb{C}$ in (16) is defined on a nonempty convex

subset $\mathbb{C}$ of a normed space $\mathbb{E}$. The hybrid iterations define the iterative sequences x_k, y_k by the next three relations

$$\begin{aligned} x_1 &= x \in \mathbb{C}, \\ x_{k+1} &= Ty_k, \\ y_k &= (1 - \alpha_k)x_k + \alpha_k Tx_k, \quad k \in \mathbb{N}. \end{aligned} \quad (15)$$

The real number $\alpha_k \in (0, 1)$ from (15) is denoted in [25] as the correction parameter. An equivalent aggregated single iteration of (15) is defined as

$$x_{k+1} = H(T)(x_k) = T[(1 - \alpha_k)x_k + \alpha_k Tx_k], \quad k \in \mathbb{N}. \quad (16)$$

Let us mention that the iterations (16) will be denoted by $H(T, x_k) = H(T)(x_k)$ in order to clarify the presentation.

The author in [15] used a chosen set of constant values for this parameter ($\alpha = \alpha_k \in (0, 1), \forall k$) for numerical experiments and showed that the process (16) converges faster than the Picard, Mann and Ishikawa iterative processes from [14,17,27].

The iterative process (16) was used in [25] to create an accelerated hybridization of the SM method (denoted by *HSM*). Using the operator T in (15) or (16) to be equal to the SM iterative rule (11):

$$T(x_k) := x_k^{SM} - (\gamma_k^{SM})^{-1} t_k g_k^{SM},$$

then the iterations (16) become so called *HSM* iterative rule of the form

$$x_{k+1}^{HSM} := H(SM)(x_k) = x_k^{HSM} - (\alpha_k + 1)(\gamma_k^{HSM})^{-1} t_k g_k^{HSM}, \quad (17)$$

where x_{k+1}^{HSM} is a new iterative point, x_k^{HSM} is the previous iterative point, g_k^{HSM} is the corresponding gradient vector, t_k is a step length, α_k correction parameter and $\gamma_k^{HSM} > 0$ is the acceleration parameter defined by

$$\gamma_{k+1}^{HSM} = 2\gamma_k^{HSM} \frac{\gamma_k^{HSM} [f(x_{k+1}^{HSM}) - f(x_k^{HSM})] + (\alpha_k + 1)t_k \|g_k^{HSM}\|^2}{(\alpha_k + 1)^2 t_k^2 \|g_k^{HSM}\|^2}.$$

In [21], the authors proposed so called modified *HSM* (*MHSM*) method by finding appropriate initial step-size parameter in the backtracking procedure.

Hybridization of the *ADD* method was proposed and investigated in [24]. The resulting iterations are of the form

$$x_{k+1}^{HADD} = x_k^{HADD} - (\alpha_k + 1)t_k (\gamma_k^{HADD})^{-1} g_k^{HADD} + (\alpha_k + 1)t_k^2 d_k,$$

wherein

$$\gamma_{k+1}^{HADD} = 2 \frac{f(x_{k+1}^{HADD}) - f(x_k^{HADD}) - (\alpha_k + 1)(g_k^{HADD})^T (t_k^2 d_k - t_k (\gamma_k^{HADD})^{-1} g_k^{HADD})}{(\alpha_k + 1)^2 t_k^2 (t_k d_k - (\gamma_k^{HADD})^{-1} g_k^{HADD})^T (t_k d_k - (\gamma_k^{HADD})^{-1} g_k^{HADD})}.$$

It is worth mentioning that the *IGD* iterations (8) in the case $\gamma_k = 1$ become the gradient-descent (*GD*) iterative scheme (3). Usually, the step length t_k is defined by the inexact or exact line search.

On the other hand, the *IGD* iterations (8) in the case $t_k = 1$ become the gradient descent (*GD*) iterative scheme

$$x_{k+1} = x_k - \gamma_k^{-1} g_k, \quad (18)$$

wherein γ_k can be defined in different ways. Barzilai and Borwein in [4] proposed two variants of the *GD* method, termed as *BB* method, with the steplength $\gamma_k^{BB} := \gamma_k^{-1}$ in (18). The steplength γ_k^{BB} in the first variant is defined after the minimization of the norm $\|s_{k-1} - \gamma y_{k-1}\|^2$ with respect to γ , which yields

$$\gamma_k^{BB} = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}. \quad (19)$$

The steplength γ_k^{BB} in symmetric case is computed on the basis of the minimization of $\|\gamma s_{k-1} - y_{k-1}\|^2$, which yields

$$\gamma_k^{BB} = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}. \quad (20)$$

As a consequence, the *BB* iterations are defined as

$$x_{k+1}^{BB} = x_k^{BB} - \gamma_k^{BB} g_k^{BB}.$$

The *BB* method was modified in many articles, such as [6,7,8–11,29,30,36,37]. So called *Scalar Correction* (*SC*) method from [18] proposed the trial steplength in (18) defined by

$$\gamma_{k+1}^{SC} = \begin{cases} \frac{s_k^T r_k}{y_k^T r_k}, & y_k^T r_k > 0 \\ \frac{\|s_k\|}{\|y_k\|}, & y_k^T r_k \leq 0. \end{cases} \quad r_k = s_k - \gamma_k y_k. \quad (21)$$

The *SC* iterations are defined as

$$x_{k+1}^{SC} = x_k^{SC} - \gamma_k^{SC} g_k^{SC}.$$

Relaxed steepest descent and *BB* method by a parameter $\theta_k \in (0, 2)$ are considered in [28].

In general, our intention is to introduce and investigate theoretically and numerically two modifications of gradient-descent algorithms. We will use the term *accelerated* gradient descent algorithms to denote these modifications. The first acceleration is termed as *modification*, and it is based on an appropriate small enlargement of basic the step size in gradient-descent methods. The second acceleration is called *hybridization*, and it is based on a proper composition of accelerated gradient-descent methods and the Picard–Mann hybrid iterative process. Globally observed, we investigate possibility to use composite step size in gradient-descent algorithms. Composite step size is generated as a function of different parameters. These parameters could be considered as multiple step sizes which produce the final step length in gradient algorithms according to certain rules.

Main highlights of the present paper can be emphasized as follows.

- (1) Transformation of gradient-descent methods, called *modification*, is proposed and investigated theoretically and numerically. The modification is defined by replacing

the basic step size t_k in *GD* and *AGD* methods as well as in the *IGD* iterative class by the step size $t_k + t_k^2 - t_k^3$. The resulting iterations will be termed as *MGD*, *MAGD* and *MIGD*, respectively.

- (2) A *hybridization* of gradient-descent methods is defined as a composition of all considered *GD* methods and modified *GD* methods with the Picard–Mann hybrid iterative process.
- (3) Convergence properties of defined methods on the class of uniformly convex as well as on strictly convex quadratic functions are investigated.
- (4) Numerical experiments analyse three main characteristics of iterative methods: number of iterations, the CPU time, and the number of function evaluations.

The rest of the paper is developed by following the next organization. Modification of gradient descent methods is introduced in Section 2. Section 2.1 describes Modified *AGD* (*MAGD*) method, while Section 2.2 is aimed to the Modified *IGD* (*MIGD*) method. In Section 3, we define the *HGD* method as a result of the hybridization of the *GD* iteration with the Picard–Mann hybrid iterative process. Hybridization of *AGD* methods (*HAGD*) and *AMGD* methods (*HMAGD*) is defined in Section 3.1. Section 3.2 defines the hybridization of *IGD* methods (*HIGD*) and *MIGD* methods (*HMIGD*). The *HMSM* method is stated in a particular case. We also present the *HMSM* method that is created by modifying the *MSM* method. Section 4 investigates the convergence properties of the presented *MSM*, *HMSM*, *MAGD* and *HMAGD* methods. In Section 5, we report some numerical results and compare the performance of the proposed methods with some existing methods. Some final conclusions are given in the Section 6.

2. Modification of gradient-descent methods

The modification of *GD* iterations (3) is denoted by $MGD = \mathcal{M}(GD)$ and defined by the iterative rule

$$x_{k+1} = \mathcal{M}(GD)(x_k) = x_k - (t_k + t_k^2 - t_k^3) g_k. \quad (22)$$

We will use the notation *MGD* to denote the iterations (22). The main idea used in defining the iterations (22) is the replacement of the basic step size t_k in the *GD* methods (3) by a new basic step size $t_k^{mod} = t_k + t_k^2 - t_k^3$.

The underlying idea in defining the step size t_k^{mod} is given by the paper [24], where the iterative scheme is given by the next relation

$$x_{k+1} = x_k - \alpha t_k \gamma_k^{-1} g_k + \alpha t_k^2 d_k, \quad \alpha \in (1, 2).$$

On the other hand, having in view that $t_k^{mod} > t_k$, we are obviously trying to use a slightly greater step size in the aim to make the method which is better than some existing methods in this area.

It is assumed that t_k is defined by the backtracking procedure defined in Algorithm 1.1, which implies $t_k \in (0, 1)$. As a consequence, the justification for this modification lies in the inequalities

$$t_k \leq t_k + t_k^2 - t_k^3 \leq t_k + t_k^2.$$

As a conclusion, (22) is based on a relatively small increase in the step length t_k inside the interval $[t_k, t_k + t_k^2]$.

2.1. Modified AGD method

Using the same notation as in the previous section, the AGD process (10) is transformed into the *Modified AGD* method (MAGD shortly) as

$$x_{k+1}^{MAGD} = \mathcal{M}(AGD)(x_k^{MAGD}) = x_k^{MAGD} - \theta_k(t_k + t_k^2 - t_k^3)g_k^{MAGD}. \quad (23)$$

In this way, we introduce an iterative method for unconstrained optimization, which can be termed as *MAGD* method.

Algorithm 2.1 Modified accelerated gradient-descent method (the *MAGD* method).

Require: Objective function $f(x)$ and chosen initial point $x_0^{MAGD} \in \text{dom}(f)$.

- 1: Set $k = 0$ and compute $f(x_0^{MAGD})$ and $g_0^{MAGD} = \nabla f(x_0^{MAGD})$.
 - 2: If test criteria are fulfilled, then stop the iteration; otherwise, go to the next step.
 - 3: (Backtracking) Find the step size $t_k \in (0, 1]$ using Algorithm 1.1.
 - 4: Compute $z^{MAGD} = x_k^{MAGD} - t_k g_k^{MAGD}$, $g_z^{MAGD} = \nabla f(z^{MAGD})$ and $y_k^{MAGD} = g_z^{MAGD} - g_k^{MAGD}$.
 - 5: Compute $a_k = t_k(g_k^{MAGD})^T g_k^{MAGD}$, $b_k = -t_k(y_k^{MAGD})^T g_k^{MAGD}$ and $\theta_k = a_k/b_k$.
 - 6: Compute $x_{k+1}^{MAGD} = x_k^{MAGD} - \theta_k(t_k + t_k^2 - t_k^3)g_k^{MAGD}$.
 - 7: Compute $f(x_{k+1}^{MAGD})$ and $g_{k+1}^{MAGD} = \nabla f(x_{k+1}^{MAGD})$.
 - 8: Set $k := k + 1$, go to the step 2.
 - 9: Return x_{k+1}^{MAGD} and $f(x_{k+1}^{MAGD})$.
-

2.2. Modified IGD methods

The modification of *IGD* iterations, denoted by $MIGD = \mathcal{M}(IGD)$, produces the class of iterations of the general form

$$x_{k+1}^{MIGD} = \mathcal{M}(IGD)(x_k^{MIGD}) = x_k^{MIGD} - \gamma_k^{-1}(t_k + t_k^2 - t_k^3)g_k^{MIGD}, \quad (24)$$

where γ_k is appropriately defined using Taylor expansion. Clearly, since $\gamma_k > 0$, it follows that

$$\gamma_k^{-1}t_k \leq \gamma_k^{-1}(t_k + t_k^2 - t_k^3),$$

which implies that (24) defines an appropriate modification of the *IGD* class of methods, termed as *MIGD* class.

It is possible to consider modifications of various *IGD* methods, such as *SM*, *ADSS*, *TADSS*. These modifications will be defined by the functions $MSM = \mathcal{M}(SM)$, $MADSS = \mathcal{M}(ADSS)$ and $MTADSS = \mathcal{M}(TADSS)$, respectively. Only $\mathcal{M}(SM)$ will be described in

details. An application of the transformation $\mathcal{M}$ to the SM method leads to iterations

$$x_{k+1}^{MSM} = \mathcal{M}(SM)(x_k^{MSM}) = x_k^{MSM} - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} g_k^{MSM}. \quad (25)$$

In that case, from Taylor expansion of the second rate, the approximation of $f(x_{k+1}^{MSM})$ can be brought as follows:

$$\begin{aligned} f(x_{k+1}^{MSM}) &\approx f(x_k^{MSM}) - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} (g_k^{MSM})^T g_k^{MSM} \\ &\quad + \frac{1}{2} (t_k + t_k^2 - t_k^3)^2 \left((\gamma_k^{MSM})^{-1} g_k^{MSM} \right)^T \nabla^2 f(\xi) (\gamma_k^{MSM})^{-1} g_k^{MSM}. \end{aligned} \quad (26)$$

The parameter ξ in (26) fulfils the condition $\xi \in [x_k^{MSM}, x_{k+1}^{MSM}]$. One possible value for ξ is

$$\xi = x_k^{MSM} + \delta(x_{k+1}^{MSM} - x_k^{MSM}) = x_k^{MSM} - \delta(t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} g_k^{MSM}, \quad 0 \leq \delta \leq 1. \quad (27)$$

Following [33], the matrix $\nabla^2 f(\xi)$ is replaced by the diagonal matrix $\gamma_{k+1}^{MSM} I$. Based on the previous, the expression (26) becomes

$$\begin{aligned} f(x_{k+1}^{MSM}) &\approx f(x_k^{MSM}) - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} \|g_k^{MSM}\|^2 \\ &\quad + \frac{1}{2} (t_k + t_k^2 - t_k^3)^2 \gamma_{k+1}^{MSM} (\gamma_k^{MSM})^{-2} \|g_k^{MSM}\|^2. \end{aligned} \quad (28)$$

Then γ_{k+1}^{MSM} can be expressed from (28) in the following way:

$$\gamma_{k+1}^{MSM} = 2\gamma_k^{MSM} \frac{\gamma_k^{MSM} [f(x_{k+1}^{MSM}) - f(x_k^{MSM})] + (t_k + t_k^2 - t_k^3) \|g_k^{MSM}\|^2}{(t_k + t_k^2 - t_k^3)^2 \|g_k^{MSM}\|^2}. \quad (29)$$

Again, similarly as in [33], here we assume that $\gamma_{k+1}^{MSM} > 0$; otherwise the Second-Order Necessary Condition and Second-Order Sufficient Condition will not be fulfilled. If the unwanted situation $\gamma_{k+1}^{MSM} < 0$ happens in any iterative step, then the difficulty can be resolved by taking $\gamma_{k+1}^{MSM} = 1$.

For the end of this section, in Algorithm 2.2 we display the MSM method:

Algorithm 2.2 Modified SM method (the MSM method).

Require: Objective function $f(x)$ and chosen initial point $x_0^{MSM} \in \text{dom}(f)$.

- 1: Set $k = 0$ and compute $f(x_0^{MSM})$, $g_0^{MSM} = \nabla f(x_0^{MSM})$ and take $\gamma_0^{MSM} = 1$.
 - 2: If test criteria are fulfilled, then stop the iteration; otherwise, go to the next step.
 - 3: (Backtracking) Find the step size $t_k \in (0, 1]$ using Algorithm 1.1.
 - 4: Compute $x_{k+1}^{MSM} = x_k^{MSM} - (\gamma_k^{MSM})^{-1} (t_k + t_k^2 - t_k^3) g_k^{MSM}$.
 - 5: Compute $f(x_{k+1}^{MSM})$ and $g_{k+1}^{MSM} = \nabla f(x_{k+1}^{MSM})$.
 - 6: Determine the scalar approximation γ_{k+1}^{MSM} of the Hessian of f at the point x_{k+1}^{MSM} using (29).
 - 7: If $\gamma_{k+1}^{MSM} < 0$, then take $\gamma_{k+1}^{MSM} = 1$.
 - 8: Set $k := k + 1$, go to the step 2.
 - 9: Return x_{k+1}^{MSM} and $f(x_{k+1}^{MSM})$.
-

3. Hybridization of gradient-descent methods

The second class of iterations is defined by the hybrid correction of the *GD* iterations (8). A hybrid form of the *GD* method is defined in the space $\mathbb{C} := \mathbb{R}^n$, assuming that the mapping $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$ in (15) is defined by the *GD* iteration, i.e. $Ty_k = GD(y_k) = y_k - t_k g_k$. Then, using (15), we are able to derive the next set of iterations:

$$\begin{aligned} x_1 &= x \in \mathbb{R}^n, \\ x_{k+1} &= GD(y_k) = y_k - t_k g_k, \\ y_k &= (1 - \alpha_k)x_k + \alpha_k GD(x_k) \\ &= (1 - \alpha_k)x_k + \alpha_k (x_k - t_k g_k) \\ &= x_k - \alpha_k t_k g_k, \quad k \in \mathbb{N}. \end{aligned} \tag{30}$$

By replacing the definition of y_k from (30) into the definition of x_{k+1} , we obtain the hybrid method $HGD = \mathcal{H}(GD)$ which is defined by

$$x_{k+1} = \mathcal{H}(GD)(x_k) = x_k - (\alpha_k + 1)t_k g_k. \tag{31}$$

Since $t_k \in (0, 1)$ and $\alpha_k + 1 \geq 1$, it follows that

$$t_k \leq (\alpha_k + 1)t_k,$$

which further means that the *HGD* iterations (31) define another increase of the step length in the *GD* class.

Moreover, we consider the acceleration (24) and hybridization (31) incorporated in a single iterative rule

$$x_{k+1} = \mathcal{H}(MGD)(x_k) = \mathcal{H}(\mathcal{M}(GD)(x_k))(x_k) = x_k - (\alpha_k + 1)(t_k + t_k^2 - t_k^3)g_k. \tag{32}$$

The class of iterations (32) is termed as *HMGD* = $\mathcal{H}(MGD)$ class. Since $t_k \in (0, 1)$ and $\alpha_k + 1 \geq 1$, it follows that

$$t_k \leq t_k + t_k^2 - t_k^3 \leq (\alpha_k + 1)(t_k + t_k^2 - t_k^3),$$

which further means that *HMGD* iterations (32) defines an increase of the step length in the *MGD* class and the *MGD* class is based on an increase of the step size in the *GD* class.

3.1. Hybridization of AGD and MAGD methods

In order to define a hybrid form of the *AGD* method, assume that $\mathbb{C} = \mathbb{R}^n$ and the mapping $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$ in (15) is defined by the *AGD* iteration, i.e. $Ty_k = AGD(y_k) = y_k - \theta_k t_k g_k$. Then, using (15), we are able to derive the next iterations:

$$\begin{aligned} x_1 &= x \in \mathbb{R}^n, \\ x_{k+1} &= AGD(y_k) = y_k - \theta_k t_k g_k, \\ y_k &= (1 - \alpha_k)x_k + \alpha_k AGD(x_k) \end{aligned}$$

$$\begin{aligned}
&= (1 - \alpha_k)x_k + \alpha_k (x_k - \theta_k t_k g_k) \\
&= x_k - \alpha_k \theta_k t_k g_k, \quad k \in \mathbb{N}.
\end{aligned} \tag{33}$$

By replacing the definition of y_k from (33) into the definition of x_{k+1} , we obtain

$$x_{k+1}^{HAGD} = \mathcal{H}(AGD)(x_k^{HAGD}) = x_k^{HAGD} - (\alpha_k + 1)\theta_k t_k g_k^{HAGD}. \tag{34}$$

Similarly, we obtain

$$x_{k+1}^{HMAGD} = \mathcal{H}(MAGD)(x_k^{HMAGD}) = x_k^{HMAGD} - (\alpha_k + 1)\theta_k(t_k + t_k^2 - t_k^3)g_k^{HMAGD}. \tag{35}$$

The step length parameter t_k is calculated using the inexact backtracking line search defined in Algorithm 1.1. It is possible to remark that we take a constant value from the interval $(0, 1)$ for the correction parameter α_k in each iteration, as proposed in [15]. Below we present Algorithm 3.1 the *HMAGD* method.

Algorithm 3.1 Hybridization of the *MAGD* method (the *HMAGD* method).

Require: Objective function $f(x)$, $\alpha_k \in (0, 1)$ and chosen initial point $x_0^{HMAGD} \in \text{dom}(f)$.

- 1: Set $k = 0$ and compute $f(x_0^{HMAGD})$ and $g_0^{HMAGD} = \nabla f(x_0^{HMAGD})$.
 - 2: If test criteria are fulfilled, then stop the iteration; otherwise, go to the next step.
 - 3: (Backtracking) Find the step size $t_k \in (0, 1]$ using Algorithm 1.1.
 - 4: Compute $z^{HMAGD} = x_k^{HMAGD} - t_k g_k^{HMAGD}$, $g_z^{HMAGD} = \nabla f(z^{HMAGD})$ and $y_k^{HAAGD} = g_z^{HMAGD} - g_k^{HMAGD}$.
 - 5: Compute $a_k = t_k (g_k^{HMAGD})^T g_k^{HMAGD}$, $b_k = -t_k (y_k^{HMAGD})^T g_k^{HMAGD}$ and $\theta_k = a_k / b_k$.
 - 6: Compute $x_{k+1}^{HMAGD} = x_k^{HMAGD} - (\alpha_k + 1)(t_k + t_k^2 - t_k^3)\theta_k g_k^{HMAGD}$.
 - 7: Compute $f(x_{k+1}^{HMAGD})$ and $g_{k+1}^{HMAGD} = \nabla f(x_{k+1}^{HMAGD})$.
 - 8: Set $k := k + 1$, go to the step 2.
 - 9: Return x_{k+1}^{HMAGD} and $f(x_{k+1}^{HMAGD})$.
-

3.2. Hybridization of *IGD* and *MIGD* methods

The second class of iterations is defined by the hybrid correction of the *IGD* iterations (8), which is defined by

$$x_{k+1} = \mathcal{H}(IGD)(x_k) = x_k - (\alpha_k + 1)\gamma_k^{-1} t_k g_k, \tag{36}$$

where $\alpha_k \geq 0$ is appropriately selected real parameter defined using (15).

Moreover, we consider the *MIGD* acceleration (24) and hybridization *HIGD* (36) incorporated in a single iterative rule

$$x_{k+1} = \mathcal{H}(MIGD)(x_k) = \mathcal{H}(\mathcal{M}(IGD)(x_k))(x_k) = x_k - (\alpha_k + 1)\gamma_k^{-1} (t_k + t_k^2 - t_k^3) g_k. \tag{37}$$

Since $t_k \in (0, 1)$ and $\alpha_k + 1 \geq 1$, it follows that

$$t_k \gamma_k^{-1} \leq (t_k + t_k^2 - t_k^3) \gamma_k^{-1} \leq (\alpha_k + 1) (t_k + t_k^2 - t_k^3) \gamma_k^{-1},$$

which further means that *HIGD* defined in (36) defines another increase of the step length in the *IGD* class and *HMIGD* (37) is an acceleration of *HIGD* and *MIGD*, and, consequently, of *IGD*. The class of iterations (36) is termed as *HIGD* class. Finally, the class of

iterations (37) will be termed as hybrid *MIGD* class (shortly *HMIGD* class). For example, the method (17) belongs to the *HIGD* class.

Particularly, the hybridization of the *MSM* iterations will be denoted by *HMSM* and it is defined as

$$x_{k+1}^{HMSM} = \mathcal{H}(MSM)(x_k^{HMSM}) = x_k^{HMSM} - (\alpha_k + 1)(t_k + t_k^2 - t_k^3)(\gamma_k^{HMSM})^{-1}g_k^{HMSM}. \quad (38)$$

To complete defining the *HMSM* method, we need to compute the value of the acceleration parameter γ_k^{HMSM} . The approximation of the Hessian of the objective function f is given by the diagonal matrix

$$\nabla^2 f \approx \gamma_{k+1}^{HMSM} I, \quad (39)$$

in which scalar $\gamma_{k+1}^{HMSM} = \gamma(x_{k+1}^{HMSM}, x_k^{HMSM})$ is the appropriately selected real number based on the Taylor approximation of the function f at the point x_{k+1}^{HMSM} , computed by means of (38):

$$\begin{aligned} f(x_{k+1}^{HMSM}) &\approx f(x_k^{HMSM}) - (\alpha_k + 1)(t_k + t_k^2 - t_k^3)(g_k^{HMSM})^T (\gamma_k^{HMSM})^{-1} g_k^{HMSM} \\ &\quad + \frac{1}{2}(\alpha_k + 1)^2(t_k + t_k^2 - t_k^3)^2 \left((\gamma_k^{HMSM})^{-1} g_k^{HMSM} \right)^T \\ &\quad \times \nabla^2 f(\xi)(\gamma_k^{HMSM})^{-1} g_k^{HMSM}. \end{aligned} \quad (40)$$

In the previous equation, the variable ξ satisfies $\xi \in [x_k^{HMSM}, x_{k+1}^{HMSM}]$. Since the point x_k^{HMSM} is close enough to the point x_{k+1}^{HMSM} , it is reasonable to take $\xi = x_{k+1}^{HMSM}$. Now, on the basis of the equality (39), one can obtain the next equality

$$\begin{aligned} f(x_{k+1}^{HMSM}) &\approx f(x_k^{HMSM}) - (\alpha_k + 1)(t_k + t_k^2 - t_k^3)(\gamma_k^{HMSM})^{-1} \|g_k^{HMSM}\|^2 \\ &\quad + \frac{1}{2}(\alpha_k + 1)^2(t_k + t_k^2 - t_k^3)^2 \gamma_{k+1}^{HMSM} (\gamma_k^{HMSM})^{-2} \|g_k^{HMSM}\|^2. \end{aligned} \quad (41)$$

According to (41), γ_{k+1}^{HMSM} is computed in the following way:

$$\gamma_{k+1}^{HMSM} = 2\gamma_k^{HMSM} \frac{\gamma_k^{HMSM} [f(x_{k+1}^{HMSM}) - f(x_k^{HMSM})] + (\alpha_k + 1)(t_k + t_k^2 - t_k^3) \|g_k^{HMSM}\|^2}{(\alpha_k + 1)^2(t_k + t_k^2 - t_k^3)^2 \|g_k^{HMSM}\|^2}. \quad (42)$$

Again, all values $\gamma_{k+1}^{HMSM} < 0$ will be replaced by $\gamma_{k+1}^{HMSM} = 1$.

Now, everything is ready to describe the algorithm of the *HMSM* method:

4. Convergence analysis

In the following proposition and lemma, we restate and derive some basic statements needful for analysing the convergence properties of Algorithms 2.1, 2.2, 3.1 and 3.2.

Algorithm 3.2 Hybridization of the MSM method (the HMSM method).

Require: Objective function $f(x)$, $\alpha_k \in (0, 1)$ and chosen initial point $x_0^{HMSM} \in \text{dom}(f)$.

- 1: Set $k = 0$ and compute $f(x_0^{HMSM})$, $g_0^{HMSM} = \nabla f(x_0^{HMSM})$ and take $\gamma_0^{HMSM} = 1$.
- 2: If test criteria are fulfilled, then stop the iteration; otherwise, go to the next step.
- 3: (Backtracking) Find the step size $t_k \in (0, 1]$ using Algorithm 1.1.
- 4: Compute $x_{k+1}^{HMSM} = x_k^{HMSM} - (\alpha_k + 1)(t_k + t_k^2 - t_k^3)(\gamma_k^{HMSM})^{-1}g_k^{HMSM}$.
- 5: Compute $f(x_{k+1}^{HMSM})$ and $g_{k+1}^{HMSM} = \nabla f(x_{k+1}^{HMSM})$.
- 6: Determine the scalar γ_{k+1}^{HMSM} using (42).
- 7: If $\gamma_{k+1}^{HMSM} < 0$, then take $\gamma_{k+1}^{HMSM} = 1$.
- 8: Set $k := k + 1$, go to the step 2.
- 9: Return x_{k+1}^{HMSM} and $f(x_{k+1}^{HMSM})$.

The following assumptions will be used in this section:

- (H₁) the function f has a lower bound on $B_0 = \{x \in \mathbb{R}^n \mid f(x) \leq f(x_0)\}$, where $x_0 \in \mathbb{R}^n$;
(H₂) the gradient g of f is Lipschitz continuous in an open convex set $B \supseteq B_0$, i.e. there exists $L > 0$ such that

$$\|g(x) - g(y)\| \leq L\|x - y\|, \quad \forall x, y \in B. \quad (43)$$

The following result, restated from [1,32], will be useful.

Proposition 4.1 ([1,32]): *Let d_k be a descent direction and the gradient $g(x) = \nabla f(x)$ satisfies the Lipschitz condition (43). If the backtracking line search in Algorithm 1.1 is used, then*

$$t_k \geq \min \left\{ 1, -\frac{\beta(1-\sigma)}{L} \frac{g_k^T d_k}{\|d_k\|^2} \right\}. \quad (44)$$

The proofs of Proposition 4.2 and Lemma 4.1 can be found in [20,31]. These statements are restated for the sake of completeness.

Proposition 4.2: *If the function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice continuously differentiable and uniformly convex on $\mathbb{R}^n$, then (H₁) and (H₂) are satisfied.*

Lemma 4.1: *Under the assumptions of Proposition 4.2 there exist real numbers m, M satisfying*

$$0 < m \leq 1 \leq M, \quad (45)$$

such that $f(x)$ has an unique minimizer x^ and*

$$m\|y\|^2 \leq y^T \nabla^2 f(x) y \leq M\|y\|^2, \quad \forall x, y \in \mathbb{R}^n; \quad (46)$$

$$\frac{1}{2}m\|x - x^*\|^2 \leq f(x) - f(x^*) \leq \frac{1}{2}M\|x - x^*\|^2, \quad \forall x \in \mathbb{R}^n; \quad (47)$$

$$m\|x - y\|^2 \leq (g(x) - g(y))^T (x - y) \leq M\|x - y\|^2, \quad \forall x, y \in \mathbb{R}^n. \quad (48)$$

Lemma 4.2 was proposed in [33] for the *IGD* iterative scheme. It estimates the iterative decreasing of the objective function when the *IGD* method is applied.

Lemma 4.2 ([33]): *For twice continuously differentiable and uniformly convex function $f : \mathbb{R}^n \mapsto \mathbb{R}$, and for the *IGD* sequence $\{x_k\}$ generated by (8) the following inequality is valid*

$$f(x_k) - f(x_{k+1}) \geq \mu \|g_k\|^2, \quad (49)$$

where

$$\mu = \min \left\{ \frac{\sigma}{M}, \frac{\sigma(1-\sigma)}{L} \beta \right\}. \quad (50)$$

In subsequent, it is assumed that d_k is a descent direction. Important observation is that the scalar approximation of Hessian allows to avoid the assumption that f is twice continuously differentiable. Consequently, instead of (46) and (45) which assumes that f twice continuously differentiable and uniformly convex function, we will use the following simple requirement for γ_k :

$$m \leq \gamma_k \leq M, \quad 0 < m \leq 1 \leq M, \quad m, M \in \mathbb{R}. \quad (51)$$

In the case $\gamma_k < 0$ it is possible to use $\gamma_k = 1$, while in the case $\gamma_k > M$ we will use $\gamma_k = M$.

Theorem 4.1 investigates the convergence of *MIGD* method for uniformly convex functions under assumptions (H_1) and (H_2) .

Theorem 4.1: *Let the assumptions (H_1) and (H_2) be satisfied, let (51) be valid and $f : \mathbb{R}^n \mapsto \mathbb{R}$ is uniformly convex function. Then the *MIGD* sequence $\{x_k\}$ generated by (24) satisfies the inequality of the form (49), (50).*

Proof: The *MIGD* iterations are of the form $x_{k+1} = x_k + t_k d_k$, wherein $d_k = -\gamma_k^{-1}(1 + t_k - t_k^2)g_k$. According to the exit condition of the backtracking Algorithm 1.1, the next inequality is valid

$$f(x_k) - f(x_{k+1}) \geq -\sigma t_k g_k^T d_k. \quad \forall k \in \mathbb{N}. \quad (52)$$

In the case $t_k < 1$, taking into consideration (52) in conjunction with $d_k = -\gamma_k^{-1}(1 + t_k - t_k^2)g_k$, the following inequalities can be derived

$$f(x_k) - f(x_{k+1}) \geq -\sigma t_k g_k^T d_k = -\sigma t_k g_k^T (-\gamma_k^{-1}(1 + t_k - t_k^2)g_k).$$

Now, (44) implies

$$t_k \geq \frac{(1-\sigma)\beta}{L} \cdot \frac{\gamma_k}{1 + t_k - t_k^2},$$

and further

$$\begin{aligned} f(x_k) - f(x_{k+1}) &\geq \sigma \frac{(1-\sigma)\beta\gamma_k}{L(1 + t_k - t_k^2)} \cdot \frac{g_k^T g_k (1 + t_k - t_k^2)}{\gamma_k} \\ &\geq \sigma \frac{(1-\sigma)\beta}{L} \|g_k\|^2. \end{aligned}$$

In accordance with (51), the following inequality holds in the case $t_k = 1$:

$$\begin{aligned} f(x_k) - f(x_{k+1}) &\geq -\sigma g_k^T d_k = -\sigma g_k^T (-\gamma_k^{-1}(1 + t_k - t_k^2)g_k) \\ &\geq \frac{\sigma}{\gamma_k} \|g_k\|^2 \\ &\geq \frac{\sigma}{M} \|g_k\|^2. \end{aligned}$$

Finally from the above two inequalities derived in both possible cases, $t_k < 1$ and $t_k = 1$, we get (50) and the proof is complete. $\blacksquare$

Remark 4.1: (a) The same result as in Theorem 4.1 can be verified for the MAGD method. In order to verify this statement, it is necessary to replace γ_k^{-1} by θ_k .
 (b) The result as in Theorem 4.1 can be directly applied to MSM method.

Theorem 4.2 investigates bounds of iterative decreasing of the goal function when the HIGD method is applied.

Theorem 4.2: Let the assumptions (H_1) and (H_2) be satisfied in conjunction with (51) and $f : \mathbb{R}^n \mapsto \mathbb{R}$ is uniformly convex function.

(a) The inequality of the form (49) is valid for any HIGD sequence $\{x_k^{HIGD}\}$, where

$$\mu = \min \left\{ \frac{\sigma}{M}(\alpha_k + 1), \frac{\sigma(1 - \sigma)}{L}\beta \right\}. \quad (53)$$

(b) The inequality of the form (49) is valid for any HMIGD sequence $\{x_k^{HMIGD}\}$, where μ satisfies (53).

Proof: (a) We analyse the next two cases which refer to the value of the iterative step size: $t_k < 1$ and $t_k = 1$. According to the exit condition of the backtracking Algorithm 1.1, the inequality (52) is valid.

In the case $t_k < 1$, taking into account (44), we get

$$t_k \geq -\frac{\beta(1 - \sigma)}{L} \frac{g_k^T d_k}{\|d_k\|^2}.$$

Since the HIGD iterations satisfy

$$d_k = -(\alpha_k + 1)\gamma_k^{-1}g_k, \quad (54)$$

it follows that

$$t_k \geq \frac{\beta(1 - \sigma)\gamma_k}{L(\alpha_k + 1)}. \quad (55)$$

An application of (55) and (54) into (52) produces the following inequalities:

$$\begin{aligned} f(x_k) - f(x_{k+1}) &\geq -\sigma t_k g_k^T (-(\alpha_k + 1) \gamma_k^{-1} g_k) \\ &\geq \frac{\sigma(1-\sigma)\beta\gamma_k}{L(\alpha_k + 1)} \cdot \frac{g_k^T g_k (\alpha_k + 1)}{\gamma_k} \\ &\geq \frac{\sigma(1-\sigma)\beta}{L} \|g_k\|^2. \end{aligned}$$

On the other hand, in accordance with (51), the following inequality holds in the case $t_k = 1$:

$$f(x_k) - f(x_{k+1}) \geq -\sigma g_k^T d_k \geq \frac{\sigma}{M} (\alpha_k + 1) \|g_k\|^2.$$

Finally from the above two inequalities derived in the case $t_k < 1$ and $t_k = 1$, we get

$$f(x_k) - f(x_{k+1}) \geq \min \left\{ \frac{\sigma}{M} (\alpha_k + 1), \frac{\sigma(1-\sigma)}{L} \beta \right\} \|g_k\|^2$$

and the proof of the part (a) is completed.

The statement in (b) can be verified similarly. ■

Remark 4.2: After comparison of inequalities (50) and (53), it can be concluded that the *HIGD* iterations warrant greater decrease of $f(x_{k+1})$ with respect to $f(x_k)$ with respect to both the *IGD* and *MIGD* rule in the case $t_k = 1$, because of $\sigma/M(\alpha_k + 1) > \sigma/M$.

In Theorems 4.3 and 4.4 we prove a linear convergence of *MAGD* and *HIGD* methods, respectively, for uniformly convex functions.

Theorem 4.3: *Let the assumptions (H_1) and (H_2) be satisfied in conjunction with (51) and $f : \mathbb{R}^n \mapsto \mathbb{R}$ be uniformly convex function. If the sequence $\{x_k^{MAGD}\}$ is generated by Algorithm 2.1, then*

$$\lim_{k \rightarrow \infty} \|g_k^{MAGD}\| = 0, \quad (56)$$

and the sequence $\{x_k^{MAGD}\}$ converges to x^ at least linearly.*

Proof: The proof is similar as in [33, Theorem 4.1], and will be omitted. ■

Theorem 4.4: *Let the assumptions (H_1) and (H_2) be satisfied in conjunction with (51) and $f : \mathbb{R}^n \mapsto \mathbb{R}$ be uniformly convex function. If the sequence $\{x_k^{HIGD}\}$ is generated by the *HIGD* iterations, then*

$$\lim_{k \rightarrow \infty} \|g_k^{HIGD}\| = 0, \quad (57)$$

and the sequence $\{x_k^{HIGD}\}$ converges to x^ at least linearly.*

Corollary 4.1: *If the assumptions (H_1) and (H_2) are satisfied in conjunction with (51) and $f : \mathbb{R}^n \mapsto \mathbb{R}$ is uniformly convex function, the inequality of the form (49) is valid for the HGD, HSM, HAGD sequences $\{x_k\}$, where μ is defined as in (53). Also,*

$$\lim_{k \rightarrow \infty} \|g_k\| = 0,$$

and the sequence $\{x_k\}$ converges to the optimal value x^ at least linearly.*

In Lemma 4.3 we want to prove convergence of the MSM method on the class of strictly convex quadratic functions. Also, we discover the condition on smallest and largest eigenvalues of the matrix A which guarantee successful application of the MSM iterative scheme on the strictly convex quadratic functions given by

$$f(x) = \frac{1}{2}x^T A x - b^T x, \quad (58)$$

where A is a real $n \times n$ symmetric positive definite matrix and $b \in \mathbb{R}^n$. Particularly, Theorem 4.1 is valid for the MSM iterations defined in Algorithm 2.2. Denote the eigenvalues of the matrix A as $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$.

Lemma 4.3: *The following inequality holds for a strictly convex quadratic function f given by the expression (58) which involves symmetric positive definite matrix $A \in \mathbb{R}^n$ and the gradient-descent method (25) with the parameters γ_k^{MSM} determined according to (29) and the primary step size t_k defined in Algorithm 1.1:*

$$\lambda_1 \leq \frac{\gamma_{k+1}^{MSM}}{t_{k+1}} \leq \frac{2\lambda_n}{\sigma}, \quad k \in \mathbb{N}, \quad (59)$$

wherein the quantities λ_1 and λ_n represent the smallest and the largest eigenvalues of A , respectively.

Proof: The difference between two successive values of f defined in (58) is equal to

$$f(x_{k+1}^{MSM}) - f(x_k^{MSM}) = \frac{1}{2}(x_{k+1}^{MSM})^T A x_{k+1}^{MSM} - b^T x_{k+1}^{MSM} - \frac{1}{2}(x_k^{MSM})^T A x_k^{MSM} + b^T x_k^{MSM}. \quad (60)$$

The replacement of (25) in (60) gives

$$\begin{aligned} f(x_{k+1}^{MSM}) - f(x_k^{MSM}) &= \frac{1}{2} \left[x_k^{MSM} - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} g_k^{MSM} \right]^T \\ &\quad \times A \left[x_k^{MSM} - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} g_k^{MSM} \right] \\ &\quad - b^T \left[x_k^{MSM} - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} g_k^{MSM} \right] \\ &\quad - \frac{1}{2} (x_k^{MSM})^T A x_k^{MSM} + b^T x_k^{MSM} \\ &= -\frac{1}{2} (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} (x_k^{MSM})^T A g_k^{MSM} \\ &\quad - \frac{1}{2} (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} (g_k^{MSM})^T A x_k^{MSM} \end{aligned}$$

$$\begin{aligned}
& + \frac{1}{2}(t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-2}(g_k^{MSM})^T A g_k^{MSM} \\
& + (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} b^T g_k^{MSM}.
\end{aligned}$$

Since the gradient of the function (58) is equal to $g_k^{MSM} = Ax_k^{MSM} - b$, one can verify

$$\begin{aligned}
f(x_{k+1}^{MSM}) - f(x_k^{MSM}) &= (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} [b^T g_k^{MSM} - (x_k^{MSM})^T A g_k^{MSM}] \\
&+ \frac{1}{2}(t_k + t_k^2 - t_k^3)^2 (\gamma_k^{MSM})^{-2} (g_k^{MSM})^T A g_k^{MSM} \\
&= (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} [b^T - (x_k^{MSM})^T A] g_k^{MSM} \\
&+ \frac{1}{2}(t_k + t_k^2 - t_k^3)^2 (\gamma_k^{MSM})^{-2} (g_k^{MSM})^T A g_k^{MSM} \\
&= -(t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} (g_k^{MSM})^T g_k^{MSM} \\
&+ \frac{1}{2}(t_k + t_k^2 - t_k^3)^2 (\gamma_k^{MSM})^{-2} (g_k^{MSM})^T A g_k^{MSM}. \tag{61}
\end{aligned}$$

After replacing (61) into (29), the parameter γ_{k+1}^{MSM} becomes

$$\begin{aligned}
\gamma_{k+1}^{MSM} &= 2\gamma_k^{MSM} \frac{\gamma_k^{MSM} [f(x_{k+1}^{MSM}) - f(x_k^{MSM})] + (t_k + t_k^2 - t_k^3) \|g_k^{MSM}\|^2}{(t_k + t_k^2 - t_k^3)^2 \|g_k^{MSM}\|^2} \\
&= \frac{(g_k^{MSM})^T A g_k^{MSM}}{\|g_k^{MSM}\|^2}.
\end{aligned}$$

The last relation reveals that γ_{k+1}^{MSM} is the Rayleigh quotient of the real symmetric matrix A at the vector g_k^{MSM} . So, the next inequalities hold:

$$\lambda_1 \leq \gamma_{k+1}^{MSM} \leq \lambda_n, \quad k \in \mathbb{N}. \tag{62}$$

The verification of the left inequality in (59) is straight from (62), since $0 < t_{k+1} \leq 1$. To prove the right inequality in (59), we use the upper bound initiated by the line search

$$t_k \geq \frac{\beta(1-\sigma)\gamma_k}{L},$$

which leads to the next inequality:

$$\frac{\gamma_{k+1}^{MSM}}{t_{k+1}} < \frac{L}{\beta(1-\sigma)}. \tag{63}$$

Taking into account that the gradient of the objective (58) is equal to $g(x) = Ax - b$ in conjunction with the assumption that the real matrix A symmetric, the next can be concluded:

$$\|g(x) - g(y)\| = \|Ax - Ay\| = \|A(x - y)\| \leq \|A\| \|x - y\| = \lambda_n \|x - y\|. \tag{64}$$

On the basis of the last equation, we can conclude that Lipschitz constant L in (63) can take the largest eigenvalue λ_n of the matrix A . Considering the estimations for the Backtracking

parameters $\sigma \in (0, 0.5)$ and $\beta \in (\sigma, 1)$ we finally get

$$\frac{\gamma_{k+1}^{MSM}}{t_{k+1}} < \frac{L}{\beta(1-\sigma)} = \frac{\lambda_n}{\beta(1-\sigma)} < \frac{2\lambda_n}{\sigma}. \quad (65)$$

Therefore, the right inequality in (59) is proved. ■

The convergence of the MSM method under the additional assumption $\lambda_n < 2\lambda_1$ is considered in Theorem 4.5.

Theorem 4.5: *Let f be a strictly convex quadratic function given by (58). If the eigenvalues of the matrix A satisfy the additional assumption $\lambda_n < 2\lambda_1$, then the MSM method (25) satisfies*

$$(d_i^{k+1})^2 \leq \delta^2 (d_i^k)^2, \quad (66)$$

where

$$\delta = \max \left\{ 1 - \frac{\sigma \lambda_1}{2\lambda_n}, \frac{\lambda_n}{\lambda_1} - 1 \right\}. \quad (67)$$

In addition,

$$\lim_{k \rightarrow \infty} \|g_k^{MSM}\| = 0. \quad (68)$$

Proof: Let $\{x_k^{MSM}\}$ be the sequence generated by Algorithm 2.2. Assume that $\{v_1, v_2, \dots, v_n\}$ are orthonormal eigenvectors of A . Then for arbitrary vector x_k^{MSM} , using $g_k^{MSM} = Ax_k^{MSM} - b$, there exist real constants $d_1^k, d_2^k, \dots, d_n^k$ such that

$$g_k^{MSM} = \sum_{i=1}^n d_i^k v_i. \quad (69)$$

Now, using (25) one can simply verify the following

$$\begin{aligned} g_{k+1}^{MSM} &= Ax_{k+1}^{MSM} - b \\ &= A(x_k^{MSM} - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} g_k^{MSM}) - b \\ &= g_k^{MSM} - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} A g_k^{MSM} \\ &= \left(I - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} A \right) g_k^{MSM}. \end{aligned}$$

Using the simple linear representation for g_{k+1}^{MSM} of the form (69), we get

$$g_{k+1}^{MSM} = \sum_{i=1}^n d_i^{k+1} v_i = \sum_{i=1}^n \left(1 - (t_k + t_k^2 - t_k^3)(\gamma_k^{MSM})^{-1} \lambda_i \right) d_i^k v_i. \quad (70)$$

To prove (66), it is enough to show that $|1 - \lambda_i/(\gamma_k^{MSM}(t_k + t_k^2 - t_k^3)^{-1})| \leq \delta$. There are two cases. First, if $\lambda_i \leq \frac{\gamma_k^{MSM}}{t_k + t_k^2 - t_k^3}$ implying (59), we can conclude the following:

$$1 > \frac{\lambda_i}{\gamma_k^{MSM}(t_k + t_k^2 - t_k^3)^{-1}} \geq \frac{\sigma \lambda_1}{2\lambda_n} \implies 1 - \frac{\lambda_i}{\gamma_k^{MSM}(t_k + t_k^2 - t_k^3)^{-1}} \leq 1 - \frac{\sigma \lambda_1}{2\lambda_n} \leq \delta. \quad (71)$$

Now, let us examine another case $\gamma_k^{MSM}/(t_k + t_k^2 - t_k^3) < \lambda_i$. Since

$$1 < \frac{\lambda_i}{\gamma_k^{MSM}(t_k + t_k^2 - t_k^3)^{-1}} \leq \frac{\lambda_n}{\lambda_1}, \quad (72)$$

it follows that

$$\left| 1 - \frac{\lambda_i}{\gamma_k^{MSM}(t_k + t_k^2 - t_k^3)^{-1}} \right| \leq \frac{\lambda_n}{\lambda_1} - 1 \leq \delta. \quad (73)$$

Now, in order to prove $\lim_{k \rightarrow \infty} \|g_k^{MSM}\| = 0$, we use the orthonormality of the eigenvectors $\{v_1, v_2, \dots, v_n\}$ as well as (69) and get

$$\|g_k^{MSM}\|^2 = \sum_{i=1}^n (d_i^k)^2. \quad (74)$$

Since (66) is satisfied and $0 < \delta < 1$ holds, in view of (74) it follows that (69) holds, which completes our proof. ■

Convergence properties of hybrid methods (*HMIGD*, *HMAGD* and *HMSM*) in the case of strictly convex quadratic functions can be proven analogously on the basis of Lemma 4.3 and Theorem 4.5.

5. Numerical experiments

In this section, we present numerical results obtained by testing *MAGD*, *MSM*, *HMAGD*, *HMSM* and *HSM* methods. It is important to mention that the *HSM* method in work [25] showed much better results than the *SM* method from [33]. In addition, the *SM* method in [33] gave better results than *AGD* and *GD* methods. For these reasons, we choose the *HSM* method for comparing versus to *MAGD*, *MSM*, *HMAGD* and *HMSM* methods. At the end of this section, we present the numerical results obtained by testing the *MSM* method and the *SM* method.

The codes used in the tests are written in the Matlab R2017a programming language, and the tests were performed on the computer Workstation Intel Core i3 2.0 GHz. The number of iterations, number of function evaluations and CPU time in all tested methods are analysed in numerical experiments.

Example 5.1: Numerical experiments are based on 28 test functions from [2]. For each of tested functions in Tables 1, 2 and 3 we have considered 12 different numerical experiments with the number of variables, equal to 100, 200, 300, 500, 1000, 2000, 3000, 5000, 7000, 8000, 10,000 and 15,000. Summary numerical results for *MAGD*, *MSM*, *HMAGD*, *HMSM* and *HSM*, tested on 28 large scale test functions, are presented in Tables 1, 2 and 3.

For each of five considered algorithms (*MAGD*, *HMAGD*, *MSM*, *HMSM* and *HSM*), we have taken the same stopping criteria used in [25]:

$$\|g_k\| \leq 10^{-6} \quad \text{and} \quad \frac{|f(x_{k+1}) - f(x_k)|}{1 + |f(x_k)|} \leq 10^{-16}.$$

Table 1. Summary numerical results for the number of iterations in Example 5.1.

Test function	Number of iterations				
	MAGD	HMAGD	MSM	HMSM	HSM
Extended penalty function	341	391	689	744	684
Perturbed quadratic function	352,325	356,402	34,828	84,420	79,198
Raydan 2 function	60	119	108	160	160
Diagonal 3 function	119,719	124,750	7030	21,287	20,626
Generalized tridiagonal 1 function	647	653	346	443	422
Extended tridiagonal 1 function	692,219	755,340	1370	82,248	74,898
Extended TET function	455	455	156	261	286
Diagonal 4 function	8084	10,590	96	1681	2055
Diagonal 5 function	48	109	72	120	120
Extended Himmelblau function	302	400	260	693	358
Perturbed quadratic diagonal function	1,060,824	5,965,848	37,454	196,373	155,484
Quadratic QF1 function	362,896	368,183	36,169	89,026	78,932
Extended quadratic penalty QP1 function	229	275	369	340	374
Extended quadratic penalty QP2 function	356,357	84,634	1674	22,385	20,432
Quadratic QF2 function	71,647	388,352	32,727	90,357	89,593
Extended quadratic exponential EP1 function	67	128	100	268	193
Extended tridiagonal 2 function	1665	1721	659	710	778
ARWHEAD function (CUTE)	12,834	71,741	430	4261	4151
Almost perturbed quadratic function	354,369	358,466	33,652	84,546	79,701
LIARWHD function (CUTE)	925,138	1,963,100	3029	271,705	244,509
ENGVAL1 function (CUTE)	822	821	461	561	522
QUARTC function (CUTE)	177	165	217	292	256
Diagonal 6 function	60	119	108	162	160
Generalized quartic function	229	270	181	209	226
Diagonal 7 function	159	216	147	266	209
Diagonal 8 function	154	216	120	202	177
Full Hessian FH3 function	63	153	63	207	186
Diagonal 9 function	325,609	614,270	10,540	79,802	63,237

The backtracking parameters for all algorithms are $\sigma = 0.0001$ and $\beta = 0.8$, which means that we accept a small decrease in f predicted by a linear approximation at the current point. The value of correction parameter $\alpha_k = 0.1$ was used in three hybrid methods (*HMAGD*, *HMSM* and *HSM*).

Table 4 contains the results corresponding to the average number of iterations, the number of function evaluations and the CPU time for all 336 numerical experiments.

Based on the results arranged in Table 4, it is observable that the *MSM* method gives four and more times better results compared to *MAGD*, *HMAGD*, *HMSM* and *HSM* methods. This conclusion is confirmed by performance profiles for the number of iterations, the number of function evaluations and the CPU time.

Performance profiles of a given metric, proposed in [13], is a widely used tool for benchmarking and comparing the performance of optimization software on a large test set. For performance measures, as usual, the number of iterations, number of function evaluations and computation time (CPU time) are used. Figure 1 (left) shows the performances of compared methods relative to the number of iterations. Figure 1 (right) illustrates the performance of these methods relative to the number of function evaluations. Figure 2 shows the performance of the considered methods relative to the CPU time. The top curve corresponds to the method that exhibits the best performances with respect to the chosen performance profile.

From the results displayed in Tables 1–3 and according to graphs included in Figure 1 (left), 1 (right) and Figure 2, we can conclude the following.

Table 2. Summary numerical results for the number of function evaluations in Example 5.1.

Test function	Number of function evaluations				
	MAGD	HMAGD	MSM	HMSM	HSM
Extended penalty function	9085	11,702	3479	3638	3460
Perturbed quadratic function	13,855,459	14,193,163	200,106	366,943	334,564
Raydan 2 function	132	250	228	332	332
Diagonal 3 function	4,244,404	4,482,972	38,158	93,632	88,698
Generalized tridiagonal 1 function	9057	9616	1191	1396	1330
Extended tridiagonal 1 function	2,077,341	3,021,492	10,989	425,411	387,939
Extended TET function	4130	4168	528	753	818
Diagonal 4 function	133,440	185,397	636	8140	9517
Diagonal 5 function	108	230	156	253	253
Extended Himmelblau function	5192	7164	976	2754	1172
Perturbed quadratic diagonal function	3,872,8371	236,316,190	341,299	1,018,378	807,185
Quadratic QF1 function	13,192,789	13,541,108	208,286	387,021	332,928
Extended quadratic penalty QP1 function	2939	3747	2196	1846	2141
Extended quadratic penalty QP2 function	8,846,145	2,282,567	11,491	116,071	105,841
Quadratic QF2 function	2,810,965	16,640,880	183,142	394,364	378,921
Extended quadratic exponential EP1 function	1513	2878	894	2500	1716
Extended tridiagonal 2 function	9613	10916	2866	2793	3010
ARWHEAD function (CUTE)	468,970	2,847,637	5322	27,050	28,015
Almost perturbed quadratic function	13,936,462	14,275,979	194,876	367,586	336,419
LIARWHD function (CUTE)	41,619,197	90,302,744	27,974	1,409,648	1,269,240
ENGVAL1 function (CUTE)	8332	8531	2285	2956	2700
QUARTC function (CUTE)	414	402	494	644	572
Diagonal 6 function	132	275	270	362	356
Generalized quartic function	1244	1696	493	526	592
Diagonal 7 function	745	1187	504	756	672
Diagonal 8 function	740	1064	383	753	589
Full Hessian FH3 function	1955	5508	566	1898	1541
Diagonal 9 function	12,984,028	25,166,521	68,189	392,059	307,951

- (1) The *MSM* method gives better results compared to other methods concerning all three considered performance criteria: number of iterations, number of function evaluations and the CPU time.
- (2) In general, the accelerated method shows better performances than the hybrid methods. Exactly, this means that *MAGD* is better than *HMAGD* as well as *MSM* with respect to *HSM*.
- (3) The class of *SM* methods (*MSM*, *HSM* and *HMSM*) exhibit better performances from the *AGD* methods (*MAGD* and *HMAGD*).

In Figure 1 (left), it is observable that all five methods successfully solve all the problems, and the *MSM* method is best in 75.0% of the test problems compared to the *MAGD*(25.0%), *HMAGD*(3.6%), *HMSM*(0%) and *HSM*(0%).

In Figure 1 (right), it is observable that all five methods successfully solve all the problems, and the *MSM* method is best in 75.0% of the test problems compared to the *MAGD*(10.7%), *HMAGD*(3.6%), *HMSM*(7.1%) and *HSM*(3.6%).

Graphs in Figure 2 show that all five methods successfully solve all the problems, and the *MSM* method is best in 78.6% of the test problems compared to the *MAGD*(10.7%), *HMAGD*(0%), *HMSM*(10.7%) and *HSM*(3.6%).

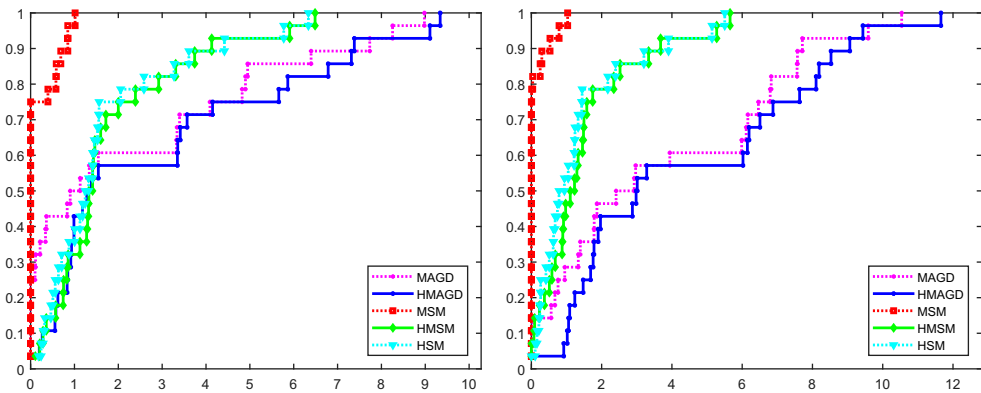
In Table 4, we can notice that the *MAGD* and *HMAGD* methods have extremely low average results for CPU time. For this reason, we will omit them from further testing.

Table 3. Summary numerical results for the CPU time (sec) in Example 5.1.

Test function	CPU time (sec)				
	MAGD	HMAGD	MSM	HMSM	HSM
Extended penalty function	2.69	3.19	1.59	1.38	1.53
Perturbed quadratic function	6049.53	6432.67	116.28	210.17	210.42
Raydan 2 function	0.17	0.23	0.23	0.20	0.28
Diagonal 3 function	6401.97	7049.88	52.61	153.34	155.52
Generalized tridiagonal 1 function	7.78	7.22	1.47	1.83	1.58
Extended tridiagonal 1 function	8853.17	11,247.73	29.05	1121.67	1018.38
Extended TET function	2.77	2.50	0.52	0.80	0.91
Diagonal 4 function	16.17	22.34	0.20	1.70	1.86
Diagonal 5 function	0.31	0.44	0.34	0.55	0.42
Extended Himmelblau function	1.03	1.36	0.30	0.66	0.31
Perturbed quadratic diagonal function	22,820.17	102,830.22	139.63	476.63	277.48
Quadratic QF1 function	6846.45	7960.61	81.53	155.34	128.80
Extended quadratic penalty QP1 function	1.06	1.25	1.00	0.92	0.84
Extended quadratic penalty QP2 function	1872.80	532.38	3.52	20.11	18.09
Quadratic QF2 function	768.56	5263.98	73.44	169.16	158.72
Extended quadratic exponential EP1 function	0.84	1.05	0.69	1.17	1.17
Extended tridiagonal 2 function	2.53	3.34	1.05	0.97	1.22
ARWHEAD function (CUTE)	138.00	1627.41	1.97	11.53	13.56
Almost perturbed quadratic function	7086.56	8258.72	73.05	148.22	131.58
LIARWHD function (CUTE)	15,372.63	32,393.92	9.25	707.83	635.06
ENGVAL1 function (CUTE)	2.64	2.47	1.05	1.42	1.27
QUARTC function (CUTE)	2.08	1.91	1.84	2.34	2.30
Diagonal 6 function	0.14	0.38	0.23	0.36	0.33
Generalized quartic function	0.50	0.70	0.28	0.28	0.38
Diagonal 7 function	0.69	1.02	0.55	0.80	0.84
Diagonal 8 function	0.66	1.08	0.47	1.13	0.67
Full Hessian FH3 function	1.19	3.13	0.39	1.72	1.39
Diagonal 9 function	6662.98	7734.27	43.61	219.52	104.09

Table 4. Average numerical outcomes for 28 test functions tested on 12 numerical experiments in Example 5.1.

Average performances	MAGD	HMAGD	MSM	HMSM	HSM
Number of iterations	165,982.11	395,281.68	7,251.96	36,918.89	32,783.11
Number of function evaluations	5,462,603.64	15,118,785.14	46,713.46	179,659.39	157,445.43
CPU time (sec)	2,961.29	6,835.19	22.72	121.85	102.46

**Figure 1.** Performance profiles based on the number of iterations (left) and function evaluations (right).

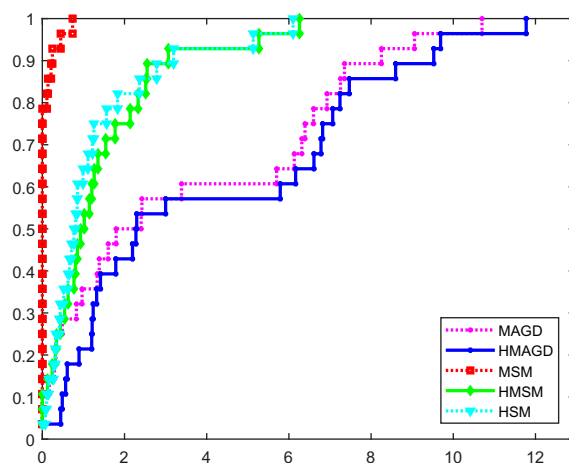


Figure 2. Performance profile based on CPU time.

Example 5.2: The goal of the next numerical experiment is to investigate the behaviour of the *MSM*, *HMSM*, and *HSM* methods with respect to an increase in the number of variables.

Table 5. Summary numerical results for the number of iterations in Example 5.2.

Test function	Number of iterations		
	MSM	HMSM	HSM
Extended penalty function	747	777	795
Perturbed quadratic function	90,056	216,249	208,908
Raydan 2 function	99	146	146
Diagonal 2 function	61,480	2761	57,062
Diagonal 3 function	9143	31,331	32,541
Generalized tridiagonal 1 function	310	409	381
Extended tridiagonal 1 function	1557	111,650	98,106
Extended TET function	143	231	254
Diagonal 4 function	88	1606	1958
Diagonal 5 function	66	110	110
Perturbed quadratic diagonal function	66,898	325,007	270,034
Quadratic QF1 function	94,931	236,085	219,454
Extended quadratic penalty QP1 function	408	353	400
Extended quadratic penalty QP2 function	1061	8496	7627
Quadratic QF2 function	89,646	246,260	244,162
Extended quadratic exponential EP1 function	93	280	211
Extended tridiagonal 2 function	650	654	711
ARWHEAD function (CUTE)	431	4396	4431
Almost perturbed quadratic function	88,120	222,865	219,976
LIARWHD function (CUTE)	6201	756,758	709,335
ENGVAL1 function (CUTE)	428	552	502
QUARTC function (CUTE)	210	277	244
Diagonal 6 function	99	151	149
COSINE function (CUTE)	220	175,767	248
Generalized quartic function	165	193	220
Diagonal 7 function	165	296	204
Diagonal 8 function	114	202	197
Full Hessian FH3 function	55	233	228
HIMMELH function (CUTE)	110	99	99
Extended Rosenbrock	55	55	55

Table 6. Summary numerical results for the number of function evaluations in Example 5.2.

Test function	Number of function evaluations		
	MSM	HMSM	HSM
Extended penalty function	4050	4125	4521
Perturbed quadratic function	531,633	941,045	883,112
Raydan 2 function	209	303	303
Diagonal 2 function	400,443	31,069	266,617
Diagonal 3 function	50,629	137,875	140,393
Generalized tridiagonal 1 function	1075	1357	1219
Extended tridiagonal 1 function	13,320	578,079	508,592
Extended TET function	484	657	736
Diagonal 4 function	583	7775	9080
Diagonal 5 function	143	232	232
Perturbed quadratic diagonal function	626,251	1,685,791	1,402,532
Quadratic QF1 function	563,538	1,027,186	927,053
Extended quadratic penalty QP1 function	2396	2083	2454
Extended quadratic penalty QP2 function	6940	43,453	39,132
Quadratic QF2 function	504,120	1,072,757	1,032,582
Extended quadratic exponential EP1 function	847	3147	2057
Extended tridiagonal 2 function	2936	2599	2708
ARWHEAD function (CUTE)	5844	29,860	32,806
Almost perturbed quadratic function	525,546	969,555	929,583
LIARWHD function (CUTE)	65,898	3,926,707	3,685,122
ENGVAL1 function (CUTE)	2324	3044	2863
QUARTC function (CUTE)	475	609	543
Diagonal 6 function	220	381	372
COSINE function (CUTE)	652	775,878	692
Generalized quartic function	451	485	579
Diagonal 7 function	621	1015	819
Diagonal 8 function	452	836	851
Full Hessian FH3 function	598	2486	2514
HIMMELH function (CUTE)	231	209	209
Extended Rosenbrock	121	121	121

Numerical experiments are based on 30 test functions from [2]. We have considered 11 different numerical experiments with the number of variables equal to 1000, 2000, 3000, 5000, 7000, 8000, 10,000, 15,000, 20,000, 30,000 and 50,000, for each function in Tables 5, 6 and 7. Summary numerical results for *MSM*, *HMSM* and *HSM*, tested on 30 large scale test functions, are presented in Tables 5, 6 and 7. The parameter values and stopping criteria are the same as in the previous numerical experiment.

Figure 3 (left), 3 (right) and Figure 4 respectively were created from the results shown in Tables 5, 6 and 7. Figure 3 (left) shows the performances of compared methods relative to the number of iterations. In this Figure, it is observable that *MSM*, *HMSM* and *HSM* methods successfully solve all the problems, and the *MSM* method is best in 90.0% of the test problems compared to the *HMSM* and *HSM*. Figure 3 (right) shows the performance of these methods relative to the number of function evaluations. Also in this Figure, it is observable that *MSM*, *HMSM* and *HSM* methods successfully solve all the problems, and the *MSM* method is best in 86.7% of the test problems compared to the *HMSM* and *HSM*.

Figure 4 shows the performance of the considered methods relative to the CPU time. In this Figure, it is observable that *MSM*, *HMSM* and *HSM* methods successfully solve all the problems, and the *MSM* method is best in 80.0% of the test problems compared to the *HMSM* and *HSM*.

Table 7. Summary numerical results for the CPU time in Example 5.2.

Test function	CPU time (sec)		
	MSM	HMSM	HSM
Extended penalty function	3.69	3.91	4.91
Perturbed quadratic function	748.80	1364.23	1312.34
Raydan 2 function	0.48	0.63	0.66
Diagonal 2 function	802.17	37.55	613.45
Diagonal 3 function	128.64	410.58	478.38
Generalized tridiagonal 1 function	3.00	4.08	3.39
Extended tridiagonal 1 function	68.13	3513.39	2850.02
Extended TET function	1.19	1.63	1.72
Diagonal 4 function	0.42	3.80	4.45
Diagonal 5 function	0.67	1.45	1.25
Perturbed quadratic diagonal function	696.84	1916.58	1609.41
Quadratic QF1 function	614.67	1056.86	976.09
Extended quadratic penalty QP1 function	1.94	1.91	2.14
Extended quadratic penalty QP2 function	11.08	52.48	49.23
Quadratic QF2 function	548.59	1201.77	1168.66
Extended quadratic exponential EP1 function	1.13	3.69	2.36
Extended tridiagonal 2 function	2.44	2.05	2.20
ARWHEAD function (CUTE)	3.98	27.72	26.77
Almost perturbed quadratic function	548.47	1010.25	984.58
LIARWHD function (CUTE)	60.05	5138.77	4750.78
ENGVAL1 function (CUTE)	3.02	3.34	2.72
QUARTC function (CUTE)	4.20	5.28	5.34
Diagonal 6 function	0.56	0.89	0.75
COSINE function (CUTE)	2.17	1776.70	1.75
Generalized quartic function	0.64	0.77	0.69
Diagonal 7 function	1.83	2.83	2.11
Diagonal 8 function	1.31	2.00	2.27
Full Hessian FH3 function	1.08	4.70	5.11
HIMMELH function (CUTE)	1.13	1.08	1.03
Extended Rosenbrock	0.13	0.30	0.20

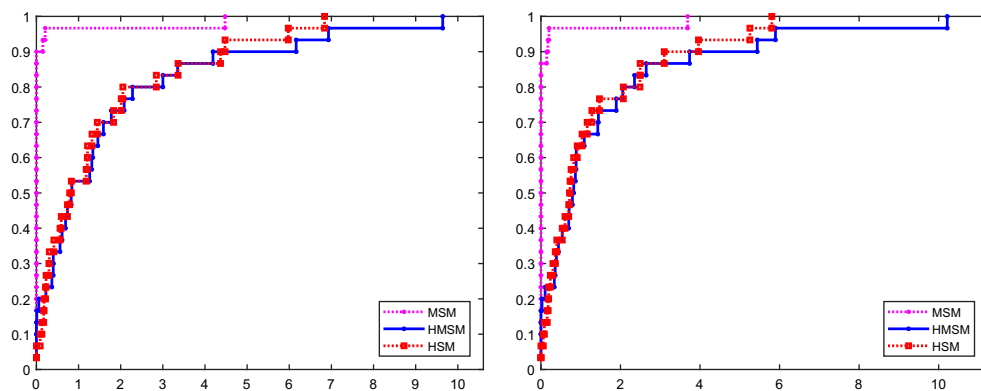
**Figure 3.** Performance profiles based on the number of iterations (left) and function evaluations (right).

Table 8 contains the average number of iterations, the CPU time, and the number of function evaluations for all 330 numerical experiments.

On the basis of the results obtained in Table 8, we can conclude that the *MSM* method has on average three and more times better results (number of iterations, the number of function evaluations and CPU time) than the other two methods.

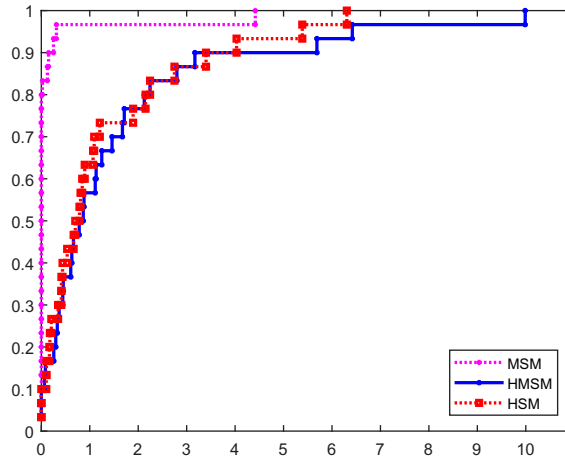


Figure 4. Performance profile based on CPU time.

Table 8. Average numerical outcomes for 30 test functions tested on 11 numerical experiments in Example 5.2.

Average performances	<i>MSM</i>	<i>HMSM</i>	<i>HSM</i>
Number of iterations	17,124.97	78,141.63	69,291.60
Number of function evaluations	110,434.33	375,023.97	329,346.57
CPU time (sec)	142.08	585.04	495.49

After increasing the number of variables from 15,000 to 50,000 in the second numerical experiment, it can be noticed that the ratio of (successful and fast) solved problems between the *MSM* method and the *HMSM* and *HSM* method remains approximately the same. The *MSM* method has three and more times better results from the other two methods.

In the following example, we compare *MSM* and *SM* methods to show superiority of the *MSM* method achieves better results with respect to number of iterations, number of function evaluations and the CPU time.

Example 5.3: The goal of the next numerical experiment is to investigate the behaviour of the *MSM* and *SM* methods with respect to a number of iterations, the number of function evaluations and CPU time.

Numerical experiments are based on 30 test functions from the Example 5.2. We have considered 11 different numerical experiments with the number of variables equal to 1000, 2000, 3000, 5000, 7000, 8000, 10,000, 15,000, 20,000, 30,000 and 50,000, for each function in Table 9. Summary numerical results for *MSM*, and *SM*, tested on 30 large scale test functions, are presented in Table 9. The parameter values and stopping criteria are the same as in the previous numerical experiments.

Figures 5 and 6 respectively were created from the results shown in Table 9. Figure 5 shows the performances of compared *MSM* and *SM* methods relative to the number of iterations (left) and the number of function evaluations(right). Figure 6 shows the performance of the considered methods relative to the CPU time. In Figures 5 and 6, it is

Table 9. Summary numerical results for the number of iterations, number of function evaluations and CPU time in Example 5.3.

Test function	Number of iterations		Number of function evaluations		CPU Time	
	MSM	SM	MSM	SM	MSM	SM
Extended penalty function	747	650	4050	2924	3.69	3.00
Perturbed quadratic function	90,056	160,542	531,633	916,964	748.80	1304.34
Raydan 2 function	99	99	209	209	0.48	0.59
Diagonal 2 function	61,480	58,288	400,443	326,472	802.17	970.42
Diagonal 3 function	9143	17,019	50,629	93,596	128.64	242.94
Generalized tridiagonal 1 function	310	293	1075	1020	3.00	3.86
Extended tridiagonal 1 function	1557	3871	13,320	32,720	68.13	143.08
Extended TET function	143	143	484	489	1.19	1.16
Diagonal 4 function	88	88	583	583	0.42	0.47
Diagonal 5 function	66	66	143	143	0.67	0.72
Perturbed quadratic diagonal function	66,898	81,148	626,251	853,416	696.84	915.50
Quadratic QF1 function	94,931	169,298	563,538	964,865	614.67	1046.86
Extended quadratic penalty QP1 function	408	298	2396	3262	1.94	2.89
Extended quadratic penalty QP2 function	1061	1630	6940	11,055	11.08	13.03
Quadratic QF2 function	89,646	176,214	504,120	986,671	548.59	1143.13
Extended quadratic exponential EP1 function	93	70	847	724	1.13	0.91
Extended tridiagonal 2 function	650	457	2936	2727	2.44	2.19
ARWHEAD function (CUTE)	431	309	5844	5308	3.98	6.06
Almost perturbed quadratic function	88,120	168,157	525,546	952,654	548.47	1050.47
LIARWHD function (CUTE)	6201	19,725	65,898	193,633	60.05	106.20
ENGVAL1 function (CUTE)	428	326	2324	2937	3.02	3.42
QUARTC function (CUTE)	210	277	475	609	4.20	5.28
Diagonal 6 function	99	99	220	220	0.56	0.61
COSINE function (CUTE)	220	198	652	636	2.17	1.63
Generalized quartic function	165	175	451	471	0.64	0.53
Diagonal 7 function	165	99	621	253	1.83	0.97
Diagonal 8 function	114	116	452	1127	1.31	2.61
Full Hessian FH3 function	55	55	598	608	1.08	1.14
HIMMELH function (CUTE)	110	110	231	231	1.13	1.25
Extended Rosenbrock	55	55	121	121	0.13	0.25

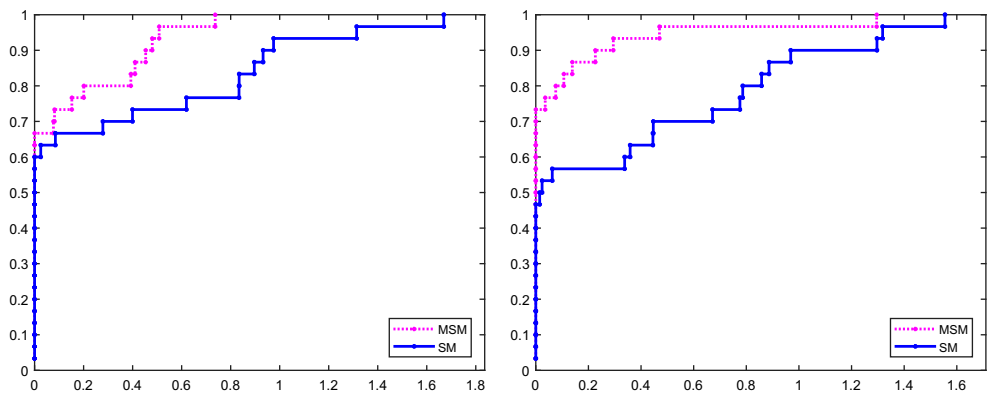


Figure 5. Performance profiles based on the number of iterations (left) and function evaluations (right).

observable that *MSM* and *SM* methods successfully solve all the problems, graph *MSM* method in all of those cases first come to the top which signifies that the *MSM* is the best. Based on the results shown in the tables, the average results, the created graphics and the comprehensive analysis, we can conclude that the *MSM* method gives the best results.

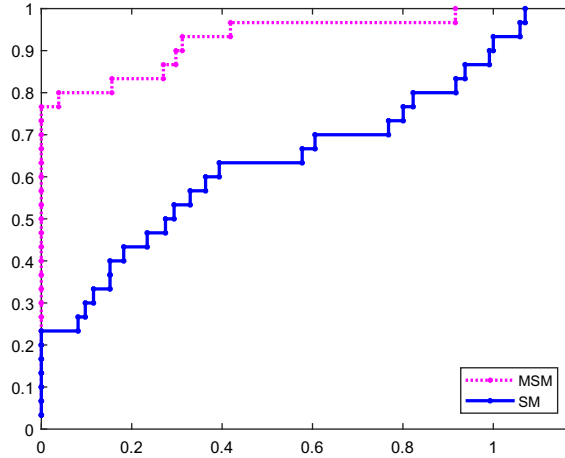


Figure 6. Performance profile based on CPU time.

6. Conclusion

The underlying iterative methods are gradient descent (*GD*) and two its accelerations, called *AGD* and *IGD*. Several classes of iterative methods are defined as two transformations of gradient-descent iterative methods for solving unconstrained optimization are proposed. The first transformation of gradient-descent methods is called *modification* and it is defined by replacing the basic step size t_k in *GD* and *AGD* methods as well as in the *IGD* iterative class by the slightly larger step size $t_k + t_k^2 - t_k^3$. The resulting iterations will be termed as *MGD*, *MAGD* and *MIGD*, respectively. The second transformation of gradient-descent methods is called as *hybridization* is defined as a composition of all considered *GD* methods and modified *GD* methods with the Picard–Mann hybrid iterative process. The resulting hybrid iterations applied on *GD*, *AGD* and *IGD* will be termed as *HGD*, *HAGD* and *HIGD*. Further, hybridization of *MGD*, *MAGD* and *MIGD* produces *HMGD*, *HMAGD* and *HMIGD* iterations. It is proved that defined methods are linearly convergent methods for the uniformly convex functions. Derived methods are numerically tested and compared. Noticeable improvement in favour of the *MSM* method regarding the number of iterations, CPU time and the number of function evaluations is observed. Applied Dolan and Moré’s performance profiles of all methods subject to the number of iterations, the CPU time and the number of function evaluations also confirm the dominance of the *MSM* method.

A summarization of different step sizes used in the methods described in the current research is presented in Table 10 in order to clarify the attributes ‘Multiple Step Size’ in the title. The strike in the table means that the corresponding step size is not in use.

According to Table 10, the method *GD* is the basic method. *Multiple Step-Size (MSS)* methods are *AGD*, *SM*, *ADD*, *ADSS*, *TADSS*, *RGDQN1*. *Accelerated Multiple Step-Size (AMSS)* methods are generated applying the *modification* $\mathcal{M}$ and *hybridization* $\mathcal{H}$ on the *MSS* methods. An arbitrary *AMSS* methods obtained using $\mathcal{M}$ (resp. $\mathcal{H}$) will be termed as $\mathcal{MAMSS}$ (resp. $\mathcal{HAMSS}$). Clearly, some new combinations of known optimization methods with their modifications and hybridizations could be further discovered, for example *MADSS*, *MTADSS*, *HADSS*, *HTADSS*, *HMADSS*, *HMTADSS* and so on.

Table 10. Summary of parameters defining step sizes.

Method	Step sizes		
	First	Second	Third
<i>GD</i>	t_k	–	–
<i>AGD</i>	t_k	θ_k	–
<i>SM</i>	t_k	$(\gamma_k^{SM})^{-1}$	–
<i>ADD</i>	t_k	$(\gamma_k^{ADD})^{-1}$	t_k^2
<i>ADSS</i>	t_k	$(\gamma_k^{ADSS})^{-1}$	l_k
<i>TADSS</i>	t_k	$(\gamma_k^{TADSS})^{-1}$	$1 - t_k$
<i>RGDQN1</i>	t_k	γ_k^{-1}	$\theta_k = \frac{\gamma_k}{t_k \gamma_{k+1}}$
<i>HGD</i>	t_k	$\alpha_k + 1$	–
<i>HSM</i>	t_k	$(\gamma_k^{HSM})^{-1}$	$\alpha_k + 1$
<i>HADD</i>	t_k	$(\alpha_k + 1)(\gamma_k^{HADD})^{-1}$	$(\alpha_k + 1)t_k^2$
<i>MGD</i>	$t_k + t_k^2 - t_k^3$	–	–
<i>MAGD</i>	$t_k + t_k^2 - t_k^3$	θ_k	–
<i>MSM</i>	$t_k + t_k^2 - t_k^3$	$(\gamma_k^{MSM})^{-1}$	–
<i>HMGD</i>	$t_k + t_k^2 - t_k^3$	$\alpha_k + 1$	–
<i>HMAGD</i>	$t_k + t_k^2 - t_k^3$	θ_k	$\alpha_k + 1$
<i>HMSM</i>	$t_k + t_k^2 - t_k^3$	$(\gamma_k^{HMSM})^{-1}$	$\alpha_k + 1$

Unlike to traditional gradient-descent (*GD*) algorithms, which are defined on a single step size, improved gradient-descent (*IGD*) algorithms are based on the usage of two or more parameters which define the step size. Is it necessary to use the product of two or more parameters to define the step size? Why to use a product of scaling parameters if you keep in mind that this product gives again a number? Theoretically, the *GD* method indicates good convergence properties, but it is usually very slow in practical applications. Numerical experiments show that introducing two scaling parameters could be useful and improves the standard *GD* method with respect to all three important criteria, the number of iterations, the CPU time and the number of function evaluations. As a support of this conclusion, it is important that the *AGD* method was compared in [1] with the *GD* method. Numerical experiments evidently show better results in favour to the *AGD* scheme with respect to the classical *GD* scheme. Also, numerical experience in [33] shows a clear advantage of the *SM* method over the *AGD* iterations. Finally, numerical testing in [24–26,34] exactly indicates that some further improvements are always possible.

Possible further research includes several new strategies. Firstly, instead of the diagonal matrix, it is possible to consider appropriately defined positive-definite matrix B_k as a better approximation of the Hessian. Later, it is possible to apply similar strategy with two parameters, where one of the parameters is defined according to the third or further term of Taylor's expansion. Also, continuous-time nonlinear optimization gives a new approach to accelerating parameters, which is based on the scaling parameter and the time interval.

Obtained results motivate further investigations of possible accelerated gradient-descent method and its transformations into corresponding variants of accelerated or hybrid methods.

Acknowledgments

This author gratefully acknowledge support from the Research Project 174013 of the Serbian Ministry of Science.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Predrag S. Stanimirović  <http://orcid.org/0000-0003-0655-3741>

References

- [1] N. Andrei, *An acceleration of gradient descent algorithm with backtracking for unconstrained optimization*, Numer. Algor. 42 (2006), pp. 63–73.
- [2] N. Andrei, *An unconstrained optimization test functions collection*, Adv. Model. Optim. 10 (2008), pp. 147–161.
- [3] N. Andrei, *Relaxed gradient descent and a new gradient descent methods for unconstrained optimization*. Visited August 19, 2018. Available at <https://camo.ici.ro/neculai/newgrad.pdf>.
- [4] J. Barzilai and J.M. Borwein, *Two-point step size gradient method*, IMA J. Numer. Anal. 8 (1988), pp. 141–148.
- [5] C. Brezinski, *A classification of quasi-Newton methods*, Numer. Algor. 33 (2003), pp. 123–135.
- [6] Y.H. Dai, *Alternate step gradient method*, Report AMSS–2001–041, Academy of Mathematics and Systems Sciences, Chinese Academy of Sciences, Beijing, 2001.
- [7] Y.H. Dai and R. Fletcher, *On the asymptotic behaviour of some new gradient methods*, Numerical Analysis Report, NA/212, Dept. of Math. University of Dundee, Scotland, UK, 2003.
- [8] Y.H. Dai, J.Y. Yuan and Y. Yuan, *Modified two-point step-size gradient methods for unconstrained optimization*, Comput. Optim. Appl. 22 (2002), pp. 103–109.
- [9] Y.H. Dai and Y. Yuan, *Alternate minimization gradient method*, IMA J. Numer. Anal. 23 (2003), pp. 377–393.
- [10] Y.H. Dai and Y. Yuan, *Analysis of monotone gradient methods*, J. Ind. Manag. Optim. 1 (2005), pp. 181–192.
- [11] Y.H. Dai and H. Zhang, *An adaptive two-point step-size gradient method*, Research report, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, Beijing, 2001.
- [12] S.S. Djordjević, *Two modifications of the method of the multiplicative parameters in descent gradient methods*, Appl. Math. Comput. 218 (2012), pp. 8672–8683.
- [13] E.D. Dolan and J.J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program. 91 (2002), pp. 201–213.
- [14] S. Ishikawa, *Fixed points by a new iteration method*, Proc. Am. Math. Soc. 44 (1974), pp. 147–150.
- [15] S.H. Khan, *A Picard-Mann hybrid iterative process*, Fixed Point Theory Appl. 2013 (2013), p. 69. Springer Open Journal 2013.
- [16] N. Kontrec and M. Petrović, *Implementation of gradient methods for optimization of underage costs in aviation industry*, Univ. Thought Publ. Nat. Sci. 6 (2016), pp. 71–74.
- [17] W.R. Mann, *Mean value methods in iterations*, Proc. Am. Math. Soc. 4 (1953), pp. 506–510.
- [18] M. Miladinović, P.S. Stanimirović and S. Miljković, *Scalar correction method for solving large scale unconstrained minimization problems*, J. Optim. Theory Appl. 151 (2011), pp. 304–320.
- [19] J. Nocedal and S.J. Wright, *Numerical Optimization*, Springer-Verlag, New York, 1999.
- [20] J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equation in Several Variables*, Academic Press, New York, 1970.

- [21] S. Panić, M.J. Petrović and M. Mihajlov-Carević, *Initial improvement of the hybrid accelerated gradient descent process*, Bull. Aust. Math. Soc. 98 (2018), pp. 331–338.
- [22] M.J. Petrović, *An accelerated double step size method in unconstrained optimization*, Applied Math. Comput. 250 (2015), pp. 309–319.
- [23] M.J. Petrović and P.S. Stanimirović, *Accelerated double direction method for solving unconstrained optimization problems*, Math. Probl. Eng. 2014 (2014), article ID 965104, 8 pages.
- [24] M.J. Petrovic, P.S. Stanimirovic, N. Kontrec and J. Mladenovic, *Hybrid modification of accelerated double direction method*, Math. Probl. Eng. 2018 (2018), Article ID 1523267, 8 pages, <https://doi.org/10.1155/2018/1523267>.
- [25] M.J. Petrović, V. Rakocević, N. Kontrec, S. Panić and D. Ilić, *Hybridization of accelerated gradient descent method*, Numer. Algor. 79 (2018), pp. 769–786.
- [26] M. Petrović, N. Kontrec and S. Panić, *Determination of accelerated factors in gradient decent iterations based on Taylor's series*, Univ. Thought Publ. Nat. Sci. 7 (2017), pp. 41–45.
- [27] E. Picard, *Memoire sur la theorie des equations aux derivees partielles et la methode des approximations successives*, J. Math. Pures Appl. 6 (1890), pp. 145–210.
- [28] M. Raydan and B.F. Svaiter, *Relaxed steepest descent and Cauchy-Barzilai-Borwein method*, Comput. Optim. Appl. 21 (2002), pp. 155–167.
- [29] M. Raydan, *On the Barzilai and Borwein choice of steplength for the gradient method*, IMA J. Numer. Anal. 13 (1993), pp. 321–326.
- [30] M. Raydan, *The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem*, SIAM J. Optim. 7 (1997), pp. 26–33.
- [31] R.T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, 1970.
- [32] Z.-J. Shi, *Convergence of line search methods for unconstrained optimization*, Appl. Math. Comput. 157 (2004), pp. 393–405.
- [33] P.S. Stanimirovic and M.B. Miladinovic, *Accelerated gradient descent methods with line search*, Numer. Algor. 54 (2010), pp. 503–520.
- [34] P.S. Stanimirović, G.V. Milovanović and M.J. Petrović, *A transformation of accelerated double step size method for unconstrained optimization*, Math. Probl. Eng. 2015 (2015), article ID 283679, 8 pages.
- [35] W. Sun and Y.-X. Yuan, *Optimization Theory and Methods: Nonlinear Programming*, Springer, Berlin, 2006.
- [36] M.N. Vrahatis, G.S. Androulakis, J.N. Lambrinos and G.D. Magoulas, *A class of gradient unconstrained minimization algorithms with adaptive step-size*, J. Comp. Appl. Math. 114 (2000), pp. 367–386.
- [37] Y. Yuan, *A new stepsize for the steepest descent method*, J. Comput. Math. 24 (2006), pp. 149–156.