mathematics

MDPI

Article

An Improved Modification of Accelerated Double Direction and Double Step-Size Optimization Schemes

Milena J. Petrović [1,*], Dragana Valjarević [1], Dejan Ilić [2], Aleksandar Valjarević [3] and Julija Mladenović [4]

[1] Faculty of Sciences and Mathematics, University of Pristina in Kosovska Mitrovica, Lole Ribara 29, 38220 Kosovska Mitrovica, Serbia; dragana.valjarevic@pr.ac.rs
[2] Faculty of Sciences and Mathematics, University of Niš, Višegradska 33, 18106 Niš, Serbia; ilicde@pmf.ni.ac.rs
[3] Faculty of Geography, University of Belgrade, Studentski Trg 3/III, 11000 Belgrade, Serbia; aleksandar.valjarevic@gef.bg.ac.rs
[4] Faculty of Mathematics, University of Belgrade, Studentski Trg 16, 11000 Belgrade, Serbia; pd202025@alas.matf.bg.ac.rs
* Correspondence: milena.petrovic@pr.ac.rs; Tel.: +381-28-425-396

Abstract: We propose an improved variant of the accelerated gradient optimization models for solving unconstrained minimization problems. Merging the positive features of either double direction, as well as double step size accelerated gradient models, we define an iterative method of a simpler form which is generally more effective. Performed convergence analysis shows that the defined iterative method is at least linearly convergent for uniformly convex and strictly convex functions. Numerical test results confirm the efficiency of the developed model regarding the CPU time, the number of iterations and the number of function evaluations metrics.

Keywords: gradient descent; line search; gradient descent methods; quasi-Newton method; convergence rate

Citation: Petrović, M.J.; Valjarević, D.; Ilić, D.; Valjarević, A.; Mladenović, J. An Improved Modification of Accelerated Double Direction and Double Step-Size Optimization Schemes. *Mathematics* **2022**, *10*, 259. https://doi.org/10.3390/math10020259

Academic Editor: Armin Fügenschuh

Received: 20 November 2021
Accepted: 13 January 2022
Published: 15 January 2022

1. Accelerated Double Direction and Double Step Size Methods Overview

In order to define an efficient optimization model for solving unconstrained nonlinear tasks, we approach the matter on multiple fronts. One of the primers is insuring a fast convergence, desirably close enough to the Newton method's convergence rate. On the other hand, we would like to avoid eventual complicated calculations that can arise from deriving Hessians' second order partial derivatives. That is why the quasi-Newton method is a good starting point in developing an optimization method with good performance profiles. The benefits of the quasi-Newton methods are well known. One of the main characteristics of these iterations is the conservation of good convergence features, although the Hessian, i.e., the Hessian's inverse, is not explicitly used. Instead, the appropriately defined Hessian's approximation, or the approximation of its inverse is used in these methods. This way, the quasi-Newton methods preserve a good convergence rate and, at same time, avoid the possible difficulties of Hessians' calculations. In this paper, we are using a quasi-Newton concept to define an efficient minimization scheme for solving unconstrained minimization problems, assigned as:

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

where $f(x)$ is an objective function.

When defining an optimization iterative models based on the quasi-Newton form, we can start with the following general iteration:

$$x_{k+1} = x_k + t_k d_k, \tag{2}$$

where x_k stands for a current iterative point, x_{k+1} is the next one, t_k is the iterative step length and d_k is the search direction of the $k-$th iteration. For iterations of the quasi-Newton type, the search direction is defined trough the gradient features. Therewith, an iterative direction vector has to fulfill the descent condition, i.e.,

$$g_k^T d_k \leq 0. \tag{3}$$

In condition (3), by g_k, we denote the gradient of the objective function at x_k. Furthermore, we adopt the usual notations:

$$g(x) = \bigtriangledown f(x), \quad G(x) = \bigtriangledown^2 f(x), \quad g_k = \bigtriangledown f(x_k), \quad G_k = \bigtriangledown^2 f(x_k), \tag{4}$$

where $\bigtriangledown f(x)$ and $\bigtriangledown^2 f(x)$ are the standard notations for the gradient and the Hessian of the goal function, respectively.

The way of defining the iterative step length t_k and the iterative search direction vector d_k directly influences the methods' efficiency. With that, some authors [1–5] segregated one parameter more, equally important as the other two, that contributes to the method's performance characteristics. That is an iterative accelerated parameter, often marked out as γ_k. In [1], the author marked this parameter as θ_k, and its iterative value is expressed by the relation (5). Researchers on this topic justifiably extricated a class of accelerated gradient schemes. In [3], for example, authors numerically confirmed more than evident performance progress in favor of the accelerated method when compared to its non-accelerated version. Here are some expressions of the accelerated factors defined in the accelerated gradient models mentioned above. These accelerated parameters are also listed in [6]:

$$\theta_k^{AGD} = -\frac{t_k g_k^T g_k}{t_k y_k^T g_k}, \tag{5}$$

$$\gamma_{k+1}^{SM} = 2\gamma_k \frac{\gamma_k [f(x_{k+1}) - f(x_k)] + t_k \|g_k\|^2}{t_k^2 \|g_k\|^2}, \tag{6}$$

$$\gamma_{k+1}^{ADD} = 2\frac{f(x_{k+1}) - f(x_k) - \alpha_k g_k^T \left(\alpha_k d_k - \gamma_k^{-1} g_k\right)}{\left(\alpha_k d_k - \gamma_k^{-1} g_k\right)^T \left(t_k d_k - \gamma_k^{-1} g_k\right)}, \tag{7}$$

$$\gamma_{k+1}^{ADSS} = 2\frac{f(x_{k+1}) - f(x_k) + \left(\alpha_k {\gamma_k}^{-1} + \beta_k\right) \|g_k\|^2}{\left(\alpha_k {\gamma_k}^{-1} + \beta_k\right)^2 \|g_k\|^2}, \tag{8}$$

$$\gamma_{k+1}^{TADSS} = 2\frac{f(x_{k+1}) - f(x_k) + \psi_k \|g_k\|^2}{\psi_k^2 \|g_k\|^2}, \quad \psi_k = [\alpha_k \gamma_k^{-1} - \alpha_k^2) + 1]. \tag{9}$$

Interesting ideas of the double step length and the double direction approach in defining an efficient minimization iteration are presented in [2,3]. In both of these studies, the authors used properly determined accelerating characteristics. In this paper, we use the proven good properties of each of these models, i.e., of the accelerated double direction, or shortly, the ADD method, as well as of the accelerated double step size-ADSS method.

The ADD iteration is defined by the following expression:

$$x_{k+1} = x_k + \alpha_k^2 d_k - \alpha_k {\gamma_k}^{-1} g_k, \tag{10}$$

where $\gamma_k = \gamma_k^{ADD} > 0$ is the acceleration parameter. The iterative step length α_k is derived using the Armijos' Backtracking inexact lines search algorithm. Variable d_k stands for the second vector direction, and it is calculated by the next rule:

$$d_k(t) = \begin{cases} d_k^*, & k \leq m-1 \\ \sum_{i=2}^{m} t^{i-1} d_{k-i+1}^*, & k \geq m \end{cases} \tag{11}$$

where d_k^* is the solution of the problem $\min_{x\in\mathbb{R}} \Phi_k(d)$,

$$\Phi_k(d) = \bigtriangledown f(x_k)^T d + \frac{1}{2}\gamma_{k+1} I = g(x_k)^T d + \frac{1}{2}\gamma_{k+1} I.$$

The two search directions in the ADD method are d_k, defined by the previous rule and $-\gamma_k^{-1} g_k$. One of the main results in [3] is that the ADD algorithm provides a lower number of iterations than the accelerated gradient descent method, marked as the SM method, which is presented in [2]. The iterative form of the SM method is given by the expression:

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

where t_k is the iterative step length value, and $\gamma_k \equiv \gamma_k^{SM}$ is the acceleration parameter of the SM iteration expressed by the relation (6).

The accelerated double step size model, i.e., the ADSS, is defined as

$$x_{k+1} = x_k - \alpha_k {\gamma_k}^{-1} g_k - \beta_k g_k = x_k - \left(\alpha_k \gamma_k^{-1} + \beta_k\right) g_k. \tag{12}$$

Parameters $\alpha_k > 0$ and $\beta_k > 0$ are two iterative step lengths, calculated by two different Backtracking procedures, and $\gamma_k = \gamma_k^{ADSS} > 0$ is the ADSS iterative accelerated parameter. In the ADSS iteration, we can identify the vector direction as:

$$-\left(\alpha_k \gamma_k^{-1} + \beta_k\right) g_k. \tag{13}$$

Transformed ADSS method, or in short, the TADSS, came from the ADSS scheme under the following condition: $\alpha_k + \beta_k = 1$. The TADSS iteration is defined as:

$$x_{k+1} = x_k - [\alpha_k({\gamma_k}^{-1} - 1) + 1] g_k. \tag{14}$$

From expression (13), we conclude that the defined vector direction has the form of a negative gradient direction. Having that in mind, it depends on the step length parameters as well as on the accelerated parameter iterative value. Numerical experiments from [4] show that the ADSS iteration outperforms the ADD [3] and the SM [2] schemes regarding all three of the analyzed metrics: the number of iterations, CPU time and the number of function evaluations.

We are motivated to define the method as an improved merged version of the accelerated double direction and double step size methods. At the same time, the proposed model should be of the simpler form than the ADD and the ADSS schemes are. We define this simpler form by ejecting one of the Backtracking algorithms from the ADSS iteration and by replacing the algorithm (11) in the ADD scheme with the gradient descent rule. Taking all these assumptions, we expect the proposed iterative method to be convergent at least at the same rate as the ADD and the ADSS methods are. That modified iteration, based on the mentioned accelerated gradient descent algorithms, should conserve the positive sides of its predecessors but also exceed them regarding the performance profiles of all tested metrics.

The paper is organized in the following way: In Section 2, we define the improved version of the ADD and the ADSS schemes. The convergence analysis of the defined model is carried out in Section 3. Numerical test results are compared, analyzed and displayed in Section 4.

2. Modified Accelerated Double Direction and Double Step Size Method

Taking into account the iterative form of the accelerated ADD method as well as good performance features of the accelerated double step size ADSS scheme, considering all three tested metrics, we propose the following iterative model for solving a large scale of unconstrained minimization problems:

$$x_{k+1} = x_k - \left(\alpha_k {\gamma_k}^{-1} + \alpha_k^2\right) g_k \equiv x_k - \alpha_k {\gamma_k}^{-1} g_k - \alpha_k^2 g_k. \tag{15}$$

Iterative scheme (15) presents the merged variant of the ADD and the ADSS methods, keeping the favorable aspects of each included gradient scheme. We denoted the iterative rule (15) as the *modified accelerated double direction and double step size method*, or in short, *modADS*. In the modADS scheme, one iterative search direction is ${\gamma_k}^{-1} g_k$, and the other is simply a negative gradient direction. Two step lengths, α_k and α_k^2, are obtained using one Backtracking procedure. Basically, our main goal in generating the modADS method is to define an improved merged version of the accelerated double direction and double step size methods. Having that in mind, we want to conserve the positive aspects of each of these two baseline models. The form of the ADD iteration contains only one iterative step length value, i.e., one Backtracking procedure is applied. That was the main motivation to substitute the second iterative value β_k from the ADSS iteration with the α_k^2. In this way, we conserve the form of the ADD iteration in the new modADS scheme.

On the other hand, from the results presented in [4], we know that the second search direction d_k defined in the ADD iteration by (11) causes an increase in the number of function evaluations. Therefore, instead of it, just like in the ADSS iteration, in the new modADS process we simply use the gradient descent direction for the second search direction, as well.

There are certainly many different options for defining the second iterative step length in the double-direction and double step size models that differ from our choice: α_k^2 . That question is still open. Since the modADS belongs to the class of accelerated double direction and double step size methods and presents a merged form of the ADD and the ADSS iteration, the choice to keep α_k^2 as the second step length value was a natural one. Additionally, according to the TADSS iteration (14), it could be said that the TADSS corresponds to a different choice of second step size β_k of the ADSS iteration. Therefore, this is also a motivation to define the modADS in a presented way and to compare the performance features of these two similar approaches.

So, the common elements of the ADD, the ADSS and the proposed modADSS iterative form represent the iterative step length value, α_k, and the search direction vector ${\gamma_k}^{-1} g_k$. The other search direction in the modADS is $-g_k$, just like in the ADSS scheme. Still, as previously explained, the second step-size value of the new method differs from the one, β_k, applied in the ADSS model. Instead of using an additional inexact line search technique to calculate the second iterative step length value, in the modADS, we use only one Backtracking procedure and define the second step length parameter as the quadratic value of the Backtracking outcome α_k. This way, we evidently provide a decrease in the computational time, number of needed iterations and function evaluations. We confirm this statement in Section 4 by comparative analysis of the performance profiles of each of the tested models.

The algorithm of the Backtracking procedure upon which we calculate the iterative step length value is given by the following steps:

1. Objective function $f(x)$, the direction d_k of the search at the point x_k and numbers $0 < \sigma < 0.5$ and $\beta \in (0, 1)$ are required;
2. $\alpha = 1$;
3. $f(x_k + \alpha d_k) > f(x_k) + \sigma \alpha g_k^T d_k$, take $\alpha := \alpha\beta$;
4. Return $\alpha_k = \alpha$.

We now derive the iterative value of the acceleration parameter using the second order Taylors' expansion of the modADS iteration (15). To avoid huge expressions in that process, we simplified the relation (15) using the next substitution:

$$x_{k+1} = x_k - s_k g_k, \tag{16}$$

where $s_k = \alpha_k {\gamma_k}^{-1} + \alpha_k^2 = \alpha_k\left({\gamma_k}^{-1} + \alpha_k\right)$. Second order Taylor polynomial of (16) is then:

$$f(x_{k+1}) \approx f(x_k) - g_k^T s_k g_k + \frac{1}{2} s_k g_k^T \nabla^2 f(\xi) g_k. \tag{17}$$

In relation (17), $\nabla^2 f(\xi)$ stands for the Hessian of the objective function, and variable ξ fulfills the following conditions:

$$\xi \in [x_k, x_{k+1}], \quad \xi = x_k + \delta(x_{k+1} - x_k) = x_k - \delta s_k g_k, \quad 0 \leq \delta \leq 1.$$

We replace Hessian $\nabla^2 f(\xi)$ with a properly defined scalar diagonal matrix

$$\gamma_k I,$$

where variable γ_{k+1} is the acceleration parameter we are searching for:

$$f(x_{k+1}) \approx f(x_k) - s_k \|g_k\|^2 + \frac{1}{2} s_k \gamma_{k+1} \|g_k\|^2. \tag{18}$$

From the previous expression, we can easily compute the iterative value of the acceleration factor:

$$\gamma_{k+1} = 2\frac{f(x_{k+1}) - f(x_k) + s_k\|g_k\|^2}{s_k^2\|g_k\|^2} = 2\frac{f(x_{k+1}) - f(x_k) + \alpha_k\left(\gamma_k^{-1} + \alpha_k\right)\|g_k\|^2}{\alpha_k^2\left(\gamma_k^{-1} + \alpha_k\right)^2\|g_k\|^2}. \tag{19}$$

We are only interested in the positive γ_{k+1} values because, in that case, both of the second order necessary and the second order sufficient conditions are fulfilled. However, if in some iterative steps we calculate a negative value for the acceleration parameter, then we simply set $\gamma_{k+1} = 1$. This choice of γ_{k+1} transforms our modADS iteration into the standard gradient descent iterative method, i.e.,

$$x_{k+2} = x_{k+1} - \alpha_{k+1}(1 + \alpha_{k+1}) g_{k+1} \equiv x_{k+1} - t_{k+1} g_{k+1},$$

for some $t_{k+1} = \alpha_{k+1}(1 + \alpha_{k+1})$.

For initial values $0 < \rho < 1$, $0 < \tau < 1$, x_0, $\gamma_0 = 1$, we now present the modADS algorithm:

1. Set $k = 0$, compute $f(x_0)$, g_0 and take $\gamma_0 = 1$;
2. If $\|g_k\| < \epsilon$, then go to Step 8, else continue by the step 3;
3. Apply Backtracking algorithm to calculate the iterative step length α_k;
4. Compute x_{k+1} using (15);
5. Determine the acceleration parameter γ_{k+1} using (19);
6. If $\gamma_{k+1} < 0$, then take $\gamma_{k+1} = 1$;
7. Set $k := k + 1$, go to Step 2;
8. Return x_{k+1} and $f(x_{k+1})$.

3. Convergence Analysis

In this section, we prove that the modADS iteration linearly converges on the sets of uniformly convex functions and strictly convex quadratic functions. We analyze these two function sets separately.

3.1. Set of Uniformly Convex Functions

To prove the linear convergence properties, we are using the following two statements from [7,8]:

Proposition 1. *If the function $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable and uniformly convex on $\mathbb{R}^n$ then:*

(1) *the function f has a lower bound on $L_0 = \{x \in \mathbb{R}^n \mid f(x) \leq f(x_0)\}$, where $x_0 \in \mathbb{R}^n$ is available;*
(2) *the gradient g is the Lipschitz continuous in an open convex set B which contains L_0, i.e., there exists $L > 0$ such that:*

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

Lemma 1. *Under the assumptions of Proposition 1, there exist real numbers m and M satisfying:*

$$0 < m \leq 1 \leq M, \tag{20}$$

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

$$m\|y\|^2 \leq y^T \nabla^2 f(x) y \leq M\|y\|^2, \quad \forall\, x, y \in \mathbb{R}^n; \tag{21}$$

$$\frac{1}{2}m\|x - x^*\|^2 \leq f(x) - f(x^*) \leq \frac{1}{2}M\|x - x^*\|^2, \quad \forall\, x \in \mathbb{R}^n; \tag{22}$$

$$m\|x - y\|^2 \leq (g(x) - g(y))^T (x - y) \leq M\|x - y\|^2, \quad \forall\, x, y \in \mathbb{R}^n. \tag{23}$$

In the following Lemma, we show that the objective function, on which the modADS iteration is applied, is bounded below. We also estimate the measure of the iterative function decreasing. The proof is analogous as in [2].

Lemma 2. *Let the sequence $\{x_k\}$ be defined by the (15), and let f be uniformly convex function. Then:*

$$f(x_k) - f(x_{k+1}) \geq \mu\|g_k\|^2, \tag{24}$$

for

$$\mu = \min\left\{\frac{\sigma}{M}, \frac{\sigma(1-\sigma)}{L}\beta\right\}, \tag{25}$$

where $L > 0$ is the Lipschitz constant from Proposition 1, and $M \in \mathbb{R}$ is defined in Lemma 1.

The fact that the modADS model converges at least linearly is proved in the next Theorem 1.

Theorem 1. *The sequence $\{x_k\}$, defined by the (15) and applied on uniformly convex and twice differentiable objective function f, converges linearly to its solution x^* and*

$$\lim_{k\to\infty} \|g_k\| = 0. \tag{26}$$

Proof. From Lemma 2, we know that the objective function f, when applied on the modADS process, is bounded below and decreases, so it is evident that:

$$\lim_{k\to\infty} (f(x_k) - f(x_{k+1})) = 0. \tag{27}$$

This equality, merged with the result of Lemma 2, i.e., the relation (24), lead us to the following conclusion:

$$\lim_{k\to\infty} \|g_k\| = 0. \tag{28}$$

Let us prove now that the sequence $\{x_k\}$, generated by the (15), converges to its solution x^*, i.e.,

$$\lim_{k\to\infty} \|x_k - x^*\| = 0. \tag{29}$$

To prove (29), we put $x^* \equiv y$ in (23):

$$m\|x - x^*\|^2 \leq (g(x) - g(x^*))^T (x - x^*) \leq M\|x - x^*\|^2.$$

Regarding the Mean Value Theorem and the Cauchy–Schwartz inequality, further on we obtain:

$$m\|x - x^*\|^2 \leq \|g(x)\| \leq M\|x - x^*\|^2. \tag{30}$$

From (24) and (30), we have the following estimations:

$$\begin{aligned}\mu\|g_k\|^2 &\geq \mu m^2\|x - x^*\|^2 \\ &\geq 2 \cdot \mu\frac{m^2}{M}(f(x_k) - f(x^*)) \to_{k\to\infty} 0,\end{aligned}$$

which confirms (29).

To complete this proof, at the end, we show that the modADS process is linearly convergent. To do this, we practically need to prove that

$$\rho \equiv \sqrt{2 \cdot \mu\frac{m^2}{M}} < 1.$$

We know from Lemma 2 that there are two values of the variable μ : $\mu = \frac{\sigma}{M}$ and $\mu = \frac{\sigma(1-\sigma)\beta}{L}$:

1. $\mu = \frac{\sigma}{M}$: In this case, we have:

$$\rho^2 = 2\mu\frac{m^2}{M} = 2 \cdot \frac{\sigma}{M}\frac{m^2}{M} = 2\frac{\sigma}{M}\frac{m^2}{M} \leq 2\sigma\frac{m^2}{M} \leq 2\sigma < 1,$$

since $\sigma \in (0, \frac{1}{2})$ and $m < M$.

2. $\mu = \frac{\sigma(1-\sigma)\beta}{L}$: For this $\mu-$value, using the inequality $m \leq L$, we show the same

$$\rho^2 = 2\mu\frac{m^2}{M} = 2 \cdot \frac{\beta\sigma(1-\sigma)}{L}\frac{m^2}{M} < 2 \cdot \frac{1}{2} \cdot 1 \cdot \frac{m^2}{L \cdot M} = \frac{m^2}{L \cdot M} \leq \frac{L \cdot m}{L \cdot M} = \frac{m}{M} < 1,$$

which completes this proof. □

3.2. Set of Strictly Convex Quadratics

Now, let us suppose that the objective function is a strictly convex quadratic function, expressed as:

$$f(x) = \frac{1}{2}x^T Ax - b^T x, \tag{31}$$

where A is a real $n \times n$ matrix, which is symmetric and positive definite, and $b \in \mathbb{R}^n$ is a given vector. Lets denote and sort the eigenvalues of the matrix A as

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

Our goal now is to prove the convergence of the modADS iteration when applied on strictly convex quadratic. However, before we reveal the main theorem of this subsection, we show one auxiliary lemma which estimates the iterative variable $s_k \equiv \alpha_k(\gamma_k^{-1} + \alpha_k)$ with respect to the smallest and the largest eigenvalues of matrix A.

Lemma 3. *The smallest and the largest eigenvalues of the matrix A satisfy inequalities:*

$$\frac{\sigma}{2\lambda_n} \leq \alpha_{k+1}(\gamma_{k+1}^{-1} + \alpha_{k+1}) \leq \frac{1}{\lambda_1} + 1, \tag{32}$$

where γ_{k+1} and α_{k+1} are the iterative acceleration parameter and step length value of the modADS iteration, respectively.

Proof. For the strictly convex quadratic function (31), the difference of its values in two successive points is:

$$\begin{aligned} f(x_{k+1}) - f(x_k) &= \frac{1}{2}x_{k+1}^T A x_{k+1} - b^T x_{k+1} - \frac{1}{2}x_k^T A x_k + b^T x_k \\ &= \frac{1}{2}(x_k - s_k g_k)^T A(x_k - s_k g_k) - b^T(x_k - s_k g_k) - \frac{1}{2}x_k^T A x_k + b^T x_k \\ &= \frac{1}{2}x_k^T A x_k - \frac{1}{2}s_k x_k^T A g_k - \frac{1}{2}s_k g_k^T A x_k \\ &\quad + \frac{1}{2}s_k^2 g_k^T A g_k - b^T x_k + s_k b^T g_k - \frac{1}{2}x_k^T A x_k + b^T x_k \\ &= -\frac{1}{2}s_k x_k^T A g_k - \frac{1}{2}s_k g_k^T A x_k + \frac{1}{2}s_k^2 g_k^T A g_k + s_k b^T g_k, \end{aligned}$$

i.e.,

$$f(x_{k+1}) - f(x_k) = -\frac{1}{2}s_k x_k^T A g_k - \frac{1}{2}s_k g_k^T A x_k + \frac{1}{2}s_k^2 g_k^T A g_k + s_k b^T g_k. \tag{33}$$

Matrix A is symmetric and positive definite, so we can apply the symmetry condition: $b^T g_k = g^T b_k$. We can also use the fact that the gradient of the function (31) is $g_k = Ax_k - b$ and transform (33) into:

$$\begin{aligned} f(x_{k+1}) - f(x_k) &= -\frac{1}{2}s_k\left(g_k^T A x_k + x_k^T A g_k - s_k g_k^T A g_k - b^T g_k - b^T g_k\right) \\ &= -\frac{1}{2}s_k\left(g_k^T(Ax_k - b^T) + g_k^T(Ax_k - b^T) - s_k g_k^T A g_k\right) \\ &= -\frac{1}{2}s_k\left(g_k^T g_k + g_k^T g_k - s_k g_k^T A g_k\right) \\ &= -s_k g_k^T g_k + \frac{1}{2}s_k^2 g_k^T A g_k. \end{aligned}$$

If we replace the derived expression of the difference between function values in two successive iterations into the (19), we obtain:

$$\gamma_{k+1} = 2\frac{-s_k g_k^T g_k + \frac{1}{2}s_k^2 g_k^T A g_k + s_k g_k^T g_k}{s_k^2 g_k^T g_k} \equiv \frac{g_k^T A g_k}{g_k^T g_k}. \tag{34}$$

From (34), we conclude that γ_{k+1} is the Rayleigh quotient of the real symmetric matrix at the gradient vector g_k, so the next is true:

$$\lambda_1 \leq \gamma_{k+1} \leq \lambda_n, \quad k \in \mathbb{N}. \tag{35}$$

Since $0 \leq \alpha_{k+1} \leq 1$, the following estimations are valid:

$$\begin{aligned} s_{k+1} &= \alpha_{k+1}(\gamma_{k+1}^{-1} + \alpha_{k+1}) = \alpha_{k+1}\gamma_{k+1}^{-1} + \alpha_{k+1}^2 \\ &\leq \frac{1}{\gamma_{k+1}} + \alpha_{k+1} \leq \frac{1}{\lambda_1} + \alpha_{k+1} \leq \frac{1}{\lambda_1} + 1 \end{aligned}$$

To prove the right side of (32), we will take the relation $t_k > \frac{\eta(1-\sigma)\gamma_k}{L}$, proved in [2]. With proper notation used in this scheme, the previous inequality becomes:

$$\alpha_k > \frac{\beta(1-\sigma)\gamma_k}{L}. \tag{36}$$

We take into account the parameter limitations, i.e., $\sigma \in (0, \frac{1}{2}), \quad \beta \in (\sigma, 1)$ and $0 \leq \alpha_{k+1} \leq 1$, and that leads us to:

$$\begin{aligned} s_{k+1} &= \alpha_{k+1}(\gamma_{k+1}^{-1} + \alpha_{k+1}) = \alpha_{k+1}\gamma_{k+1}^{-1} + \alpha_{k+1}^2 \\ &> \frac{\alpha_{k+1}}{\gamma_{k+1}} \geq \frac{\beta(1-\sigma)\gamma_{k+1}}{L} \cdot \frac{1}{\gamma_{k+1}} \\ &\geq \frac{\beta(1-\sigma)}{L} \geq \frac{\sigma\left(1-\frac{1}{2}\right)}{L} \\ &= \frac{\sigma}{2L} \geq \frac{\sigma}{2\lambda_n}. \end{aligned}$$

The last inequality arises from the fact that the largest eigenvalue λ_n has the property of the Lipschity constant L:

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

This analysis confirms that (32) is truly assured. □

Theorem 2. *Suppose the relation $\lambda_n < 2\frac{2\lambda_1}{1+\lambda_1}$ holds for the smallest and the largest eigenvalues of the strictly convex quadratic function (31). Then, considering the modADS iteration applied on (31), the following holds:*

$$g_k = \sum_{i=1}^{n} d_i^k v_i, \tag{37}$$

where

$$(d_i^{k+1})^2 \leq \delta^2 (d_i^k)^2, \quad \delta = \max\left\{1 - \frac{\lambda_1}{2\lambda_n}, \lambda_n\left(\frac{1}{\lambda_1} + 1\right) - 1\right\}, \tag{38}$$

for some real parameters $d_1^k, d_2^k, \ldots, d_n^k$. With that:

$$\lim_{k\to\infty} \|g_k\| = 0. \tag{39}$$

Proof. Let $\{v_1, v_2, \ldots, v_n\}$ be the set of orthonormal eigenvalues of matrix A in expression (31). Assume that the sequence $\{x_k\}$ is generated by iterative rule (15). Then, the gradient of the function (31) in $k+1-$th iterative point is:

$$g_{k+1} = A(x_k - s_k g_k) - b = Ax_k - b - s_k A g_k = g_k - s_k A g_k = (I - s_k A) g_k, \tag{40}$$

since $g_k = Ax_k - b$. Applying (37), we obtain:

$$g_{k+1} = \sum_{i=1}^{n} d_i^{k+1} v_i = \sum_{i=1}^{n} (1 - s_k\lambda_i) d_i^k v_i.$$

To prove (37), it is enough to show that $\mid 1 - s_k\lambda_i \mid \leq \delta$.

$$\mid 1 - s_k\lambda_i \mid \leq \delta \Leftrightarrow \begin{cases} 1 - s_k\lambda_i & s_k\lambda_i \leq 1 \\ s_k\lambda_i - 1 & s_k\lambda_i > 1, \end{cases} \tag{41}$$

so, we analyze two cases:

1. $1 \geq s_k\lambda_i \geq \frac{\lambda_1}{2\lambda_n} \Rightarrow 1 - s_k\lambda_i \leq 1 - \frac{\lambda_1}{2\lambda_n} \leq \delta$;
2. $1 < s_k\lambda_i \leq \lambda_n\left(\frac{1}{\lambda_1} + 1\right) \Rightarrow \lambda_n\left(\frac{1}{\lambda_1} + 1\right) - 1 < \delta$.

From (37), we have that the measure of the gradient norm square is:

$$\|g_k\|^2 = \sum_{i=1}^{n} (d_i^k)^2, \tag{42}$$

and since parameter $\delta \in (0,1)$, we derive the final conclusion (39). □

4. Numerical Outcomes and Comparative Analysis

In this section, we display the numerical results, using which we compare the relevant methods. For comparative models, in addition to the objective modADSS method presented in this paper, we primarily chose the accelerated double direction (ADD) method introduced in [3] and the accelerated double step-size (ADSS) iteration from [4]. This is a natural choice of comparative optimization processes since the derived modADS algorithm originates from these two gradient accelerated schemes and our basic goal is the improvement of this class of methods. Then, we investigate the impact of Backtracking parameter β by testing two more values for this parameter. The TADSS method, presented in [5], and the modADS introduced in this paper present two different ways of reducing the double step-size ADSS scheme into a single step length iteration. Due to this fact, we compare these two methods as well. Finally, we complete the numerical comparative analysis by comparing the defined modADS model with two more general gradient descent methods: Cauchy's gradient method (GD) and Andrei's accelerated gradient method (AGD) from [1].

The ADD scheme brought benefits regarding the reduction in the needed number of iterations towards its non-linear version and the SM method from [2]. Furthermore, in [4], the ADSS shown undisputed advances with respect to all three of the tested metrics: the number of iterations, the CPU time and the number of function evaluations. It has been compared with the SM and the ADD schemes.

All codes are written in the visual C++ programming language and run on a Workstation Intel(R) Core(TM) 2.3 GHz. The following values of the Backtracking parameters are taken: $\sigma = 0.0001$ and $\beta = 0.8$.

The stopping criteria are:

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

We chose 10 values for the number of parameters for each test function: 100; 500; 1000; 3000; 5000; 10,000; 15,000; 20,000; 25,000 and 30,000. As a final result for 1 test function, we sum all 10 outcomes. We measured all three performance characteristics: the number of iterations, CPU and the number of evaluations. If for a certain number of iterations and for some test functions the applied model does not finish the test process in some defined time, we put the constant t_e, the time-limiter parameter, in Tables 1 and 2.

Table 1. Number of iterations, modADS, ADD and ADSS.

Function Number	modADS	ADD	ADSS
1.	50	73	50
2.	432	82	432
3.	31	88	31
4.	60	83	135
5.	41	82	44
6.	80	110	76
7.	70	120	70
8.	40	100	40
9.	783	100	781
10.	70	100	70
11.	428	91	428
12.	470	84	470
13.	60	91	60
14.	61	81	61
15.	60	85	60
16.	40	100	40

Table 1. *Cont.*

Function Number	modADS	ADD	ADSS
17.	80	111	80
18.	60	89	60
19.	432	82	432
20.	70	100	70
21.	10	10	10
22.	76	102	70
23.	2202	$t_e > l$	2203
24.	2215	$t_e > l$	2215
25.	32	80	34
26.	235	131	235
27.	10	10	10
28.	10	10	10
29.	3870	$t_e > l$	5083
30.	2061	$t_e > l$	$t_e > l$

Table 2. Number of function evaluations, modADSS, ADD and ADSS.

Function Number	modADS	ADD	ADSS
1.	1242	228,132	1703
2.	1240	154,355	1793
3.	837	131,637	4804
4.	3484	140,862	16,997
5.	5384	127,188	876
6.	410	176,018	690
7.	250	186,657	420
8.	220	104,690	350
9.	1756	223,240	2593
10.	320	206,110	480
11.	1245	249,238	1797
12.	1283	159,256	1861
13.	570	254,480	824
14.	573	154,821	827
15.	582	189,159	809
16.	350	278,890	490
17.	300	71,354	420
18.	602	254,487	854
19.	1239	154,050	1792
20.	300	130,390	460
21.	30	40	40
22.	7023	123,052	617
23.	4424	$> t_e$	6639
24.	4480	$> t_e$	6715
25.	457	143,701	714
26.	1218	251,955	1692
27.	30	40	40
28.	30	40	40
29.	7760	$> t_e$	15,279
30.	126,094	$> t_e$	$> t_e$

Remark 1. *Time-limiter parameter is introduced in [3]. It is posed as an indicator for stopping the code execution, after some defined time,* $t_e \approx 120$ *s.*

In the next Listing 1, we list the set of test functions examined in this research. We applied all three compared methods to each of these functions. The proposed functions are taken from a collection of unconstrained optimization test functions introduced in [9].

Listing 1. Test functions.

1. Extended Penalty
2. Perturbed Quadratic
3. Raydan-1
4. Diagonal 1
5. Diagonal 3
6. Generalized Tridiagonal-1
7. Extended Tridiagonal-1
8. Extended Three Expon. Terms
9. Diagonal 4
10. Extended Himmelblau
11. Quadr. Diag. Perturbed
12. Quadratic QF1
13. Exten. Quadr. Penalty QP1
14. Exten. Quadr. Penalty QP2
15. Quadratic QF2
16. Extended EP1
17. Extended Tridiagonal-2
18. Arwhead
19. Almost Perturbed Quadratic
20. Engval1
21. Quartc
22. Generalized Quartic
23. Diagonal 7
24. Diagonal 8
25. Diagonal 9
26. DIXON3DQ
27. NONSCOMP
28. HIMMELH
29. Power (Cute)
30. Sine

In Table 1, we display the results concerning the number of iterations metric. All three of the models provide very good numerical outcomes regarding the number of needed iterations. As expected, modADS and ADSS have an equal number of iterations for many test functions, precisely, 21 out of 30. This is due to the modADS iterative form having similar characteristics to those of the ADSS iteration. All three models give the same number of iterations for three cases. With that, each of the modADS and ADD give the lowest number of iterations in 6 out of 30 cases while ADSS does so in only 1 of 30 cases. A general view shows that modADS gives the final outcomes for all 30 test functions, ADD for 26 and ADSS for 29. ADD broke the time-limiter constant for the Diagonal 7, Diagonal 8, Power (Cute) and Sine functions. Execution time is exceeded only for the Sine function when the ADSS model is applied.

Regarding the speed of execution of each comparative model, from the obtained numerical outcomes, we can see that the modADS and ADSS models perform almost equally, and that is why we did not display the results obtained on this metric. Both models give zeros for CPU time in 29 out of 30 cases, and only modADS was successfully applied on the test function (Sine), while the ADSS iteration broke the execution time in this case. The ADD model has the worst outcomes in testing this characteristic with four t_e breaks.

The contents of the Table 2 show the number of function evaluations for all three of the tested models. It is obvious that the modADS achieved the greatest improvement regarding this performance characteristic, when compared to the other two test processes. This method convincingly gives the lowest number of function evaluations in 29 out of

30 cases. The ADSS has the best outcome in 1 case only, while the ADD has very high numbers as results regarding this metric for almost all 30 test functions.

The average values concerning the three analyzed criteria for all comparative models are displayed in Table 3. We included the results of these computations achieved on 26 out of 30 test functions, on which we could apply all methods without breaking the execution time. From this Table, we can obtain a general impression about the performance features of the generated modADS process in comparison to its forerunners. We see that this new accelerated variant is equally fast as the ADSS scheme , it slightly goes beyond the ADSS regarding the number of iterations metric and evidently gives a significant shift in the number of evaluations. When compared with the ADD iteration, the modADS iteration upgrades it multiple times regarding all three performance profiles. More precisely, the modADS gives a 4 times lower average number of iterations, more than a 142 times lower number of function evaluations and it is multiple times faster than the ADD process.

Table 3. ModADS, ADD and ADSS average outcomes of all 3 analyzed metrics obtained on 26 test functions from Listing 1.

Average Metrics	modADS	ADD	ADSS
Number of iterations	145.81	583.23	148.42
CPU time (sec)	0	135.85	0
Number of function evaluations	1191.35	157,455.46	1691.65

We now analyze the dependency of the approaches regarding the Backtracking parameter beta. As mentioned before in this Section, in all previously displayed results, in the algorithms of all three comparative models, the value of this parameter was set to $\beta = 0.8$. We conducted 600 additional tests over the modADS, the ADD and the AGD algorithms for 2 more values of this parameter: $\beta = 0.3$ and $\beta = 0.6$. For that purpose, we chose the first 10 test functions from the Listing 1. In the following Tables 4 and 5, we display the sums of the obtained results regarding the number of iterations and the number of evaluations for these three comparative models. As expected, the modADS demonstrates similar performance regarding the analyzed metrics when compared to the ADD and the ADSS methods, just as in the case of $\beta = 0.8$. Concerning the number of iterations, for both beta values, the modADS acts similar to the ADSS method. Regarding the number of evaluations, again for each of the 2 additional beta values, it gives the best results in 7 out of 10 cases when compared to the ADSS and in all 10 cases in comparison to the ADD scheme.

Table 4. Number of iterations for $\beta = 0.3$ and $\beta = 0.6$.

Function Number	modADS 0.3	ADD 0.3	ADSS 0.3	modADS 0.6	ADD 0.6	ADSS 0.6
1.	50	72	50	50	72	50
2.	432	81	432	432	82	432
3.	725	86	35	49	76	31
4.	33	78	76	40	83	102
5.	40	81	46	43	82	44
6.	83	100	78	80	110	76
7.	340	100	70	70	110	70
8.	40	100	40	40	100	40
9.	788	100	781	783	100	781
10.	70	90	70	70	100	70

Table 5. Number of evaluations for $\beta = 0.3$ and $\beta = 0.6$.

Function Number	modADS 0.3	ADD 0.3	ADSS 0.3	modADS 0.6	ADD 0.6	ADSS 0.6
1.	342	165,094	778	620	184,537	1068
2.	945	117,659	1498	1037	128,657	1587
3.	4078	101,862	273	743	103,993	338
4.	195	96,897	944	665	121,558	13,181
5.	436	109,838	450	3642	116,573	594
6.	250	142,482	2137	296	148,475	532
7.	896	84,460	360	220	83,040	380
8.	220	116,101	220	280	90,350	270
9.	1626	182,688	2453	1656	189,410	2493
10.	210	126,380	400	250	183,600	266

Furthermore, we compare performance metrics between the modADS and the transformed ADSS, i.e., the TADSS. In [5], the authors confirmed that the TADSS provides better numerical outcomes regarding the number of iterations, CPU time and number of function evaluations in comparison with the ADSS scheme on 22 chosen test functions. From the results presented in the previous Tables 1–5, we concluded that the modADSS behaves similarly to the ADSS regarding the number of iterations and the CPU time, but it provides a lower number of evaluations. Due to results from [5], we may expect that the TADSS has better performance results than the modADSS with respect to the number of iterations. In Table 6, we present the achieved test results not only for the 22 test functions from [5] but for all 30 test functions from Listing 1. With that, we show in Table 7 a more general overview of the average results regarding all analyzed metrics.

Table 6. Number of iterations and number of function evaluations, modADS and TADSS.

Function Number	modADS num.it.	TADSS num.it.	modADS num.eval.	TADSS num.eval.
1.	50	40	1242	1082
2.	432	10,973	1240	29,624
3.	31	1183	837	9355
4.	60	22	3484	349
5.	41	23	5384	439
6.	80	60	410	412
7.	70	60	250	250
8.	40	40	220	400
9.	783	40	1756	270
10.	70	60	320	300
11.	428	6915	1245	34,053
12.	470	5314	1283	14,650
13.	60	50	570	570
14.	61	86	573	672
15.	60	50	582	563
16.	40	167	350	776
17.	80	620	300	1993
18.	60	50	602	582
19.	432	10,715	1239	29,150
20.	70	60	300	290
21.	10	10	30	30
22.	76	60	7023	256
23.	2202	199	4424	572
24.	2215	174	4480	696

Table 6. *Cont.*

Function Number	modADS num.it.	TADSS num.it.	modADS num.eval.	TADSS num.eval.
25.	32	24	457	448
26.	235	10	1218	30
27.	10	10	30	30
28.	10	10	30	40
29.	3870	1752	7760	8644
30.	2061	$t_e > l$	126,094	$t_e > l$

Although the results from Table 6 illustrate that the TADSS provides a lower number of iterations in even 17 out of 30 test functions, still the general average outcomes confirm that the modADS provides more than 3 times better outcomes with respect to this metric than the TADSS process. According to the Table 6 results, when we analyze the number of function evaluations, the modADS and the TADSS obtain an equal number of the best outcomes. Yet, from the results presented in Table 7, we are assured that the modADS is almost three times more effective on this matter when compared to the TADSS iteration. From Table 6, we can also notice that for the Sine function, the TADSS process exceeds the execution time.

Table 7. ModADS and TADSS average outcomes of all 3 analyzed metrics obtained on 29 test functions from Listing 1.

Average Metrics	modADS	TADSS
Number of iterations	416.48	1337.14
CPU time (sec)	0.07	2.97
Number of function evaluations	47,639	136,506

To achieve more general view of the performance features of the modADS method, we conducted additional comparisons with a classical gradient method, defined by Cauchy, and with the accelerated gradient method from [1]. We further denote these comparative methods by GD and AGD, respectively. The execution times were very long for the previously chosen number of variables. Due to that reason, we changed this set into the set of the next 10 decreased values: 10, 100, 200, 300, 500, 700, 800, 1000, 2000 and 3000. We tested the first 15 test functions from the Listing 1 by applying the modADS, the GD and the AGD iterative rules. The sums of 450 additional tests outcomes are displayed in the following Tables 8–10.

From Table 8, we can see that it is undoubtedly evident that the modADS gives the lowest number of iterations compared to the GD and the AGD methods in all 15 test functions.

Table 8. The number of iterations for first 15 test functions obtained by modADS, GD and AGD methods.

Function Number	modADS	GD	AGD
1.	52	2058	271
2.	599	50,863	61,678
3.	44	20,823	15,344
4.	58	11,650	11,563
5.	59	19,178	29,673
6.	80	888	583
7.	70	678,648	1768
8.	40	1784	396
9.	788	8484	100

Table 8. *Cont.*

Function Number	modADS	GD	AGD
10.	70	1295	321
11.	595	354,364	549,164
12.	608	53,103	62,996
13.	61	579	182
14.	61	86,323	109,632
15.	61	63,745	11,797

The CPU execution time needed when 3 comparative models are applied on first 15 test functions is listed in the Table 9. We see that, except in four cases when all three methods have the same (zero) outcomes, the modADSS is again a dominant model regarding this aspect, as well.

Table 9. CPU for first 15 test functions obtained by modADS, GD and AGD methods.

Function Number	modADS	GD	AGD
1.	0	1	0
2.	0	116	150
3.	0	11	6
4.	0	7	8
5.	0	22	37
6.	0	0	0
7.	0	198	0
8.	0	0	0
9.	0	0	0
10.	0	0	0
11.	0	1414	3000
12.	0	1445	192
13.	0	0	0
14.	0	673	785
15.	0	767	20

The number of objective function evaluations achieved by the modADS, the GD and the AGD are illustrated in the Table 10. General conclusions over this performance metric are the same as regarding the number of iterations (Table 8), i.e., the modADS has the best outcomes for all 15 test functions.

Table 10. The number of evaluations for first 15 test functions obtained by modADS, GD and AGD methods.

Function Number	modADS	GD	AGD
1.	929	42549	5822
2.	1469	1,747,145	1,971,495
3.	1260	416,274	240,666
4.	3137	355,313	316,838
5.	3126	577,545	838,896
6.	414	14,456	8321
7.	250	3457,777	7102
8.	220	17,968	3413
9.	1766	165,938	1110
10.	320	24,565	5591

Table 10. *Cont.*

Function Number	modADS	GD	AGD
11.	1472	11,880,543	16,276,884
12.	1466	1,656,738	1,823,829
13.	466	9679	2163
14.	467	2,489,732	2,719,409
15.	477	2,390,405	356,569

As a summary, we display in Table 11 the comparisons of the average results obtained by three comparative methods (modADS, GD and AGD) regarding all three performance characteristics. The results displayed in this table confirm that the modADSS requires an approximately 417 times lower number of iterations compared to the GD method and an about 263 times lower number of iterations compared to the AGD method. Regarding the needed number of evaluations, the modADS outperforms the GD and the AGD methods over the 1420 times.

Table 11. The average number of all 3 analyzed metrics obtained on first 15 test functions from Listing 1.

Average Metrics	modADS	GD	AGD
Number of iterations	216.4	90,252.33	57,031.2
CPU time (sec)	0	310.27	279.87
Number of function evaluations	1149.27	1,683,108.47	1,638,540.53

5. Discussion

We defined an optimization model for solving a large scale of unconstrained minimization problems. This method belongs to the class of accelerated gradient iterations with quasi-Newton features. The presented modADS method could be classified in this manner since it contains the scalar matrix approximation of the Hessian, instead of the Hessian itself, with a guiding scalar, the so-called approximation parameter. Previous research on accelerated gradient optimization models confirms that the existence of this parameter directly improves the performance profiles vice versa to the relevant non-accelerated version [3]. In this paper, we chose to develop this acceleration parameter based on the second order Taylor expansion of the posed iteration.

The modADS originates from the accelerated double direction and double step size methods, and the so conducted convergence analysis is similar to those taken in [4]. It confirmed that the developed model is linearly convergent on the sets of uniformly convex and strictly convex functions.

The outcomes of the numerical experiments conducted on the modADS, the ADD and the ADSS methods for three values of the Backtracking parameter β, show the convincing improvement in reducing the number of function evaluations in favor of the developed model. The ADSS method has one execution break, while the ADD has even four. The modADS highly outperforms the ADD method regarding all analyzed metrics.

When compared with the Cauchy's gradient method and the Andrei's accelerated gradient descent method from [1], the modADS outperforms these models multiple times concerning all performance metrics.

6. Conclusions

The proposed iterative rule has the elements of the accelerated double step size-ADSS method [4] and accelerated double direction-ADD method. [3]. In defining modADS, as in previously mentioned methods, we kept the inexact line search Backtracking technique [10] to define an iterative step length value.

We conducted the convergence analysis and proved that the proposed modADS process is at least linearly convergent for the uniformly convex and strictly convex quadratic functions.

Through numerical experiments, we generally conclude that, when compared with the baseline methods, the modADS algorithms has more similarities with the ADSS scheme than with the ADD method. With that, it upgrades both comparative models, primarily because only the modADS method provides numerical outcomes for all 30 test functions, without exception, which confirms the stability of the defined model. In comparison to the classical gradient descent method and accelerated gradient descent method from [1], the defined modADS shows a convincing progress regarding all monitored features.

From all exposed, we conclude that the proposed accelerated gradient minimization model is an effective and efficient algorithm which can be applied for solving many unconstrained optimization tasks.

Author Contributions: Conceptualization, M.J.P.; methodology, M.J.P. and D.V.; software, M.J.P.; validation, M.J.P., D.V., D.I. and A.V.; formal analysis, M.J.P., D.V. and D.I.; investigation, M.J.P., A.V. and J.M.; resources, M.J.P. and D.V.; data curation, M.J.P., D.V., A.V. and J.M.; writing—original draft preparation, M.J.P.; writing—review and editing, D.V., D.I. and A.V.; visualization, A.V. and J.M.; supervision, M.J.P., D.V. and D.I.; project administration, A.V. and J.M.; funding acquisition, A.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research was financially supported by internal-junior project IJ-0202, Faculty of Sciences and Mathematics, University of Priština in Kosovska Mitrovica.

Data Availability Statement: Data results are available on readers request.

Acknowledgments: The first author gratefully acknowledges support from the project Grant No. 174025 by Ministry of Education and Science of Republic of Serbia.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
DOAJ	Directory of open access journals
TLA	Three letter acronym
LD	Linear dichroism

References

1. Andrei, N. An acceleration of gradient descent algoritham with backtracing for unconstrained optimization. *Numer. Algor.* **2006**, *42*, 63–173.
2. Stanimirovic, P.S.; Miladinović, M.B. Accelerated gradient descent methods with line search. *Numer. Algor.* **2010**, *54*, 503–520.
3. Petrović, M.J.; Stanimirovic, P.S. Accelerated Double Direction Method For Solving Unconstrained Optimization Problems. *Math. Probl. Eng.* **2014**, *2014*, 965104. https://doi.org/10.1155/2014/965104
4. Petrović, M.J. An accelerated Double Step Size method in unconstrained optimization. *Appl. Math. Comput.* **2015**, *250*, 309–319. https://doi.org/10.1016/j.amc.2014.10.104
5. Stanimirovic, P.S.; Petrović, M.J.; Milovanović, G.V. A Transformation of Accelerated Double Step Size Method for Unconstrained Optimization. *Math. Probl. Eng.* **2015**, *2015*, 283679. https://doi.org/10.1155/2015/283679
6. Petrović, M.J.; Ivanović, M.; Djordjević, M. Comparative performance analysis of some accelerated and hybrid accelerated gradient models. *Univ. Thought Publ. Nat. Sci.* **2019**, *9*, 57–61. https://doi.org/10.5937/univtho9-18174
7. Ortega, J.M.; Rheinboldt, W.C. Iterative Solution of Nonlinear Equation in Several Variables. In *Iterative Solution of Nonlinear Equation in Several Variables*; Academic Press: London, UK, 1970.
8. Rockafellar, R.T. Convex Analysis. In *Convex Analysis*; Princeton University Press: Princeton, NJ, USA, 1970.
9. Andrei, N. An Unconstrained Optimization Test Functions Collection. *Adv. Model. Optim.* **2008**, *10*, 1–15. Available online: http://www.apmath.spbu.ru/cnsa/pdf/obzor/An%20Unconstrained%20Optimization%20Test%20Functions%20Collection.pdf (accessed on 14 January 2022).
10. Armijo, L. Minimization of functions having Lipschitz continuous first partial derivatives. *Pac. J. Math.* **2008**, *16*, 1–3.