mathematics

MDPI

Article

A New Finite-Difference Method for Nonlinear Absolute Value Equations

Peng Wang [1,2,3,*], Yujing Zhang [1,2,3] and Detong Zhu [4]

[1] Key Laboratory of Data Science and Smart Education Ministry of Education, Hainan Normal University, Haikou 570203, China; 150115@hainnu.edu.cn
[2] Key Laboratory of Computational Science and Application of Hainan Province, Haikou 571158, China
[3] Mathematics and Statistics College, Hainan Normal University, Haikou 570203, China
[4] Mathematics and Science College, Shanghai Normal University, Shanghai 200234, China; dtzhu@shnu.edu.cn
* Correspondence: pengwang621@163.com

Abstract: In this paper, we propose a new finite-difference method for nonconvex absolute value equations. The nonsmooth unconstrained optimization problem equivalent to the absolute value equations is considered. The finite-difference technique is considered to compose the linear programming subproblems for obtaining the search direction. The algorithm avoids the computation of gradients and Hessian matrices of problems. The new finite-difference parameter correction technique is considered to ensure the monotonic descent of the objective function. The convergence of the algorithm is analyzed, and numerical experiments are reported, indicating the effectiveness by comparison against a state-of-the-art absolute value equations.

Keywords: nonsmooth unconstrained optimization; linear programming methods; approximation gradient; global convergence; multi-directional search

MSC: 49M37; 65K05; 90C30; 90C56

check for updates

Academic Editor: Aleksandr Rakhmangulov

Received: 5 February 2025
Revised: 27 February 2025
Accepted: 3 March 2025
Published: 5 March 2025

Citation: Wang, P.; Zhang, Y.; Zhu, D. A New Finite-Difference Method for Nonlinear Absolute Value Equations. *Mathematics* **2025**, *13*, 862. https://doi.org/10.3390/math13050862

1. Introduction

1.1. Problem Description Motivation

In this paper, we propose a new finite-difference method for solving the nonconvex absolute value equations of the following from:

$$Ax - B|x| = d. \tag{1}$$

where $A, B \in \mathbb{R}^{m\times n}$ are $m \times n$ matrices, and $x \in \mathbb{R}^n$ and $d \in \mathbb{R}^m$ are n-dimensional and m-dimensional column vectors, respectively. The problem (1) arises frequently in bimatrix games, contact problems and linear and convex quadratic programming [1–3]. In addition, the importance and practicality of developing numerical algorithms and theoretical frameworks for absolute value equations are not limited to their theoretical meanings but also include a wide range of potential applications and enormous economic value. Therefore, addressing the challenges brought by absolute value equations is not only valuable in theory but also of significant economic importance.

In order to obtain the solution of Problem (1), we consider solving the following optimization problem:

$$\min_{x\in\mathbb{R}^n} \quad f(x) = \sum_{i=1}^{m} |A_i x + B_i|x| - d_i|, \tag{2}$$

where A_i and B_i are the $i-$th row of Matrices A and B, respectively, and d_i is the $i-$th component of d. It is obvious that the optimal solution of Problem (2) is the solution of Problem (1). Hence, we consider the new algorithm to solve Problem (2). However, Problem (2) is a nonsmooth problem; thus, it is difficult to obtain the optimal solution of Problem (2) using traditional smooth optimization algorithms. In this paper, we introduce the nonsmooth optimization algorithm to obtain the solution of Problem (2).

The nonlinear absolute value equations (AVEs) already have a wide spectrum of applications, such as economics, engineering, transportation science, and mathematical programming, especially mixed integer linear programming problems. It is very difficult to deal with nonlinear and nonsmooth ones. The general form of (1) was first presented by Rohn [4]. Then, Mangasarian [1,5,6] promoted it in a more general context. In many cases, AVE problems were equivalent to linear complementarity (LPC) problems (see [7,8]). Some studies introduced several theoretical results to guarantee the uniqueness of the solution [9–12]. Furthermore, if an AVE problem was linked to an LCP, then an AVE problem could be associated with nonlinear complementarity problems (NCPs), taking profit from the huge amount of literature concerning the existence, uniqueness and numerical resolution of NCPs (see [13]).

In order to solve AVE problems, various numerical algorithms can be used. Generally speaking, there are three categories: iterative linear algebra based methods, semi-smooth Newton-like methods and smoothing methods. The aforementioned methods can solve AVE problems, but assumption such as P_0-matrices and P-matrices are needed [14]. In addition, Caccetta et al. [15] proposed a smoothing Newton method for AVEs. They showed that the method has a global and quadratic convergence rate when singular values of A exceed 1. Li [16] introduced a preconditioned accelerated over-relaxation iterative method coupled with a preconditioning technique for solving absolute value equations. Studies have shown that the convergence rate of the preconditioned accelerated over-relaxation iterative method is better than that of the accelerated over-accelerated over-relaxation iterative method. Yong [17] proposed a particle swarm optimization (PSO), which was based on aggregate function for solving AVE problems. They used an aggregate function to replace the absolute value function, and the nonsmooth AVEs could be solved by solving smooth nonlinear equations. Moosaei et al. [18] proposed a new algorithm for solving NP-hard absolute value equations in which the singular values of A exceed 1. Some randomly generated problems were solved by proposed methods and other other known methods. The comparison results showed that these methods were more effective than other known methods. Wu and Li [19] introduced a special shift splitting iteration method for solving nonlinear absolute value equations. They showed that the special shift splitting iteration method was absolutely convergent under proper conditions. Yong [20] also introduced an iterative method for solving AVE problems. He showed that the sequence generated by the algorithm converged to a solution of AVEs after finite iterations. Finally, this method was applied to solve a two-point boundary value problem. Saeed Ketabchi et al. [21] created an algorithm to compute the minimum norm solution to the absolute value equation (AVE) in a special case. They pointed out that AVEs could be reduced to an unconstrained minimization problem with a once-differentiable convex objective function by using an exterior penalty method. Hence, a quasi-Newton method was used for solving this problem. Ref. [22] studied the optimum correction of the absolute value equation through making minimal changes in the coefficient matrix and the right-hand side using the l_2 norm. Hence, a genetic algorithm could be used for obtaining the solution of this problem. Moreover, many methods used for solving absolute value equations were also used to solve partial differential equations (PDEs); for example, Liu et al. [23] introduced the bilinear neural network method for solving partial differential equations. Taylor et al. [24] also introduced a residual neural network method for solving partial differential equations. These methods were also useful for solving AVE problems.

1.2. Contribution of This Paper

In order to solve the nonlinear absolute value equation (Equation (1)), a new finite-difference method is introduced in this paper. The algorithm's contributions are as follows:

- The finite-difference method is proposed for finding the solution of Problem (1). We prove that the sequence generated by the algorithm converges to the solution of Problem (1).
- The proposed algorithm avoids computing the gradients and Hessian matrices' information to obtain the subproblem for solving the search direction. The finite-difference technique is only used for proposing the linear programming subproblem to obtain the search direction.
- Unconstrained nonsmooth optimization problems are established, and their optimal solutions are guaranteed to be absolute value equations. A new finite-difference parameter correction technique is used to ensure the monotonic descent of the objective function of unconstrained nonsmooth optimization problems.
- In contrast to general smooth optimization algorithms for solving absolute value equations, P_0-matrix and $P-$matrix approximations for Problem (1) are not needed in this paper for the convergence of the algorithm.

This paper is organized into six sections. The finite-difference linear programming subproblem is introduced in Section 2. The finite-difference algorithm is introduced in Section 3. We prove that the sequence generated by the algorithm converges to the solution of Problem (1) in Section 4. Numerical experiments illustrating the practical performance of the algorithm are reported in Section 5. The conclusions of this paper are outlined in Section 6.

2. The Linear Programming Subproblem

In this paper, we consider a new algorithm for obtaining the solution of Problem (1). First, we consider building the linear programming subproblems using the information of the gradient of objective function $f(x)$. However, gradient information is not obtained. Hence, the finite difference is used to approximate the gradient of objective function $f(x)$. The finite difference relies on an appropriate finite-difference parameter, t.

We define the finite-difference interval, as suggested in [25]. The ith component of the approximation finite difference which approximates the gradient of $f(x)$ at x is defined by

$$\begin{aligned} [g^+(x)]_i &= f(x+te_i) - f(x), \quad (3) \\ [g^-(x)]_i &= f(x-te_i) - f(x), \quad (4) \end{aligned}$$

where $e_i, i \in \{1, 2, \cdots, n\}$ is the n-dimensional unit vector whose ith element is 1.

Using (3) and (4), we define $g^+(x)$ and $g^-(x)$ as follows:

$$\begin{aligned} g^+(x) &= [g_1^+(x), g_2^+(x), \cdots, g_n^+(x)]^T, \quad (5) \\ g^-(x) &= [g_1^-(x), g_2^-(x), \cdots, g_n^-(x)]^T. \quad (6) \end{aligned}$$

Hence, the finite-difference gradient is given as follows:

$$g(x) = \begin{bmatrix} g^+(x) \\ g^-(x) \end{bmatrix}. \quad (7)$$

We hope that the algorithm can generate a search direction relying on (8). In order to achieve this goal, i.e., in order to obtain the search direction of Problem (1), we will present the following subproblem:

$$\begin{aligned} \min \quad & g_k^T \begin{bmatrix} \lambda \\ \mu \end{bmatrix} = [g_k^+]^T \lambda + [g_k^-]^T \mu \\ s.t. \quad & \sum_{i=1}^{n} (\lambda_i + \mu_i) = 1, \\ & |\lambda_i| \le \Delta_k, \ \ |\mu_i| \le \Delta_k, i = 1, 2, \cdots, n \end{aligned} \tag{8}$$

where $\Delta_k > 0$, $\lambda \in \mathbb{R}^n$, $\mu \in \mathbb{R}^n$, $g_k = g(x_k)$, $g_k^+ = g^+(x_k)$, $g_k^- = g^-(x_k)$, $g(x)$, $g^+(x)$ and $g^-(x)$ are generated by (7), (5) and (6), respectively.

3. Algorithm

In this section, we introduce a new finite-difference method for finding the optimal solution of Problem (2), and we obtain the solution of Problem (1). The design of the algorithm is mainly inspired by the idea of a multi-directional search. However, the basic idea of a multi-directional search is to explore each direction given until a direction that can cause the objective function to decrease is found. Our algorithm will utilize the theory of spatial basis in higher algebra and construct a linear programming model to avoid a one-by -one search.

Remark 1. *In Step 6 of the algorithm, we control the finite-difference parameter such that it is also a step size along the search direction d_k.*

Remark 2. *Algorithm 1 may obtain a local optimal solution of Problem (1) if it is not a convex optimization problem and this solution is not the solution of Problem (1). From Figure 1, we can see that if we expand the finite-difference parameter, t_k, solving Subproblem (8) will generate a new descent direction until the algorithm produces a solution to Problem (1). Hence, the parameter correction plan in Steps 3, 4 and 5 of Algorithm 1 is necessary for finding the solution to Problem (1).*

Algorithm 1: FDM

```
Input: Given initial iteration $x_0 \in \mathbb{R}^n$, the constants $t_0 \in (0,1)$, $\Delta_0 > 0$, $0 < \sigma < 1 < \chi$ and $\epsilon > 0$. Set $k = 0$.
Main Steps:
1. Calculate the approximate gradient $g_k = g(x_k)$ using (7).
2. If $|f(x_k)| \le \epsilon$, then stop; $x_k$ is a solution of Problem (1).
3. If $\min\{[g_k^+]_1, \cdots, [g_k^+]_n, [g_k^-]_1, \cdots, [g_k^-]_n\} > 0$, then let $t_k^1 = \sigma t_k$.
     If $t_k^1 > \epsilon$, compute $g_k^1 = \min\{[g_k^+]_1^1, \cdots, [g_k^+]_n^1, [g_k^-]_1^1, \cdots, [g_k^-]_n^1\}$ where $[g_k^+]_i^1 = f(x_k + t_k^1 e_i)$, $[g_k^-]_i^1 = f(x_k - t_k^1 e_i)$.
     Else, $g_k^1 = (1, \cdots, 1, 1, \cdots, 1)$, and go to 4.
4. Let $t_k^2 = \chi t_k$, compute $g_k^2 = \min\{[g_k^+]_1^2, \cdots, [g_k^+]_n^2, [g_k^-]_1^2, \cdots, [g_k^-]_n^2\}$ where $[g_k^+]_i^2 = f(x_k + t_k^2 e_i)$, $[g_k^-]_i^2 = f(x_k - t_k^2 e_i)$.
5. If $\min\{g_k^1, g_k^2\} \ge 0$, then go to 3.
6. If $g_k^1 \le g_k^2$, then $t_k = t_k^1$; go to 5.
   Else $t_k = t_k^2$, go to 5.
7. Calculate Subproblem (8) and obtain the search direction $d_k = \lambda_k - \mu_k$, where $\lambda_k = \lambda(x_k)$ and $\mu_k = \mu(x_k)$.
8. If $f(x_k + t_k d_k) \le \sum_{i=1}^{n} [\lambda_k]_i f(x_k + te_i) + \sum_{i=1}^{n} [\mu_k]_i f(x_k - te_i)$, then let $x_{k+1} = x_k + t_k d_k$.
     Else, let $d_k = \pi_k e_{j_0}$ and $x_{k+1} = x_k + d_k$ such that $f(x_k + \pi_k e_{j_0}) - f(x_k) = f(x_k + d_k) - f(x_k) = \min\{f(x_k + t_k e_1) - f(x_k), \cdots, f(x_k + t_k e_n) - f(x_k), f(x_k - t_k e_1) - f(x_k), \cdots, f(x_k + t_k e_n) - f(x_k)\}$.
   Set $k = k + 1$, and go to 1.
```

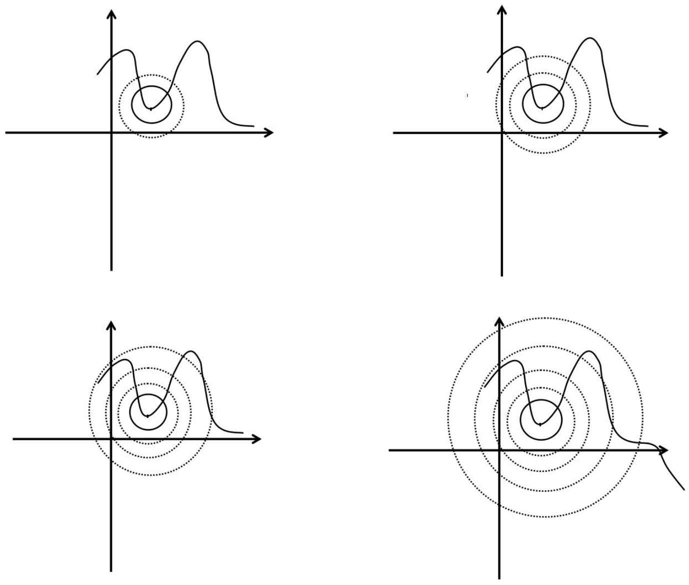

Figure 1. Schematic diagram of selecting parameter t_k in Algorithm 1.

Figure 1 shows how parameter t_k is selected in Algorithm 1, such that Algorithm 1 can obtain the solution to Problem (1). The radius of the dashed circle in the four subgraphs in Figure 1 represents the length of parameter t_k. These four subgraphs show that as the radius increases (parameter t_k increases), the dashed circle can intersect with a function value lower than the center point (current iteration point x_k). This indicates that with the increase in the value of parameter t_k, Algorithm 1 can further reduce the problem and ultimately obtain the solution to Problem (1).

4. Convergence Analysis

In this section, we provide the global convergence analysis of Algorithm 1. Similar to the convergence proofs in other studies, the convergence proof of the algorithm in this paper mainly states two points: Firstly, it proves that the algorithm can reduce the objective function of Problem (2) with each iteration before termination under the general assumptions mentioned in the other studies. Secondly, it proves that the solution to Problem (1) can be obtained when Algorithm 1 is terminated. We require the model to satisfy the following assumption:

Assumption 1. *We define the level set as*

$$L(x_0) = \{x \in \boldsymbol{R}^n,\ f(x) \leq f(x_0)\}.$$

Suppose that the level set is bounded.

Next, we will prove that the limit point x^* of the sequence $\{x_k\}$ generated by Algorithm 1 is a solution to Problem (1). First, we show that the sequence generated by Algorithm 1 is such that the objective function of Problem (2) is monotonically decreasing.

Lemma 1. *Under Assumption 1, sequence $\{x_k\}$ is generated by Algorithm 1 such that the objective function of Problem (2) is monotonically decreasing, i.e., for any k and $k+1$, we have that*

$$f(x_{k+1}) < f(x_k).$$

Proof. According to Step 3 of Algorithm 1, if there exists $j_0 \in \{1, 2, \cdots, n\}$ such that $[g_k^+]_i < 0$ or $[g_k^-]_i < 0$, then let $\lambda_{j_0} = 1$ or $\mu_{j_0} = 1$ and $\lambda_i = 0, i \neq j_0$ or $\mu_i = 0, i \neq j_0$. It is obvious that either $[\lambda_k^T, \mu_k^T] = [0, \cdots, [\lambda_k]_{j_0}, 0, \cdots, 0]$ or $[\lambda_k^T, \mu_k^T] = [0, \cdots, [\mu_k]_{j_0}, 0, \cdots, 0]$ is a feasible solution of Subproblem (8), and we have that

$$g_k^T \begin{bmatrix} \lambda_k \\ \mu_k \end{bmatrix} = [g_k^+]^T \lambda_k + [g_k^-]^T \mu_k = [g_k^+]_{j_0} < 0, \tag{9}$$

or

$$g_k^T \begin{bmatrix} \lambda_k \\ \mu_k \end{bmatrix} = [g_k^+]^T \lambda_k + [g_k^-]^T \mu_k = [g_k^-]_{j_0} < 0. \tag{10}$$

If $\min\{[g_k^+]_1, \cdots, [g_k^+]_n, [g_k^-]_1, \cdots, [g_k^-]_n\} > 0$, then, according to Steps 3–6 of Algorithm 1, there exists $t_k > 0$ such that $\min\{[g_k^+]_1, \cdots, [g_k^+]_n, [g_k^-]_1, \cdots, [g_k^-]_n\} < 0$, similar to (9) and (10). Therefore, we have

$$g_k^T \begin{bmatrix} \lambda_k \\ \mu_k \end{bmatrix} < 0. \tag{11}$$

According to the definition of g_k, we obtain

$$\begin{aligned} g_k^T \begin{bmatrix} \lambda_k \\ \mu_k \end{bmatrix} &= \sum_{i=1}^{n} [\lambda_k]_i [g_k]_i^+ + \sum_{i=1}^{n} [\mu_k]_i [g_k]_i^- \\ &= \sum_{i=1}^{n} [\lambda_k]_i [f(x_k + te_i) - f(x_k)] + \sum_{i=1}^{n} [\mu_k]_i [f(x_k - te_i) - f(x_k)]. \end{aligned}$$

If $f(x_k + t_k d_k) \leq \sum_{i=1}^{n} [\lambda_k]_i f(x_k + te_i) + \sum_{i=1}^{n} [\mu_k]_i f(x_k - te_i)$, by $\sum_{i=1}^{n} (\lambda_i + \mu_i) = 1$, we have

$$\begin{aligned} f(x_k + t_k d_k) - f(x_k) =& f(x_k + t_k d_k) - \left(\sum_{i=1}^{n} [\lambda_k]_i + \sum_{i=1}^{n} [\mu_k]_i \right) f(x_k) \\ \leq& \sum_{i=1}^{n} [\lambda_k]_i f(x_k + te_i) + \sum_{i=1}^{n} [\mu_k]_i f(x_k - te_i) - \left(\sum_{i=1}^{n} [\lambda_k]_i + \sum_{i=1}^{n} [\mu_k]_i \right) f(x_k) \\ =& \sum_{i=1}^{n} [\lambda_k]_i (f(x_k + te_i) - f(x_k)) + \sum_{i=1}^{n} [\mu_k]_i (f(x_k + te_i) - f(_k)) \\ =& g_k^T \begin{bmatrix} \lambda_k \\ \mu_k \end{bmatrix} < 0. \end{aligned} \tag{12}$$

From (12), we have $f(x_k + t_k d_k) - f(x_k) < 0$, i.e., $f(x_k + t_k d_k) < f(x_k)$, which means that the objective function is monotonically decreasing.

If $f(x_k + t_k d_k) > \sum_{i=1}^{n} [\lambda_k]_i f(x_k + te_i) + \sum_{i=1}^{n} [\mu_k]_i f(x_k - te_i)$, according to Step 8 of Algorithm 1, we have

$$\begin{aligned} f(x_{k+1}) - f(x_k) =& f(x_k + d_k) - f(x_k) = f(x_k + \pi_k e_{j_0}) - f(x_k) \\ &= \min\{f(x_k + t_k e_1) - f(x_k), \cdots, f(x_k + t_k e_n) - f(x_k), \\ &\quad f(x_k - t_k e_1) - f(x_k), \cdots, f(x_k + t_k e_n) - f(x_k)\} < 0. \end{aligned} \tag{13}$$

Combining (12) and (13), we have that

$$f(x_{k+1}) - f(x_k) < 0,$$

which means that

$$f(x_{k+1}) < f(x_k), \tag{14}$$

and the objective function of Problem (2) is monotonically decreasing. □

The next theorem shows the convergence of Algorithm 1, i.e., the sequence generated by Algorithm 1 converges to the solution of Problem (1).

Theorem 1. *Under Assumption 1, the sequence generated by Algorithm 1 converges to the solution of Problem (1), i.e., we have*

$$\lim_{k\to\infty} f(x_k) = 0.$$

Proof. For any Index $N \in \{1, 2, \cdots\}$, we have that

$$f(x_N) - f(x_0) = \sum_{k=0}^{N} (f(x_{k+1}) - f(x_k)).$$

Hence,

$$f(x_N) = \sum_{k=0}^{N} (f(x_{k+1}) - f(x_k)) + f(x_0).$$

By Lemma 1, we can obtain

$$f(x_{k+1}) - f(x_k) < 0,$$

which means that $f(x_N) \to -\infty$ as $N \to +\infty$. This contradicts the boundedness of function $f(x)$ in Assumption 1. Hence, we have

$$\lim_{k\to\infty} f(x_k) = 0,$$

which means that the sequence generated by Algorithm 1 converges to the solution of Problem (1). □

5. Numerical Results

In this section, we will use actual examples to test the effectiveness of Algorithm 1. In order to reflect the computational efficiency of Algorithm 1, we would like to compare the practical performance of Algorithm 1 with the following codes:

- SM [26]: The nonlinear absolute value equations can be restated as nonlinear complementarity problems and solved efficiently using smoothing regularizing techniques. SM is a smooth method for solving nonlinear complementarity problems. It uses a softmax function that approximates the nonsmooth parts of problems, in which the main idea is to approximate the complementarity condition via the limit

$$\max_{i\in\{1,\cdots,d\}} x_i = \lim_{r\searrow 0} \log\left(\sum_{i=1}^{d} e^{x_i/r}\right),$$

which was widely used in many optimization problems.

- IP [27]: The interior method introduced by Haddou, Migot and Omer in 2019 with the full Newton step for monotone linear complementarity problems. The specificity of the method was to compute the Newton step using a modified system similar to that introduced by Darvay in Stud Univ Babe-Bolyai Ser Inform 47:15-26, 2017. The method considered a general family of smooth concave functions in the Newton system instead of the square root. The method also possessed the best-known upper bound complexity.

In order to test the efficiency of our algorithm, we chose five nonlinear absolute value equations examples from [28–30]. A specific form of the example is as follows:

Example 1. *We solve the systems* $F(x) - |x| = b$*, where* $F(x) = Ax$*,* $A = tridiag(-1, 4, -1)$*,* $x^* \in \mathbb{R}^m$*,* $B = Ax^* - |x^*|$ *and* $x^* = (1, 2, 1, 2, \cdots, 1, 2, \cdots)^T \in \mathbb{R}^m$*.*

Example 2. *We solve the systems* $F(x) - |x| = b$*, where* $F(x) : \mathbb{R}^3 \to \mathbb{R}^3$ *is defined by*

$$F(x) = \begin{pmatrix} 2x_1 - 2 \\ 2x_2 + x_2^3 - x_3 + 3 \\ x_2 + 2x_3 + 2x_3^3 - 3 \end{pmatrix},$$

and we, respectively, consider $b_1 = (-1, -5, 10)^T$*,* $b_2 = (9, -100, 10)^T$*,* $b_3 = (200, 0, 900)^T$*.*

Example 3. *We solve the systems* $F(x) - |x| = b$*, where* $F(x) : \mathbb{R}^4 \to \mathbb{R}^4$ *is defined by*

$$F(x) = \begin{pmatrix} 3x_1^2 + x_1 + 2x_1x_2 + 2x_2^2 + x_3 + 3x_4 \\ 2x_1^2 + x_1 + x_2^2 + x_2 + 10x_3 + 2x_4 \\ 3x_1^2 + x_1x_2 + 2x_2^2 + 3x_3 + 9x_4 \\ x_1^2 + 3x_2^2 + 2x_3 + 4x_4 \end{pmatrix},$$

and we, respectively, consider $b_1 = (10, 10, -12, 0)^T$*,* $b_2 = (20, -100, -12, 1)^T$*,* $b_3 = (200, 10, -5, -5)^T$*.*

Example 4. *Assume that m is a predetermined positive integer and that* $n = m^2$*; moreover, suppose that* $B = I$*,* $A = M + I$*, where*

$$M(x) = \begin{pmatrix} S & -0.5I & 0 & \cdots & 0 & 0 \\ -1.5I & S & -0.5I & \cdots & 0 & 0 \\ 0 & -1.5I & S & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & S & -0.5I \\ 0 & 0 & 0 & \cdots & -1.5I & S \end{pmatrix} \in \mathbb{R}^{n \times n},$$

with

$$S = \begin{pmatrix} 5 & -0.5 & 0 & \cdots & 0 & 0 \\ -1.5 & 4 & -0.5 & \cdots & 0 & 0 \\ 0 & -1.5 & 4 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 4 & -0.5 \\ 0 & 0 & 0 & \cdots & -1.5I & 4 \end{pmatrix} \in \mathbb{R}^{m \times m},$$

and $b = Ax^* - |x^*|$*, in which* $x^* = (1, 2, 1, 2, \cdots, 1, 2, \cdots)^T$*.*

Example 5. *Let m be a prescribed positive integer and $n = m^2$. Consider Problem (1) with $A = tridiag(-I, S, -I) \in \mathbb{R}^{n\times n}$, where $S = tridiag(-1, 8, -1)$ and $B = I$.*

In order to test the computational efficiency of Algorithm 1, different dimensions were selected for Examples 1, 4, and 5 for the calculations. Details are outlined in Table 1.

Table 1. Details of Examples 1, 4 and 5.

Example	**Vector *b***	**Error$_1$**	**Error$_2$**	**Error$_3$**	**Error$_4$**
Example 1	d = 10	10^{-3}	10^{-5}	10^{-8}	10^{-10}
	d = 50	10^{-3}	10^{-5}	10^{-8}	10^{-10}
	d = 100	10^{-3}	10^{-5}	10^{-8}	10^{-10}
	d = 200	10^{-3}	10^{-5}	10^{-8}	10^{-10}
Example 4	d = 100	10^{-3}	10^{-5}	10^{-8}	10^{-10}
	d = 500	10^{-3}	10^{-5}	10^{-8}	10^{-10}
	d = 1000	10^{-3}	10^{-5}	10^{-8}	10^{-10}
	d = 2000	10^{-3}	10^{-5}	10^{-8}	10^{-10}
Example 5	d = 100	10^{-3}	10^{-5}	10^{-8}	10^{-10}
	d = 500	10^{-3}	10^{-5}	10^{-8}	10^{-10}
	d = 1000	10^{-3}	10^{-5}	10^{-8}	10^{-10}
	d = 2000	10^{-3}	10^{-5}	10^{-8}	10^{-10}

Before solving Examples 1–5, we will introduce the parameters selected in the actual calculation of the algorithm proposed in this paper:

$$\Delta_0 = 5, \;\; \sigma = 0.5 \;\text{ and }\; \chi = 2.$$

In order to solve the test problems in Table 1 by using the algorithm in this paper, we use MATLAB (2014a) to write a computer program. The computer we used was the ThinkPad T480 (CPU: i7-8850U; main frequency: 2.0 Hz; memory: 16 G).

In order to draw a comparison diagram of the results of the three algorithms, we used the performance comparison formula of the algorithm proposed by Dolan and More [31] to calculate the computational efficiency between different algorithms. Specific formulas are outlined as follows:

$$r_{p,s} = \frac{\tau_{p,s}}{\min\{\tau_{p,u} : u \in S\}}, \tag{15}$$

where S denotes the set of algorithms. Let P denote the problem sets, $n_s = |S|$ and $n_p = |P|$. For $\forall t \geq 1$, let

$$\rho_s(t) = \frac{1}{n_p} size\{p \in P : r_{p,s} \leq t\}, \tag{16}$$

where $\rho_s(t)$ represents the efficiency of each solver.

We choose Examples 1–5, where the calculation information for Examples 1, 4, and 5 is shown in Table 1. In actual calculations, we randomly selected 50 different initial points and used three algorithms to calculate Examples 1–5. The average number of iterations and the average CPU time are shown in Figures 2 and 3. In Figures 2 and 3, the average number of iterations and average CPU time taken by the three algorithms to solve the problem when ϵ selects different termination accuracies are shown, i.e., $\epsilon = 10^{-3}, \epsilon = 10^{-5}, \epsilon = 10^{-8}$ and $\epsilon = 10^{-10}$.

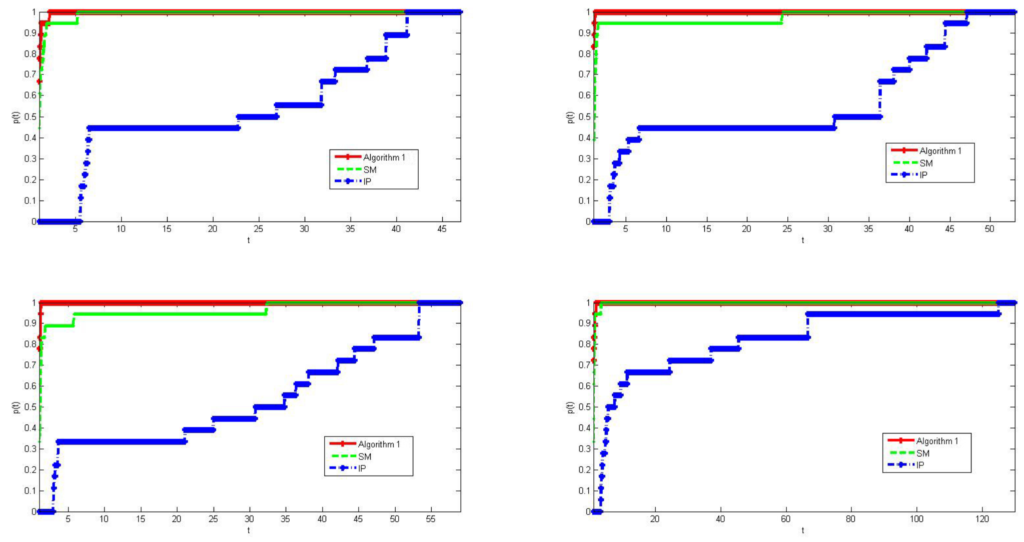

Figure 2. Left of the first row: the comparison results of the iterations of Algorithm 1 with SM and IP when we chose $\epsilon = 10^{-3}$. Right of the first row: the comparison results of the iterations of Algorithm 1 with SM and IP when we chose $\epsilon = 10^{-5}$. Left of the second row: the comparison results of the iterations of Algorithm 1 with SM and IP when we chose $\epsilon = 10^{-8}$. Right of the second row: the comparison results of the iterations of Algorithm 1 with SM and IP when we chose $\epsilon = 10^{-10}$.

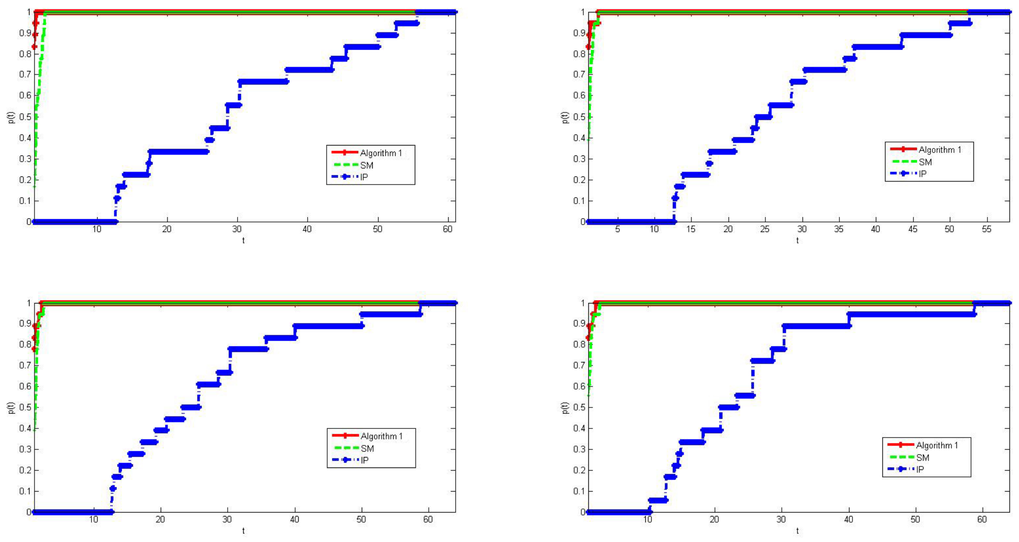

Figure 3. Left of the first row: the comparison results of CPU time of Algorithm 1 with SM and IP when we chose $\epsilon = 10^{-3}$. Right of the first row: the comparison results of CPU time of Algorithm 1 with SM and IP when we chose $\epsilon = 10^{-5}$. Left of the second row: the comparison results of CPU time of Algorithm 1 with SM and IP when we chose $\epsilon = 10^{-8}$. Right of the second row: the comparison results of CPU time of Algorithm 1 with SM and IP when we chose $\epsilon = 10^{-10}$.

Figure 2 shows the results that used Algorithm 1, SM and IP for calculating Examples 1–5. From the subgraph on the left side of the first row, it can be seen that when we chose the the termination accuracy of the algorithm as $\epsilon = 10^{-3}$, our algorithm had a high computational efficiency in calculating all cases of Examples 1–5. According to (15) and (16), the speed at which the red curve, which represents our algorithm, tends to 1 is relatively fast. At the initial position, the red curve reaches 0.7 ($\rho_s(t) = 0.7$), but the green curve only reaches 0.45 ($\rho_s(t) = 0.45$). Meanwhile, the blue curve reaches almost zero. This means that the algorithm can solve more problems by spending a fewer number of iterations than other methods. Meanwhile, from other subgraphs, with the improvement in the algorithm's termination accuracy, the initial position values of the red curve are much higher than other two algorithms, and the speed at which it approaches 1 is also the fastest among the three algorithms. From the comparison results of the three methods, it can be seen that our algorithm has a much higher computational efficiency than the other two algorithms, which means that our algorithm is more efficient in solving test problems.

From Figure 3, we can see that compared with the other three algorithms, the cost of CPU time of Algorithm 1 also increases with the increase in the algorithm's termination accuracy, and our algorithm's computational efficiency continues to improve. From the subgraph on the left of the first row, it can be seen that when the termination accuracy of the algorithm is set to $\epsilon = 10^{-3}$, the value of the red curve at its initial position is much higher than that of the green curve and blue curve, and the red curve tends towards 1 at a faster rate. This means that our algorithm has more effective function calculations than the SM represented by the green curve. Specifically, the cost of CPU time of Algorithm 1 is much smaller than the interior point algorithm, which corresponds to the blue curve. Similarly, from other subgraphs, it can be seen that although we improved the termination accuracy of the algorithm, the red curve corresponding to Algorithm 1 still has significant advantages in terms of the initial position value and the rate at which its speed tends towards 1. This indicates that Algorithm 1 has a higher computational efficiency than the other two algorithms when they solve Examples 1–5.

6. Concluding Remarks

This paper proposes a new finite-difference method for obtaining a solution to absolute value equations. The finite-difference technique is used to avoid computing the gradients and Hessian matrices of the problem. The search direction is obtained by solving the linear programming subproblem. In the numerical calculation section, we first chose five absolute value equations and computed these problems with different dimensions and algorithm termination accuracy. The finite-difference parameter is important for our algorithm. We compared our algorithm with SM and IP. The number of iterations and the number of function evaluations were reported. It can be seen from the comparison results that our algorithm can successfully solve absolute value equations. In comparison with other algorithms, it can also be seen that our algorithm has advantages in both the number of iterations and the cost of CPU time.

With the advent of the big data era, there will inevitably be large-scale absolute value linear equation system problems. Therefore, inspired by the bilinear neural network method and bilinear residual network method, we will study more practical and efficient neural network and residual network methods for solving large-scale absolute value equation systems to meet practical needs.

Author Contributions: Formal analysis, P.W. and D.Z.; Writing—review & editing, Y.Z. All authors have read and agreed to the published version of the manuscript.

Funding: The author would like to thank National Natural Science Foundation (11371253), Hainan Natural Science Foundation (120MS029) and Scientific Research Projects of Higher Learning Institutions in Hainan (Hnky2022-23) for their support.

Data Availability Statement: The original contributions presented in the study are included in the article; further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Mangasarian, O.L. Absolute value programming. *Comput. Optim. Appl.* **2007**, *36*, 43–53. [CrossRef]
2. Mangasarian, O.L. Equilibrium points of bimatrix games. *J. Soc. Ind. Appl. Math.* **1964**, *12*, 778–780. [CrossRef]

3. Okeke, G.A.; Abbas, M. A solution of delay differential equations via PicardCKrasnoselskii hybrid iterative process. *Arab. J. Math.* **2017**, *6*, 21–29. [CrossRef]
4. Rohn, J. A theorem of the alternatives for the equation $Ax + B|x| = b$. *Linear Multilinear Algebra* **2004**, *52*, 421–426. [CrossRef]
5. Mangasarian, O.L. Absolute value equations via concave minimization. *Optim. Lett.* **2007**, *1*, 1–8. [CrossRef]
6. Mangasarian, O.L. A generalized Newton method for absolute value equations. *Optim. Lett.* **2009**, *3*, 101–108. [CrossRef]
7. Mangasarian, O.L. Linear complementarity as absolute value equation solution. *Optim. Lett.* **2014**, *8*, 1529–1534. [CrossRef]
8. Prokopyev, O. On equivalent reformulations for absolute value equations. *Comput. Optim. Appl.* **2009**, *44*, 363–372. [CrossRef]
9. Lotfi, T.; Veiseh, H. A note on unique solvability of the absolute value equation. *J. Lin. Top. Alg.* **2013**, *2*, 77–81.
10. Mangasarian, O.L.; Meyer, R.R. Absolute value equations. *Linear Algebra Appl.* **2006**, *419*, 359–367. [CrossRef]
11. Rohn, J. On unique solvability of the absolute value equation. *Optim. Lett.* **2009**, *3*, 603–606. [CrossRef]
12. Rohn, J.; Hooshyarbakhsh, V.; Farhadsefat, R. An iterative method for solving absolute value equations and sufficient conditions for unique solvability. *Optim. Lett.* **2014**, *8*, 35–44. [CrossRef]
13. Hu, S.L.; Huang, Z.H. A note on absolute value equations. *Optim. Lett.* **2010**, *4*, 417–424. [CrossRef]
14. Abdallah, L.; Haddou, M.; Migot, T. Solving absolute value equation using complementarity and smoothing functions. *J. Comput. Appl. Math.* **2018**, *327*, 196–207. [CrossRef]
15. Caccetta, L.; Qu, B.; Zhou, G.L. A globally and quadratically convergent method for absolute value equations. *Comput. Optim. Appl.* **2011**, *48*, 45–58. [CrossRef]
16. Li, C.X. A preconditioned AOR iterative method for the absolute value equations. *Int. J. Comput. Methods* **2017**, *14*, 1750016. [CrossRef]
17. Yong, L. Particle swarm optimization for absolute value equations. *J. Comput. Informat. Syst.* **2010**, *6*, 2359–2366.
18. Moosaei, H.; Ketabchi, S.; Noor, M.A.; Iqbald, J.; Hooshyarbakhshe, V. Some techniques for solving absolute value equations. *Appl. Math. Comput.* **2015**, *268*, 696–705. [CrossRef]
19. Wu, S.; Li, C. A special shift splitting iteration method for absolute value equation. *Aims Math.* **2020**, *5*, 5171–5183. [CrossRef]
20. Yong, L. Iteration method for absolute value equation and applications in two-point boundary value problem of linear differential equation. *J. Interdiscip. Math.* **2014**, *18*, 355–374. [CrossRef]
21. Ketabchi, S.; Moosaei, H. Minimum norm solution to the absolute value equation in the convex case. *J. Optim. Theory Appl.* **2012**, *154*, 1080–1087. [CrossRef]
22. Ketabchi, S.; Moosaei, H.; Fallahi, S. Optimal error correction of the absolute value equation using a genetic algorithm. *Math. Comput. Model.* **2013**, *57*, 2339–2342. [CrossRef]
23. Liu, J.G.; Zhu, W.H.; Wu, Y.K.; Jin, G.H. Application of multivariate bilinear neural network method to fractionalpartial differential equations. *Results Phys.* **2023**, *47*, 106341. [CrossRef]
24. Taylor, J.M.; Pardo, D.; Muga, I. A deep fourier residual method for solving PDEs using neural networks. *Comput. Methods Appl. Mech. Eng.* **2023**, *405*, 115850. [CrossRef]
25. Rohn, J. An algorithm for solving the absolute value equations. *Electron. J. Linear Algebra* **2009**, *18*, 589–599. [CrossRef]
26. Nesterov, Y. Smooth minimization of nonsmooth functions. *Math. Program. Ser. A* **2005**, *103*, 127–152. [CrossRef]
27. Haddou, M.; Migot, T.; Omer, J. A generalized direction in interior point method for monotone linear complementarity problems. *Optim. Lett.* **2019**, *13*, 35–53. [CrossRef]
28. Alcantara, J.H.; Chen, J.-S. A new class of neural networks for NCPs using smooth perturbations of the natural residual function. *J. Comput. Appl. Math.* **2022**, *407*, 114092. [CrossRef]
29. Fakharzadeh, A.J.; Shams, N.N. An Efficient Algorithm for Solving Absolute Value Equations. *J. Math. Ext.* **2021**, *15*, 1–23.
30. Shindo, M.K.S. Extension of Newton and quasi-Newton methods to systems of PC1 equations. *J. Oper. Res. Soc. Jpn.* **1986**, *29*, 352–374.
31. Dolan, E.D.; More, J.J. Benchmarking optimization software with performance profiles. *Math. Profiles* **2002**, *91*, 201–213. [CrossRef]