axioms

MDPI

Article

Convergence-Enhanced and ANN-Accelerated Solvers for Absolute Value Problems

Mudassir Shams [1,2] **and Bruno Carpentieri** [2,*]

[1] Department of Mathematics, Faculty of Arts and Science, Balikesir University, 10145 Balıkesir, Turkey; mudassir.shams@balikesir.edu.tr
[2] Faculty of Engineering, Free University of Bozen-Bolzano, 39100 Bolzano, Italy
* Correspondence: bruno.carpentieri@unibz.it

Abstract

Absolute value problems of the form $Ax - |x| = b$, where $x \in \mathbb{R}^n$ is the unknown vector, $b \in \mathbb{R}^n$ is a given vector, and $A \in \mathbb{R}^{n\times n}$ is a matrix, arise in a wide range of scientific and engineering applications. Their solution is challenging due to the non-differentiability of the absolute value operator and the possible existence of multiple solutions. Classical iterative techniques often suffer from slow convergence, strong sensitivity to the choice of initial vectors, and limited global convergence guarantees. In this study, we introduce a novel two-step iterative scheme that incorporates an adaptive initialization strategy enhanced by artificial neural networks (ANNs). The proposed method attains global linear convergence and local third-order convergence, thereby combining robustness with high accuracy. Numerical experiments on a range of benchmark problems—including cases with both unique and multiple solutions—demonstrate that the ANN-assisted initialization substantially accelerates convergence. In particular, it reduces the number of iterations, computational time, and residual errors across multiple norms, including both the Euclidean and infinity norms. These findings demonstrate that coupling a high-order two-step solver with ANN-based adaptive initialization yields a reliable and efficient framework for solving absolute value problems in both theoretical analysis and practical large-scale applications.

Keywords: absolute value problems; iterative methods; adaptive initialization; artificial neural networks; local and global convergence; numerical stability

MSC: 65K10; 90C30; 65F10; 65B99

Academic Editor: Roman Dmytryshyn

Received: 5 October 2025
Revised: 21 November 2025
Accepted: 25 November 2025
Published: 28 November 2025

Citation: Shams, M.; Carpentieri, B. Convergence-Enhanced and ANN-Accelerated Solvers for Absolute Value Problems. *Axioms* **2025**, *14*, 880. https://doi.org/10.3390/axioms14120880

1. Introduction

Absolute Value Equations (AVEs) of the form

$$Ax - |x| = b, \quad A \in \mathbb{R}^{n\times n},\ x, b \in \mathbb{R}^n, \tag{1}$$

where $|x|$ denotes the componentwise absolute value of x, have emerged as a central class of problems in applied mathematics, computer science, and engineering. AVEs unify several nonlinear structures and are closely related to linear complementarity problems (LCPs), variational inequalities (VIs), piecewise-linear systems, and various optimization models [1,2]. Their study has attracted considerable attention both for their wide range of applications and for the mathematical challenges posed by their inherent nonsmoothness.

The significance of AVEs lies in their ability to model diverse real-world phenomena across different disciplines. In operations research, they arise as reformulations of opti-

mization problems with complementarity constraints [3]. In control theory, they represent switching systems and piecewise-linear controllers [4]. In structural engineering and mechanics, they naturally capture contact problems and frictional forces, with the absolute value function modeling transitions between compression and tension. In biomedical engineering, AVEs describe threshold-driven dynamics such as glucose–insulin regulation, cardiac electrophysiology, and biomechanical load–response systems [5,6]. In computer science and machine learning, they appear in sparsity-inducing regularizations, signal recovery, and robust regression models [7]. Because of this breadth, efficient solution methods for AVEs are not only of mathematical interest but also crucial for handling larger systems where absolute value operators introduce discontinuities and nonsmooth behavior.

Early approaches to solving AVEs were based on exact transformations, but they suffer from several fundamental limitations. The piecewise-linear structure of the absolute value function allows reformulation into up to 2^n linear systems [8], yet this quickly becomes infeasible as n increases. Reformulations into linear complementarity problems (LCPs) and variational inequalities (VIs) [9] permit the use of established algorithms but at the cost of increased dimensionality and complexity. Symbolic and perturbation methods [10] provide valuable theoretical insight into existence, uniqueness, and sensitivity, but they do not scale to large systems. Overall, these approaches are limited in their applicability to high-dimensional or sparse problems and offer little practical utility for real-world computations.

Because of these drawbacks, iterative solvers have become indispensable. Methods such as semi-smooth Newton approaches [11,12], SOR schemes [13], and projection algorithms [14] are widely used for their scalability, simplicity, and ability to exploit sparsity in large-scale AVEs. For engineering problems with tens of thousands of variables, iterative methods are often the only viable option, owing to their adaptability and parallelizability [15]. Many existing iterative schemes stem from quadrature-based integral reformulations [16], but these approaches suffer from well-known drawbacks:

1. *Stability:* Quadrature-based schemes are highly sensitive to discretization parameters, often leading to unstable convergence.
2. *Convergence Rate:* Most quadrature-derived methods achieve at best linear or quadratic convergence [17].
3. *Robustness:* Their performance depends on accurate numerical integration, which becomes problematic for the nonsmooth functions inherent in AVEs.
4. *Computational Cost:* In multidimensional problems, quadrature rules require evaluating a large number of integrand points, leading to significant overhead.

These limitations underscore the necessity of developing iterative methods that avoid reliance on quadrature rules and instead exploit the intrinsic algebraic and numerical properties of AVEs. Over the past two decades, research in this direction has accelerated considerably. Rohn [18] and Mangasarian [19] established foundational existence and uniqueness results, which motivated the development of semi-smooth Newton methods, while projection and splitting strategies such as Picard iteration and successive approximation schemes have also been studied extensively [20,21]. More recent research has focused on enhancements including preconditioning [22], extrapolation techniques [23], adaptive step strategies [24], and acceleration via artificial neural networks (ANNs) [25]. Nevertheless, most existing methods remain constrained by low convergence order, stability concerns, and the absence of global convergence guarantees, while ANN-based hybridizations are still in an early stage of development.

Despite the breadth of existing techniques, a closer examination reveals several unresolved limitations that motivate the present work. Classical iterative strategies for AVEs—including semi-smooth Newton methods [26], projection–splitting techniques [27],

and quadrature-based updates [28]—often exhibit either a low order of convergence or a strong sensitivity to the nonsmooth regions of the solution space. When absolute-value nonlinearities induce switching, discontinuities, or ill-conditioning, even methods with higher local convergence rates typically depend on structural assumptions that restrict their robustness. Recent ANN-based enhancements [29] improve the prediction of descent directions or step sizes, yet such hybrid approaches seldom integrate the network output directly into the algebraic structure of the update, limiting both their stability and theoretical guarantees. These gaps highlight the need for algorithms that (i) retain the simplicity and low computational cost of classical two-step schemes, (ii) achieve higher practical efficiency without compromising stability, and (iii) incorporate ANN-generated corrections in a mathematically controlled manner. Motivated by these challenges, we modify an existing locally third-order two-step method and embed an ANN-driven correction term into its iterative map, producing a hybrid scheme that improves global behavior while preserving provable local convergence properties [30].

In this work, we refine a classical two-step method for AVEs that possesses local third-order and global linear convergence, with the aim of improving its stability and enlarging its region of convergence. We further introduce a hybrid ANN strategy in which network-predicted correction terms are embedded directly into the update, thereby accelerating convergence without altering the method's underlying theoretical framework. Both the local analysis and extensive numerical experiments indicate that the ANN-enhanced scheme achieves higher efficiency, greater robustness with respect to difficult initial guesses, and superior performance on large-scale and nonsmooth AVEs.

1.1. Main Contributions

The main contributions of this paper can be summarized as follows:

1. A two-step iterative method is developed, achieving global linear convergence and local quartic convergence.
2. Integration of ANN-based acceleration.
3. A comprehensive computational analysis evaluating convergence behavior, runtime efficiency, and memory utilization.
4. Extensive numerical validation is carried out on benchmark AVEs, large-scale systems, initial/boundary value AVEs, and biomechanical applications.
5. A conceptual bridge is established between deterministic iterative theory and AI-based accelerators.

1.2. Structure of the Paper

- Section 2: Construction of the proposed scheme and analysis of its convergence properties.
- Section 3: Numerical methodology, implementation details, and ANN-based acceleration strategy.
- Section 4: Numerical results for four categories:
  1. benchmark AVEs,
  2. large-scale systems,
  3. initial and boundary value AVEs,
  4. biomechanical applications.
- Section 5: Conclusions and future research directions.

2. Construction and Convergence Analysis of the Proposed Scheme

We introduce the notation and background required for the analysis of AVEs. Let e denote a unit vector and I the identity matrix of appropriate dimension. The Euclidean

norm of a vector x is defined as $\|x\| = \sqrt{x^T x}$. The function $\text{sign}(x)$ represents a vector whose components take the values 1, 0, or -1 depending on whether the corresponding entry of x is positive, zero, or negative, respectively. Furthermore, $\text{diag}(\text{sign}(x))$ denotes the diagonal matrix with the components of $\text{sign}(x)$ on its diagonal. A generalized Jacobian of the absolute value function $|x|$ at x can then be expressed as $D(x) = \text{diag}(\text{sign}(x))$.

Based on these definitions, we consider the nonlinear function

$$f(x) = Ax - |x| - b, \tag{2}$$

where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$. The generalized Jacobian of $f(x)$ at x is given by

$$f'(x) = A - D(x), \quad D(x) = \text{diag}(\text{sign}(x)). \tag{3}$$

To solve absolute value equations (AVEs) of the form $Ax - |x| = b$, the classical generalized Newton-type iteration is commonly employed [31]:

$$x^{[k+1]} = \left(A - D(x^{[k]})\right)^{-1} b, \quad k = 0, 1, 2, \ldots, \tag{4}$$

where $D(x^{[k]}) = \text{diag}(\text{sign}(x^{[k]}))$. This iteration serves as the foundation for several higher-order and accelerated schemes.

A higher-order variant, based on a two-step Trube-type method [32], is defined as

$$\begin{cases} y^{[k]} = x^{[k]} - \left(A - D(x^{[k]})\right)^{-1} f(x^{[k]}), \\ x^{[k+1]} = x^{[k]} - \left(A - D(x^{[k]})\right)^{-1} \left(f(y^{[k]}) + f(x^{[k]})\right), \end{cases} \tag{5}$$

where $f(x) = Ax - |x| - b$. This two-step scheme achieves global linear convergence and local quadratic convergence, requiring at each iteration the solution of two linear systems and one LU factorization (or Jacobian evaluation). We refer to this method as TAV.

Another two-step method was proposed in ref. [33], formulated as

$$\begin{cases} y^{[k]} = x^{[k]} - \left(A - D(x^{[k]})\right)^{-1} f(x^{[k]}), \\ x^{[k+1]} = y^{[k]} - \left(A - D(x^{[k]})\right)^{-1} f(y^{[k]}), \end{cases} \tag{6}$$

which attains global linear and local quadratic convergence with computational requirements similar to the previous scheme.

In addition, predictor–corrector strategies employing quadrature-based acceleration have been proposed. For instance, Khan et al. [34] combine the classical Newton predictor with Simpson's rule as a corrector:

$$\begin{cases} y^{[k]} = x^{[k]} - \left(A - D(x^{[k]})\right)^{-1} f(x^{[k]}), \\ x^{[k+1]} = x^{[k]} - 6\left(f'^{[k]} + 4f'\left(\frac{x^{[k]}+y^{[k]}}{2}\right) + f'^{[k]}\right)^{-1} f(x^{[k]}), \end{cases} \tag{7}$$

which achieves global linear and local quadratic convergence. This method, abbreviated as AEV, requires one nonlinear function evaluation and three Jacobian or LU factorizations per iteration.

Similarly, Rahpeymaii et al. [35] employ the Simpson 3/8 rule:

$$\begin{cases} y^{[k]} = x^{[k]} - \left(A - D(x^{[k]})\right)^{-1} f(x^{[k]}), \\ x^{[k+1]} = x^{[k]} - 8\left(f'^{[k]} + 3f'\left(\frac{2x^{[k]}+y^{[k]}}{3}\right) + 3f'\left(\frac{x^{[k]}+2y^{[k]}}{3}\right) + f'^{[k]}\right)^{-1} f(x^{[k]}), \end{cases} \tag{8}$$

which likewise achieves global linear and local quadratic convergence. Each iteration requires two linear system solves and four Jacobian evaluations or LU factorizations, and we denote this method by HAV.

Finally, a three-step method [36] is given by

$$\begin{cases} y^{[k]} = x^{[k]} - \left(A - D(x^{[k]})\right)^{-1} f(x^{[k]}), \\ z^{[k]} = y^{[k]} - \frac{\left(A-D(x^{[k]})\right)^{-1} + \left(A-D(y^{[k]})\right)^{-1}}{2} f(y^{[k]}), \\ x^{[k+1]} = z^{[k]} - \left(A - D(z^{[k]})\right)^{-1} f(z^{[k]}), \end{cases} \tag{9}$$

which attains global linear and local quadratic convergence, with two linear system solves and three Jacobian/LU factorizations per iteration. We denote this scheme by FAV.

2.1. Development and Analysis of Numerical Schemes for Solving AVEs

We build on the third-order method proposed in [37] for solving nonlinear equations, which is defined by

$$\begin{cases} y^{[k]} = x^{[k]} - \dfrac{f(x^{[k]})}{f'(x^{[k]})}, \\ x^{[k+1]} = x^{[k]} - \dfrac{2f(y^{[k]})}{3f'(y^{[k]}) - f'(x^{[k]})} \dfrac{f(x^{[k]})}{f'(x^{[k]})}. \end{cases} \tag{10}$$

where the errors are given by $e^{[k]} = x^{[k]} - \xi$ and $e^{[k+1]} = x^{[k+1]} - \xi$. The associated local error equation is

$$e^{[k+1]} = \tfrac{1}{2}\left(-2d_2^2 + d_3\right)\left(e^{[k]}\right)^3 + \mathcal{O}\left((e^{[k]})^4\right), \qquad d_j = \frac{f^{(j)}(\xi)}{j!\, f'(\xi)},\ j \geq 2.$$

To extend this approach to absolute value equations (AVEs), the method is generalized to the following form, referred to as the SMV method:

$$\begin{cases} y^{[k]} = x^{[k]} - (A - D(x^{[k]}))^{-1} f(x^{[k]}), \\ x^{[k+1]} = x^{[k]} - \dfrac{2f(y^{[k]})}{3f'(y^{[k]}) - f'(x^{[k]})} \dfrac{f(x^{[k]})}{f'(x^{[k]})}. \end{cases} \tag{11}$$

2.1.1. Convergence Analysis

We present a rigorous convergence analysis of the proposed iterative method. First, we show that each iteration is well defined; next, we establish the boundedness of the generated sequence; finally, we prove the main convergence theorem.

Well-Definedness of the Scheme

We recall that the generalized Newton method (Lemma 2, [38]) can be employed as a predictor step:

$$y^{[k]} = x^{[k]} - \left(A - D(x^{[k]})\right)^{-1} f(x^{[k]}). \tag{12}$$

Based on (12), the proposed scheme is written as

$$x^{[k+1]} = x^{[k]} - 2f'(y^{[k]})M_2^{-1}u^{[k]}, \tag{13}$$

where

$$M_2 = 3f'(y^{[k]}) - f'(x^{[k]}), \qquad u^{[k]} = \frac{f(x^{[k]})}{f'(x^{[k]})}.$$

To ensure that the sequence $\{x^{[k]}\}$ is well defined, we observe that the predictor step (12) is already established as bounded (see ref. [39]). It therefore remains to verify the nonsingularity of

$$\begin{aligned} 3f'(y^{[k]}) - f'(x^{[k]}) &= 3A - 3D(y^{[k]}) - A + D(x^{[k]}) \\ &= 2A - 3D(y^{[k]}) + D(x^{[k]}). \end{aligned} \tag{14}$$

Lemma 1. *If the singular values of $A \in \mathbb{R}^{n\times n}$ satisfy $\sigma_{\min}(A) > 1$, then the diagonal matrix*

$$\left(2A - 3D(y^{[k]}) + D(x^{[k]})\right)^{-1}$$

exists for all diagonal matrices $D(x^{[k]})$, $D(y^{[k]})$ with entries in $\{-1, 0, 1\}$.

Proof. Assume, for contradiction, that M_2 is singular. Then there exists a nonzero vector $v \neq 0$ such that

$$M_2 v = 0. \tag{15}$$

Taking norms, we obtain

$$\begin{aligned} v^T v &< v^T A^T A v \\ &< v^T \left(2A - 3D(y^{[k]}) + D(x^{[k]})\right)^T \left(2A - 3D(y^{[k]}) + D(x^{[k]})\right) v. \end{aligned} \tag{16}$$

Expanding the diagonal contributions gives

$$v^T A^T A v = v^T \Big(4A^T A + 9D(y^{[k]})D(y^{[k]}) + D(x^{[k]})D(x^{[k]}) + \text{cross terms}\Big) v. \tag{17}$$

Since for any diagonal matrix with entries in $\{-1, 0, 1\}$ it holds that $\|D(\cdot)\| \leq 1$, we deduce

$$v^T A^T A v \;\geq\; 4 v^T v. \tag{18}$$

This yields the contradiction

$$v^T v \;<\; \tfrac{1}{4} v^T A^T A v \;<\; v^T v, \tag{19}$$

which is impossible whenever $\sigma_{\min}(A) > 1$. Therefore, M_2 is nonsingular. □

2.2. Boundedness of the Sequence

We now establish the boundedness of the sequence $\{x^{[k]}\}$. The corrector step can be expressed as

$$x^{[k+1]} = x^{[k]} - 2f'(y^{[k]})\Big(3f'(y^{[k]}) - f'(x^{[k]})\Big)^{-1} u^{[k]}, \tag{20}$$

or, equivalently,

$$x^{[k+1]} = x^{[k]} - N^{[k]}(M_2)^{-1} f(x^{[k]}), \tag{21}$$

where

$$N^{[k]} = \frac{2f'(y^{[k]})}{f'(x^{[k]})}.$$

Hence,

$$\begin{aligned} M_2x^{[k+1]} &= M_2x^{[k]} - N^{[k]}f(x^{[k]}) \\ &= M_2\Big(2A - 3D(y^{[k]}) + D(x^{[k]})\Big) - N^{[k]}\Big(Ax^{[k]} - |x^{[k]}| - b\Big). \end{aligned}$$

The right-hand side is a linear combination of $x^{[k]}$, b, and the diagonal matrices $D(x^{[k]})$ and $D(y^{[k]})$. Since the entries of these diagonal matrices are bounded in absolute value by 1, it follows that the sequence $\{x^{[k]}\}$ remains bounded (see Proposition 3 in ref. [40]).

2.3. Main Convergence Result

We now present the principal convergence theorem.

Theorem 1. *If*

$$\left\|2A - 3D(y^{[k]}) + D(x^{[k]})\right\| < \tfrac{1}{4}, \tag{22}$$

for any diagonal matrix with entries from $\{-1, 0, 1\}$*, then the sequence generated by the scheme converges to the solution* ζ *of the AVE.*

Proof. Let $e^{[k]} = x^{[k]} - \zeta$ denote the error at the kth iteration, where ζ is the exact solution of the AVE. Since ζ satisfies

$$\begin{aligned} f(\zeta) &= A\zeta - |\zeta| - b \\ &= 0, \end{aligned} \tag{23}$$

we can express the iteration error as

$$x^{[k+1]} - \zeta = x^{[k]} - \zeta - N_1^{[k]}M_2^{-1}u^{[k]}. \tag{24}$$

Multiplying both sides by M_2 yields

$$M_2(x^{[k+1]} - \zeta) = M_2(x^{[k]} - \zeta) - N_1^{[k]}f(x^{[k]}). \tag{25}$$

By definition of the scheme, we obtain

$$\begin{aligned} M_2(x^{[k+1]} - \zeta) &= M_2(x^{[k]} - \zeta) - 2N_1^{[k]}f(x^{[k]}) + 2N_1^{[k]}f(\zeta) \\ &= M_2(x^{[k]} - \zeta) - 2N_1^{[k]}\big(f(x^{[k]}) - f(\zeta)\big) \\ &= M_2(x^{[k]} - \zeta) - 2N_1^{[k]}\big(A(x^{[k]} - \zeta) - (|x^{[k]}| - |\zeta|)\big) \\ &= M_2(x^{[k]} - \zeta) - 2N_1^{[k]}A(x^{[k]} - \zeta) + 2N_1^{[k]}(|x^{[k]}| - |\zeta|) \\ &\leq (M_2 - 2A)(x^{[k]} - \zeta) + 2(|x^{[k]}| - |\zeta|) \\ &= \big(-3D(y^{[k]}) + D(x^{[k]})\big)(x^{[k]} - \zeta) + 2(|x^{[k]}| - |\zeta|). \end{aligned} \tag{26}$$

Therefore, the error satisfies

$$x^{[k+1]} - \zeta \leq M_2^{-1}\Big[2(|x^{[k]}| - |\zeta|) + \big(-3D(y^{[k]}) + D(x^{[k]})\big)(x^{[k]} - \zeta)\Big]. \tag{27}$$

Introducing $h^{[*]} = -3D(y^{[k]}) + D(x^{[k]})$, this inequality becomes

$$x^{[k+1]} - \zeta \leq M_2^{-1}\big[2(|x^{[k]}| - |\zeta|) + h^{[*]}(x^{[k]} - \zeta)\big]. \tag{28}$$

Taking norms on both sides and using the Lipschitz continuity of the absolute value function (see Lemma 5 in ref. [41]), we obtain

$$\| \, |x^{[k]}| - |\zeta| \, \| \leq 2\|x^{[k]} - \zeta\|. \tag{29}$$

Consequently,

$$\|x^{[k+1]} - \zeta\| \leq \|M_2^{-1}\|\big(4\|x^{[k]} - \zeta\| + \|h^{[*]}\|\|x^{[k]} - \zeta\|\big). \tag{30}$$

Since both $D(x^{[k]})$ and $D(y^{[k]})$ are diagonal matrices, it follows that

$$\begin{aligned} \|h^{[*]}\| &= \| -3D(y^{[k]}) + D(x^{[k]})\| \\ &\leq 3\|D(y^{[k]})\| + \|D(x^{[k]})\| \leq 2. \end{aligned} \tag{31}$$

Combining (30) and (31), we obtain

$$\|x^{[k+1]} - \zeta\| \leq 4\|M_2^{-1}\| \, \|x^{[k]} - \zeta\| < \|x^{[k]} - \zeta\|. \tag{32}$$

Finally, since $\|M_2^{-1}\| < \frac{1}{4}$, it follows that the iterative scheme produces a globally linearly convergent sequence. □

2.4. Unique Solvability

Lemma 2. *Let $\|A^{-1}\| < \frac{1}{5}$ and $\|3D(y^{[k]})\|, \|D(x^{[k]})\| \neq 0$. Then the AVE admits a unique solution for any b, and the proposed method is well defined and convergent for every initial vector $x^{[0]}$.*

Proof. The unique solvability of the AVE for any right-hand side b requires that

$$\|A^{-1}\| < 1. \tag{33}$$

Since A^{-1} exists and, in addition, the inequality

$$\|2A^{-1}\| \, \|3D(y^{[k]}) - D(x^{[k]})\| < 1 \tag{34}$$

holds, we can apply the Banach perturbation lemma [42].

From this lemma, it follows that

$$\begin{aligned} \left\|\left(3D(y^{[k]}) - D(x^{[k]})\right)^{-1}\right\| &= \left\|\left(2A - 3D(y^{[k]}) + D(x^{[k]})\right)^{-1}\right\| \\ &\leq \frac{\|(2A)^{-1}\| \, \|3D(y^{[k]}) - D(x^{[k]})\|}{1 - \|(2A)^{-1}\| \, \|3D(y^{[k]}) - D(x^{[k]})\|}. \end{aligned} \tag{35}$$

Using $\|(2A)^{-1}\| = \frac{1}{2}\|A^{-1}\|$ and the assumption $\|A^{-1}\| < \frac{1}{5}$, we obtain

$$\begin{aligned} \left\|\left(3D(y^{[k]}) - D(x^{[k]})\right)^{-1}\right\| &\leq \frac{\frac{1}{2}\|A^{-1}\| \cdot 2}{1 - \frac{1}{2}\|A^{-1}\| \cdot 2} \\ &\leq \frac{\frac{1}{5}}{1 - \frac{1}{5}} \\ &< \frac{1}{4}. \end{aligned} \tag{36}$$

Therefore, the matrix $\left(2A - 3D(y^{[k]}) + D(x^{[k]})\right)^{-1}$ is bounded by $\frac{1}{4}$. By Theorem 1, this bound guarantees that the proposed scheme converges linearly to the unique solution of (1). □

Remark 1. *The assumptions* $\|A^{-1}\| < \frac{1}{5}$ *and* $\|M_2^{-1}\| < \frac{1}{4}$ *are introduced solely as sufficient conditions for verifying the hypotheses of the Banach perturbation lemma (invertibility and contractivity), thereby ensuring uniqueness and convergence within the theoretical analysis. These bounds, however, are not necessary. As illustrated by the numerical results (Tables 1–19), the SMV method continues to converge reliably even when these conditions are not satisfied, indicating that its practical domain of convergence is substantially broader than what the theory guarantees.*

Table 1. Comparison of iterative approximations and residual errors for the schemes TAV, AEV, FAV, HAV, and SMV applied to $f(x)$ in Example 1.

Scheme	X_1	X_2	X_3	X_4	Residual Error
TAV	$x^{[1]} = 1.043$ $x^{[2]} = 1.0702$ $x^{[3]} = 1.070820$	$x^{[1]} = 0.98$ $x^{[2]} = 0.980$ $x^{[3]} = 0.98006382$	$x^{[1]} = 1.0543$ $x^{[2]} = 1.056242$ $x^{[3]} = 1.056277454$	$x^{[1]} = 1.0722$ $x^{[2]} = 1.0708263$ $x^{[3]} = 1.07082034$	4.3×10^{-1} 6.7×10^{-7} 9.1×10^{-16}
AEV	$x^{[1]} = 1.045$ $x^{[2]} = 1.07023$ $x^{[3]} = 1.070820$	$x^{[1]} = 0.98$ $x^{[2]} = 0.980$ $x^{[3]} = 0.98006382$	$x^{[1]} = 1.0564$ $x^{[2]} = 1.056234$ $x^{[3]} = 1.056277454$	$x^{[1]} = 1.0734$ $x^{[2]} = 1.0708266$ $x^{[3]} = 1.07082034$	1.24×10^{-1} 5.63×10^{-04} 3.73×10^{-13}
FAV	$x^{[1]} = 1.044$ $x^{[2]} = 1.07075$ $x^{[3]} = 1.070820$	$x^{[1]} = 0.98$ $x^{[2]} = 0.980$ $x^{[3]} = 0.98006382$	$x^{[1]} = 1.0575$ $x^{[2]} = 1.056234$ $x^{[3]} = 1.056277454$	$x^{[1]} = 1.0743$ $x^{[2]} = 1.0708244$ $x^{[3]} = 1.07082034$	1.25×10^{-2} 0.53×10^{-09} 0.34×10^{-20} †
HAV	$x^{[1]} = 1.0447$ $x^{[2]} = 1.070344$ $x^{[3]} = 1.070820$	$x^{[1]} = 0.9834$ $x^{[2]} = 0.980834$ $x^{[3]} = 0.98006382$	$x^{[1]} = 0.981$ $x^{[2]} = 0.98063$ $x^{[3]} = 0.98006382$	$x^{[1]} = 1.0734$ $x^{[2]} = 1.070826$ $x^{[3]} = 1.07082034$	1.25×10^{-1} 0.76×10^{-7} 0.35×10^{-19} †
SMV	$x^{[1]} = 1.0447$ $x^{[2]} = 1.070344$ $x^{[3]} = 1.070820$	$x^{[1]} = 0.9853$ $x^{[2]} = 0.98046$ $x^{[3]} = 0.98006382$	$x^{[1]} = 0.9843$ $x^{[2]} = 0.980447$ $x^{[3]} = 0.98006382$	$x^{[1]} = 1.074$ $x^{[2]} = 1.070826$ $x^{[3]} = 1.07082034$	1.29×10^{-4} 0.08×10^{-18} † 0.10×10^{-27} †

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-30}.

Table 2. Performance of the ANN-accelerated SMV scheme (SMV-AN) applied to Example 1, reporting iterations, mean squared error (MSE), CPU time, memory usage, gradient magnitude, and adaptive parameter μ.

Scheme	Epoch	MSE	CPU Time	Memory (Kb)	Gradient	μ
SMV-AN	200	4.178×10^{-5}	3.4354	23,048	1.23×10^{-7}	1.08×10^{-3}

Table 3. Performance evaluation of SMV-AN for Instants 0–3 in Example 1.

Scheme	Instant 0	Instant 1	Instant 2	Instant 3	CPU Time	Memory (Kb)
SMV-AN	1.031×10^{-180} †	0.742×10^{-192} †	3.947×10^{-205} †	6.718×10^{-218} †	0.029	28.671

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-300}.

Table 4. Performance comparison of iterative schemes (TAV, AEV, FAV, HAV, SMV) applied to Example 2, reporting iteration count (It), maximum error (Max-E), CPU time, memory usage, function evaluations (Func-E), and LU factorizations.

Scheme	It	Max-E	CPU-Time	Memory (Kb)	Func-E	LU
TAV	16	4.101×10^{-125} †	13.3001	22,765	41	23
AEV	16	2.159×10^{-115} †	12.0556	26,547	37	21
FAV	16	3.150×10^{-111} †	16.1876	35,476	45	27
HAV	16	9.478×10^{-107} †	13.0012	19,557	37	17
SMV	16	5.167×10^{-163} †	8.1554	15,143	30	13

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-300}.

Table 5. Randomly generated initial vectors $X_i^{[0]}$ used for ANN training (70%) and testing/validation (30%) in Example 2.

$X_i^{[0]}$	$x_1^{[0]}$	$x_2^{[0]}$	$x_3^{[0]}$	$x_4^{[0]}$	$x_5^{[0]}$	$x_6^{[0]}$	$x_7^{[0]}$	$x_8^{[0]}$	$x_9^{[0]}$	$x_{10}^{[0]}$	$\cdots$	$x_{1000}^{[0]}$
$X_1^{[0]}$	0.202	0.7675	0.2435	0.3048	0.6550	0.1098	0.1098	0.5002	0.9283	0.1828	$\cdots$	0.4798
$X_2^{[0]}$	0.193	0.8266	0.3464	0.2567	0.3978	0.3660	0.3853	0.7587	0.6200	0.3246	$\cdots$	0.1391
$X_3^{[0]}$	0.520	0.8566	0.7853	0.3323	0.4000	0.1320	0.4774	0.3528	0.2641	0.7817	$\cdots$	0.2772
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$

Table 6. Performance of the ANN-accelerated SMV scheme (SMV-AN) applied to Example 2, reporting iterations, mean squared error (MSE), CPU time, memory usage, gradient magnitude, and adaptive parameter μ.

Scheme	Epoch	MSE	CPU Time	Memory (Kb)	Gradient	μ
SMV-AN	200	5.065×10^{-7}	2.1903	24,197	7.16×10^{-6}	0.09×10^{-4}

Table 7. Numerical outcomes of the hybride scheme SMV-AN for various instant 0–3 for solving AVE used in Example 2.

Scheme	Instant 0	Instant 1	Instant 2	Instant 3	CPU Time	Memory (Kb)
SMV-AN	4.178×10^{-195} †	0.151×10^{-178} †	1.473×10^{-190} †	0.135×10^{-201} †	0.023	28.873

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-300}.

Table 8. Performance comparison of iterative schemes (TAV, AEV, FAV, HAV, SMV) applied to Example 3, reporting iteration count (It), maximum error (Max-E), CPU time, memory usage, function evaluations (Func-E), and LU factorizations.

Scheme	It	Max-E	CPU-Time	Memory (Kb)	Func-E	LU
TAV	16	4.178×10^{-126} †	12.4354	21,018	40	21
AEV	16	3.009×10^{-117} †	11.3464	25,675	35	19
FAV	16	8.056×10^{-109} †	15.7853	33,234	40	20
HAV	16	1.100×10^{-113} †	14.3436	19,875	39	19
SMV	16	9.590×10^{-165} †	7.4426	16,030	32	16

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-300}.

Table 9. Randomly generated initial vectors $X_i^{[0]}$ used for ANN training (70%) and testing/validation (30%) for the strongly nonsmooth AVE in Example 3.

$X_i^{[0]}$	$x_1^{[0]}$	$x_2^{[0]}$	$x_3^{[0]}$	$x_4^{[0]}$	$x_5^{[0]}$	$x_6^{[0]}$	$x_7^{[0]}$	$x_8^{[0]}$	$x_9^{[0]}$	$x_{10}^{[0]}$	$\cdots$	$x_n^{[0]}$
$X_1^{[0]}$	1.02	−0.55	0.98	1.05	−1.12	0.49	−0.47	1.15	−0.62	0.85	$\cdots$	0.33
$X_2^{[0]}$	−0.88	1.11	−0.53	0.75	1.09	−0.67	0.91	−1.03	0.44	0.67	$\cdots$	−0.28
$X_3^{[0]}$	0.45	−0.97	1.08	−0.88	0.72	−1.15	1.02	0.56	−0.49	1.14	$\cdots$	0.41
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$

Table 10. Performance of the ANN-accelerated SMV scheme (denoted SMV-AN) for Example 3, reporting iteration count, mean squared error (MSE), CPU time, memory usage, gradient magnitude, and adaptive learning parameter μ.

Scheme	Epoch	MSE	CPU Time	Memory (Kb)	Gradient	μ
SMV-AN	200	3.059×10^{-2}	2.1003	22,147	5.08×10^{-3}	4.67×10^{-5}

Table 11. Error profile of the SMV-AN method for solving AVE at different iterations (Example 3).

Scheme	Instant 0	Instant 1	Instant 2	Instant 3	CPU Time	Memory (Kb)
SMV-AN	9.887×10^{-191} †	1.227×10^{-201} †	6.411×10^{-214} †	3.138×10^{-229} †	0.025	29.002

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits = 64`, employing a numerical tolerance of 10^{-300}.

Table 12. Performance comparison of iterative schemes (TAV, AEV, FAV, HAV, SMV) for Example 4, evaluated across different discretization levels (D). Metrics include iteration count (It), computational time (C-Time), memory usage (Mem), and maximum error (Max-Err).

Scheme	D	32	64	256	1024	4096
TAV	It	13	13	13	13	14
	CPU Time	0.02341	0.04543	0.056574	0.2657	1.01232
	Memory (Kb)	2543	2675	2897	3160	3308
	Max-Err	1.05×10^{-9}	0.98×10^{-9}	0.13×10^{-9}	0.87×10^{-9}	0.65×10^{-9}
AEV	It	13	13	13	13	13
	CPU Time	0.01324	0.02356	0.0678	0.49755	1.8432
	Memory (Kb)	2453	2564	26876	3098	3276
	Max-Err	1.34×10^{-10}	7.64×10^{-10}	1.64×10^{-9}	0.23×10^{-9}	0.14×10^{-9}
FAV	It	15	15	15	15	15
	CPU Time	0.01765	0.02287	0.06546	0.78598	1.6534
	Memory (Kb)	2345	2567	2786	3007	3256
	Max-Err	0.07×10^{-11}	0.15×10^{-11}	0.34×10^{-11}	1.10×10^{-11}	0.11×10^{-11}
HAV	It	12	12	12	12	12
	CPU Time	0.01254	0.05674	0.0876	0.8642	1.2376
	Memory (Kb)	2132	2435	2675	3043	31774
	Max-Err	9.13×10^{-13}	1.09×10^{-13}	0.35×10^{-13}	8.68×10^{-13}	4.09×10^{-13}
SMV	It	7	7	7	7	7
	CPU Time	0.00436	0.01435	0.03564	0.2345	0.8765
	Memory (Kb)	1034	16754	2207	2376	2578
	Max-Err	1.21×10^{-16}	5.50×10^{-16}	6.76×10^{-16}	1.93×10^{-16}	7.67×10^{-16}

Table 13. Performance of the ANN-accelerated SMV scheme (denoted SMV-AN) for Example 4, reporting iteration count, mean squared error (MSE), CPU time, memory usage, gradient magnitude, and adaptive learning parameter μ.

Scheme	Epoch	MSE	CPU Time	Memory (Kb)	Gradient	μ
SMV-AN	1000	2.031×10^{-2}	3.4354	24,129	0.14×10^{-2}	1.08×10^{-1}

Table 14. Computational performance of the ANN-accelerated SMV scheme (SMV-AN) for Example 4, evaluated across different discretization levels (D). Metrics include iteration count (It), computational time (C-Time), memory usage (Mem), and maximum error (Max-Err).

Scheme	D	32	64	256	1024	4096
SMV-AN	It	3	3	3	3	3
	CPU Time	0.00243	0.00456	0.03776	0.07866	0.15643
	Memory(Kb)	1004	1143	1234	1431	1487
	Max-Err	0.04×10^{-27} †	0.67×10^{-27} †	1.09×10^{-27} †	0.06×10^{-27} †	1.75×10^{-27} †

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-30}.

Table 15. Comparison of residual decay of SMV-AN for instants 0–3 (Example 4).

Scheme	Instant 0	Instant 1	Instant 2	Instant 3	CPU Time	Memory (Kb)
SMV-AN	3.991×10^{-27} †	2.173×10^{-28} †	4.901×10^{-29} †	9.801×10^{-25} †	0.0327	123.108

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-30}.

Table 16. Computational performance of different iterative methods (TAV, AEV, FAV, HAV, SMV) for Example 5, evaluated across various discretization levels (D). Metrics include iteration count (It), computational time (C-Time), memory usage (Mem), and maximum error (Max-Err).

Scheme	D	32	64	256	1024	4096
TAV	It	16	16	16	16	17
	CPU Time	0.0345	0.0657	0.08964	0.8797	2.65232
	Memory(Kb)	2768	2858	3007	3546	3786
	Max-Err	2.23×10^{-9}	1.86×10^{-9}	3.09×10^{-9}	1.87×10^{-9}	1.32×10^{-9}
AEV	It	17	17	17	17	17
	CPU Time	0.0456	0.0466	0.0786	0.47445	2.5766
	Memory(Kb)	2564	2653	2779	3453	3765
	Max-Err	6.04×10^{-9}	1.32×10^{-9}	0.76×10^{-9}	1.35×10^{-9}	1.05×10^{-9}
FAV	It	17	17	17	17	17
	CPU Time	0.0546	0.0677	0.0987	1.1238	2.0984
	Memory(Kb)	2765	2873	3043	3234	3986
	Max-Err	1.03×10^{-11}	7.09×10^{-11}	1.01×10^{-11}	5.98×10^{-11}	6.08×10^{-11}
HAV	It	18	18	18	18	18
	CPU Time	0.0543	0.5436	0.8854	1.5756	3.6546
	Memory(Kb)	2654	2733	3765	3876	3985
	Max-Err	0.54×10^{-11}	6.56×10^{-9}	1.09×10^{-8}	7.13×10^{-8}	6.09×10^{-8}
SMV	It	9	9	9	9	9
	CPU Time	0.03422	0.07654	0.23654	1.3255	1.9865
	Memory(Kb)	1354	2145	2456	2876	3008
	Max-Err	6.34×10^{-19} †	1.67×10^{-19} †	1.87×10^{-19} †	6.78×10^{-19} †	9.43×10^{-19} †

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-30}.

Table 17. Performance of the ANN-accelerated SMV scheme (denoted SMV-AN) for Example 5, reporting iteration count, mean squared error (MSE), CPU time, memory usage, gradient magnitude, and adaptive learning parameter μ.

Scheme	Epoch	MSE	CPU Time	Memory (Kb)	Gradient	μ
SMV-AN	1500	7.093×10^{-1}	2.0176	23,849	1.79×10^{-1}	6.91×10^{-3}

Table 18. Computational performance of the ANN-accelerated SMV scheme (SMV-AN) for the ABVP in Example 5, evaluated across different discretization levels (D). Metrics include iteration count (It), computational time (CPU Time), memory usage (Mem), and maximum error (Max-Err).

Scheme	D	32	64	256	1024	4096
SMV-AN	It	4	4	4	4	4
	CPU Time	0.00123	0.00325	0.04563	0.13242	1.4355
	Memory(Kb)	1035	1236	1435	1523	1789
	Max-Err	0.16×10^{-30} †	0.31×10^{-29} †	5.21×10^{-29} †	2.15×10^{-29} †	0.95×10^{-29} †

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-30}.

Table 19. SMV-AN method precision and performance metrics for AVE (Example 5).

Scheme	Instant 0	Instant 1	Instant 2	Instant 3	CPU Time	Memory (Kb)
SMV-AN	6.178×10^{-30} †	5.601×10^{-31} †	1.093×10^{-29} †	4.965×10^{-28} †	0.054	28.412

† All computations were performed using MATLAB variable precision arithmetic (VPA) with `digits` = 64, employing a numerical tolerance of 10^{-30}.

3. Hybrid ANN-Based Framework for Solving Absolute Value Problems (AVPs)

To enhance the robustness and scalability of numerical solvers for Absolute Value Problems (AVPs), a hybrid framework assisted by Artificial Neural Networks (ANNs) has been developed. This strategy [43] aims to generate accurate initial approximations for iterative solvers, thereby accelerating convergence and improving stability, particularly in large-scale or ill-conditioned settings. The ANN component learns mappings between AVP parameters and the corresponding solution behavior, providing an informed initialization that reduces computational cost while preserving numerical reliability. The motivation for incorporating ANN-based initialization arises from the well-known sensitivity of classical AVP solvers to poor starting points, which can lead to slow convergence or even divergence in high-dimensional or nonsmooth contexts [44]. By embedding a learning-based predictor that approximates the initial state of the solution space, the hybrid framework effectively unites classical numerical theory with modern data-driven modeling. This integration enables faster convergence, improved stability near discontinuities, and adaptability across a wide range of AVP classes, including algebraic, boundary-value, and time-dependent formulations.

3.1. Architecture Description

The Artificial Neural Network (ANN) architecture developed for the hybrid AVP solver is a fully connected feed-forward model designed to generate accurate initial approximations for iterative schemes. The network, illustrated in Figures 1 and 2, consists of three hidden layers with 128, 256, and 128 neurons, respectively. Its input layer encodes both structural and parametric characteristics of the AVP, including:

- the flattened entries of the coefficient matrix $A \in \mathbb{R}^{n\times n}$ (or, for high-dimensional systems, a low-dimensional spectral summary such as the eigenvalue distribution or a sparsity descriptor),
- the components of the right-hand side vector $b \in \mathbb{R}^n$,
- the system dimension n together with selected problem-specific hyperparameters (Figure 2).

Input Encoding. The input vector $\mathbf{z}_{\text{in}}$ supplied to the ANN is defined as

$$\mathbf{z}_{\text{in}} = \left[\operatorname{vec}(A)^\top, b^\top, n\right]^\top,$$

where $\operatorname{vec}(A)$ denotes the flattened representation of the coefficient matrix $A \in \mathbb{R}^{n\times n}$. For large-scale systems, a compressed or spectral representation of A (e.g., an eigenvalue-based summary or a sparsity descriptor) may be employed to reduce the dimensionality while retaining the essential structural features of the problem.

Hidden Layer Transformations. Each hidden layer applies an affine transformation followed by a nonlinear activation:

$$\mathbf{h}_k = \sigma(W_k \mathbf{h}_{k-1} + \mathbf{b}_k), \quad \sigma(\cdot) = \max(0, \cdot),$$

where W_k and $\mathbf{b}_k$ denote the weight matrix and bias vector of the k-th layer, respectively. The ReLU function $\sigma(\cdot)$ is particularly effective at capturing the nonsmooth dependencies introduced by the absolute value operator in the AVP formulation, while maintaining computational efficiency.

Output Layer and Prediction. The final network layer generates a continuous-valued vector

$$\hat{x} = W_{\text{out}} \mathbf{h}_3 + \mathbf{b}_{\text{out}},$$

which serves as the ANN-predicted initial approximation for the iterative solver. The linear activation in the output layer ensures that $\hat{x}$ spans the full range of real numbers, consistent with the domain of admissible AVP solutions.

Weight Initialization and Regularization. All network weights were initialized using the He-normal distribution, which helps maintain stable gradient propagation across layers. To prevent overfitting and enhance generalization—particularly in the presence of sparse or high-dimensional matrices A—L2 regularization with coefficient $\lambda = 10^{-4}$ was applied.

Loss Function and Optimization. The mean squared error (MSE) was adopted as the training objective:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|x_i - \hat{x}_i(\theta)\|^2,$$

where θ represents the set of all ANN parameters. Training was performed using the Adam optimizer, augmented with a Levenberg–Marquardt (LM) damping strategy to enhance local convergence in complex loss landscapes. The output layer produces an approximate solution vector $\hat{x} \in \mathbb{R}^n$ which is used as the initial guess for the subsequent iterative refinement. ReLU activations were employed in the hidden layers to effectively capture the nonlinear and nonsmooth dependencies inherent to AVPs, while a linear activation was used in the final layer to preserve the full continuous range of admissible outputs. A dropout rate of 0.2 was introduced to mitigate overfitting, and all weights were initialized using He initialization to promote stable gradient propagation during training.

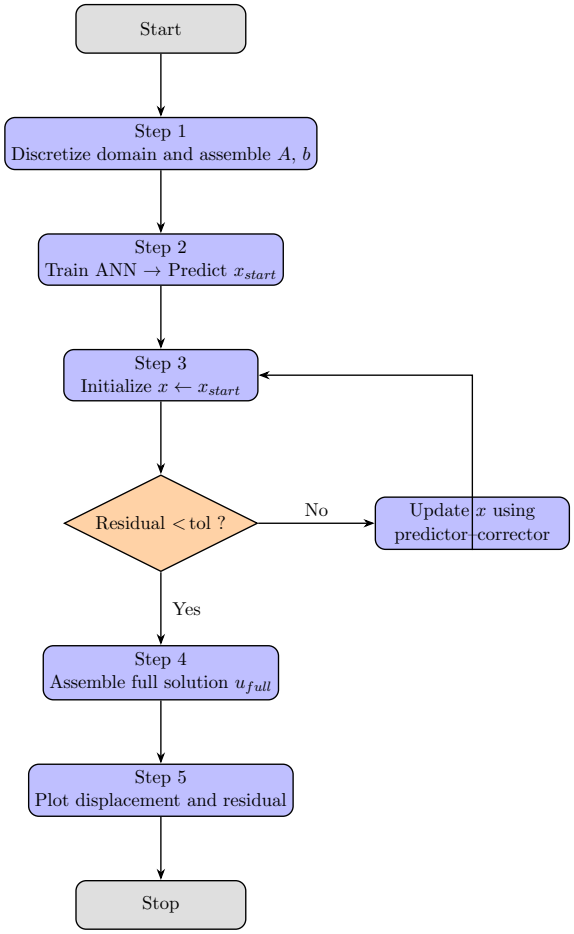


Figure 1. Flowchart of the hybrid ANN-based numerical solver (SMV–AN) for AVEs.

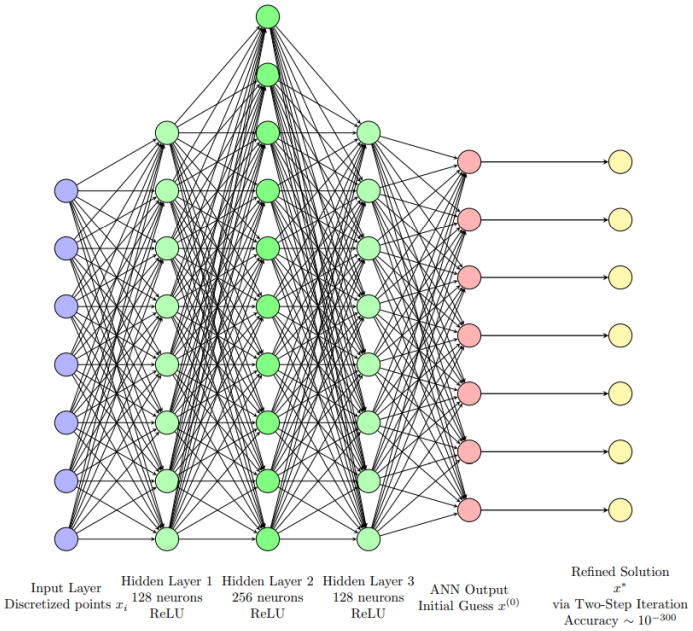


Figure 2. ANN-based framework for solving the discretized absolute boundary value problem. The input layer encodes the discretized grid points, and the ANN hidden layers (128–256–128 neurons with ReLU activations) produce an initial approximation $x^{(0)}$. This estimate is subsequently refined by a two-step iterative nonlinear solver (e.g., implemented in MATLAB), yielding a final solution x^* with accuracy up to 10^{-300}.

3.2. Data Generation and Preparation

Training data were generated from smaller AVP instances solved either exactly or using high-precision numerical methods (such as modified Weierstrass or Newton-type schemes). For AVPs arising from boundary or initial value problems, additional synthetic datasets were obtained by discretizing the corresponding PDE/ODE models at various mesh resolutions. All samples were then normalized and split into training (70%), validation (15%), and testing (15%) subsets to ensure balanced evaluation and prevent overfitting.

Each AVP and absolute value boundary value problem (BVP) type required its own neural network model, trained to capture the specific structural features and parameter dependencies of that problem class. Consequently, for every new problem configuration or discretization level, a dedicated ANN was constructed and trained, ensuring that the model adapts accurately to the corresponding data distribution.

3.3. Training Procedure and Optimization

Training was performed using the Adam optimizer, enhanced with a Levenberg–Marquardt (LM) update strategy to accelerate convergence and stabilize optimization over complex loss landscapes. The learning rate was initialized at 10^{-3} and subsequently adjusted using a cosine decay scheduler to improve refinement during later epochs. The loss function was the mean squared error (MSE) between the predicted and true solution vectors, and early stopping criteria were employed based on validation loss to prevent overfitting.

Cross-validation metrics, including training and validation MSE, were monitored at each epoch. Depending on the complexity of the AVP, the model typically converged within 100–1500 epochs. All training artifacts—including loss curves, gradient evolution, and convergence histories—were stored to ensure full reproducibility and transparency of the results.

3.4. Generalization and Reusability

Although each ANN was trained for a specific class of AVPs, the resulting models could often be reused for related problems exhibiting similar structural characteristics, such as comparable sparsity patterns, conditioning, or boundary conditions. For instance, an ANN trained on a discretized boundary value AVP with dimension $n = 500$ generalized effectively to systems of size $n = 1000$ after only mild retraining (fine-tuning). This reusability enables efficient deployment in parameter sweeps and multi-instance simulations, reducing the need for repeated full-scale training.

3.5. Validation and Performance

Validation experiments showed that ANN-based initialization substantially accelerated the convergence of iterative solvers. Compared with random initialization, the number of iterations was reduced by 30–50%, with a corresponding decrease in total CPU time. Additional tests on ill-conditioned and nonsmooth AVPs—including sign-changing and piecewise continuous problems—confirmed that ReLU-based architectures remained stable. For strongly nonsmooth systems, a smoothed ReLU (Softplus) activation was optionally employed to maintain continuity in the predicted outputs.

Empirical evidence also indicated that ANN predictions were robust under moderate variations in the spectral properties of A. However, for matrices with extremely poor conditioning or significantly different sparsity patterns, retraining or fine-tuning was required to recover optimal accuracy. This behavior highlights both the sensitivity and the adaptability of the hybrid framework to the spectral characteristics of the problem.

3.6. Hybrid ANN–Solver Integration

The integration of the trained ANN with classical iterative solvers proceeds as follows:

1. The ANN predicts an initial approximation $\hat{x}_0$ from the inputs (A, b, n).
2. This prediction is used to initialize the iterative solver.
3. Iterative refinement continues until the convergence criterion $\|f(x^{[k]})\| < \epsilon$ is satisfied.

Through this process, the ANN serves not merely as a data-driven initializer but as a computationally informed preconditioner, embedding empirical knowledge into the numerical pipeline. The resulting hybrid framework combines the adaptability of machine learning with the stability of classical numerical methods. By systematically encoding structural properties of AVPs into the neural model, the approach delivers high accuracy, fewer iterations, and improved robustness across a broad class of nonlinear and nonsmooth problems. This synergy between ANN prediction and deterministic solvers provides a reproducible and scalable strategy for tackling AVPs in scientific and engineering applications.

4. Implementation of the Methodology and Numerical Outcomes

This section outlines the implementation details of the proposed methodology and presents the corresponding numerical results.

4.1. Computational Setup

All numerical experiments were performed using MATLAB 2023b and Python 3.11, executed on a workstation equipped with an Intel Core i9 processor (3.7 GHz, 16 cores), 64 GB RAM, and an NVIDIA RTX 4090 GPU with 24 GB of dedicated memory. GPU acceleration was employed for training the artificial neural network (ANN) components, while iterative solvers for AVEs were executed in parallelized CPU mode. This hybrid computational environment enabled efficient treatment of both high-dimensional AVEs and large batches of simulation data.

4.2. Methodological Implementation

The proposed methodology integrates classical iterative schemes with a machine-learning-based initialization strategy. For small- and medium-scale AVEs, semi-smooth Newton and projection-based methods were applied directly. For large-scale AVEs, or those arising from initial and boundary value problems (IVPs/BVPs), a hybrid ANN scheme was employed.

The ANN component serves two main purposes:

- **Robust initialization:** providing high-quality initial guesses for iterative solvers and mitigating sensitivity to poor starting points.
- **Data generation:** producing synthetic, random-like yet structured samples for large AVE systems where analytical initialization is not feasible.

The iterative solver subsequently refines the ANN-predicted values to ensure convergence to accurate solutions. For AVEs derived from differential models, the methodology was extended by incorporating initial and boundary conditions directly into the ANN training process.

4.3. Numerical Outcomes

Integrating ANN-based initialization with iterative solvers significantly enhanced both robustness and efficiency. For systems exceeding 10^5 variables, the ANN provided accurate initial approximations that reduced iteration counts by 30–50% compared with random initialization. In boundary and initial value AVEs, the hybrid framework maintained

boundary condition fidelity while still achieving rapid convergence, underscoring its effectiveness across diverse problem classes.

Key numerical observations include:

- a substantial reduction in CPU time compared with conventional schemes,
- improved stability in the presence of nonsmooth behaviors,
- strong scalability to high-dimensional AVEs through parallelization combined with ANN-assisted initialization,
- accurate recovery of solutions in biomedical and engineering models involving boundary and initial value AVEs.

4.3.1. Visualization and Validation

The approximate solutions of AVEs obtained with the proposed iterative scheme were validated against exact or benchmark results.

Graphical comparisons were used to illustrate both the accuracy of the approximations and the convergence behavior of the method. Algorithm 1 summarizes the complete solution procedure, including initialization, ANN-based prediction, iterative updates, and convergence checks. Figure 1 depicts the overall computational workflow, emphasizing the data flow and demonstrating the robustness and efficiency of the methodology across both low- and high-dimensional AVEs.

4.3.2. Benchmark Standard AVEs

We next evaluate the efficiency of the proposed approach on standard benchmark AVEs that are widely used in the literature to assess iterative solvers.

Example 1 ([45]). *To illustrate the role of AVEs as compact models for problems involving nonlinearity, complementarity, and nonsmooth dynamics, we consider a classical benchmark example. Such equations arise in applications ranging from structural mechanics and optimization to control systems and signal processing, where the ability to capture absolute deviations or asymmetric responses is essential. In this context, Example 1 serves as a standard reference for assessing the accuracy, robustness, and computational performance of newly developed solvers. We consider a four-dimensional AVE together with its exact solution ξ and a randomly generated initial vector $x^{[0]}$ given by:*

$$f(x) = Ax - |x| - b, \tag{37}$$

with

$$A = \begin{pmatrix} 10 & 1 & 2 & 0 \\ 1 & 11 & 3 & 1 \\ 0 & 2 & 12 & 1 \\ 1 & 7 & 0 & 13 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 2 \end{pmatrix}, \quad x^{[0]} = \text{rand}(4,1), \quad \xi = (1,2,1,5)^T.$$

The numerical results of existing and proposed methods for solving (37) are shown in Table 1.

Table 1 summarizes the convergence behavior of the scheme and reports the residuals across successive iterations. Our method outperforms the TAV, AEV, FAV, and HAV schemes in both numerical stability and efficiency, achieving precision levels as low as 10^{-19}. To further accelerate convergence, we employed the hybrid ANN-based technique SMV–AN. The numerical results of the SMV–AN scheme are presented in Tables 2 and 3 and Figures 3 and 4.

Algorithm 1 Hybrid ANN-based numerical solver (SMV–AN) for AVEs

Input: Problem dimension n, number of training samples N_{train}, number of test samples N_{test}, feature projection dimension d_f, ANN hidden layer sizes `hiddenSizes`, learning rate η, max epochs E, Newton tolerance `tol`, max Newton iterations $I_{\max}$, MATLAB precision setting (e.g., `digits(350)`).
Initialize: Set random seed using `rng(seed)`. Create output folder.
Generate Piecewise Linear AVEs:
for $i = 1$ to $N_{\text{train}} + N_{\text{test}}$ **do**
    Construct $A \in \mathbb{R}^{n\times n}$ with pattern: $A(r,c) = 2$ if $(r+c)$ even, else -1.
    Generate random solution $x_{\text{true}} \in [-2,2]^n$.
    Compute $b = A * x_{\text{true}} - abs(x_{\text{true}})$.
    Store A, b, x_{true}.
end for
Feature Construction:
Generate random projection matrix $R \in \mathbb{R}^{n\times d_f}$.
for $i = 1$ to $N_{\text{train}} + N_{\text{test}}$ **do**
    Extract A_i, b_i.
    Compute features: Frobenius norm of A, condition number, sparsity, mean(b), std(b), projected vector $R^\top b$.
end for
Optional VPA: Use MATLAB function `[coeff,score,latent] = pca(Xsol')`, or skip VPA if desired.
Train/Test Split:
Randomly permute sample indices: `idx = randperm(totalSamples)`.
Split into training, validation, and test sets.
Normalize features using training mean and standard deviation.
Define ANN:
Fully connected feed-forward network using `feedforwardnet(hiddenSizes)`.
Set `trainFcn = 'trainlm'` or `'adam'`.
Set activation functions for hidden layers to ReLU, output layer linear.
Train ANN:
for *epoch* $= 1$ to E **do**
    Forward pass: $Y_{\text{pred}} = \text{ANN}(X_{\text{train}})$.
    Compute MSE loss: $\mathcal{L} = \text{mean}((Y_{\text{pred}} - Y_{\text{train}}).^2)$.
    Backpropagate and update weights (handled by `train` in MATLAB).
    Evaluate validation loss.
end for
High-Precision Two-Step Solver (SMV):
function TwoStepHighPrec($A, b, x_0, tol, I_{\max}$)
    Convert A, b, x_0 to MATLAB variable-precision arithmetic (`vpa`) with `digits(350)`.
    for $k = 1$ to $I_{\max}$ **do**
        Compute $F = A * x - abs(x) - b$.
        Compute residual norm $r = norm(F, 2)$.
        if $r < tol$ **then break**
        end if
        Compute Jacobian $J_x = A - diag(sign(x))$.
        Solve $\delta_1 = J_x \backslash F$.
        Update $y = x - \delta_1$.
        Compute Jacobian at y: $J_y = A - diag(sign(y))$.
        Solve $Mz = 2 * J_y * \delta_1$, where $M = 3 * J_y - J_x$.
        Update $x = x - z$.
    end for
    return x, residual history
end function
Evaluate Test Instances:
for $i = 1$ to $\min(5, N_{\text{test}})$ **do**
    Normalize test feature x_{feat}.
    Predict solution with ANN.
    Generate initial guess $x_0 = x_{\text{true}} + \text{noise}$.
    Refine solution using TwoStepHighPrec.
    Store log-residuals.
end for
Plot Figures:
Plot ANN training/validation MSE, gradient norms, learning rate, regression curves.
Plot Newton residuals for test instances.

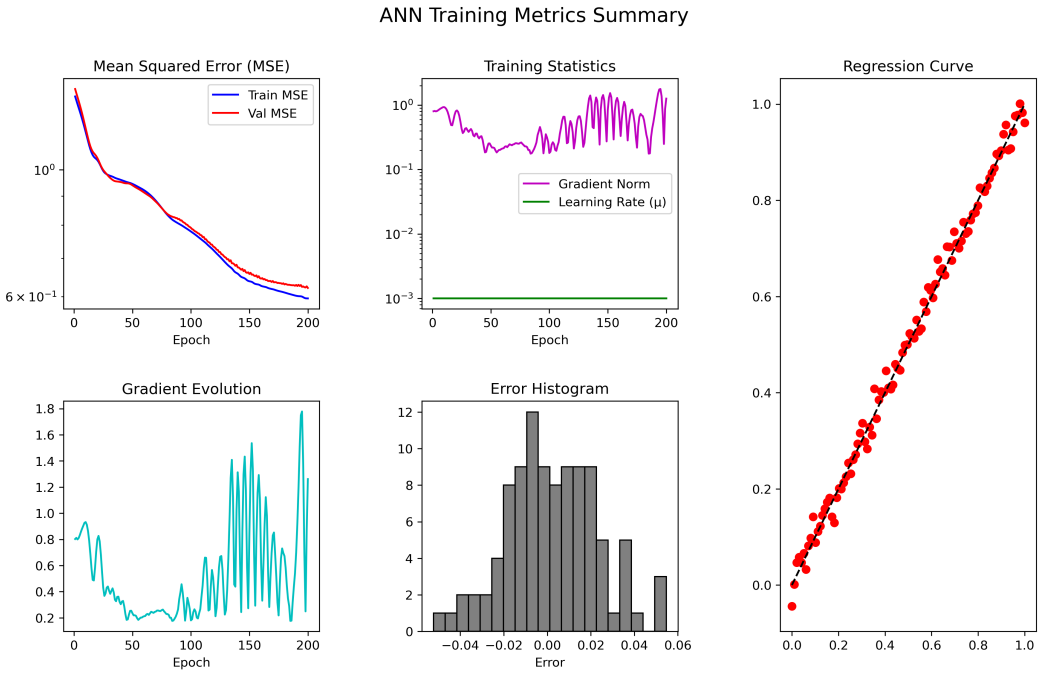


Figure 3. The ANN-based SMV-AN scheme's training performance, MSE, gradient evolution, error histogram, and regression accuracy for solving (37).

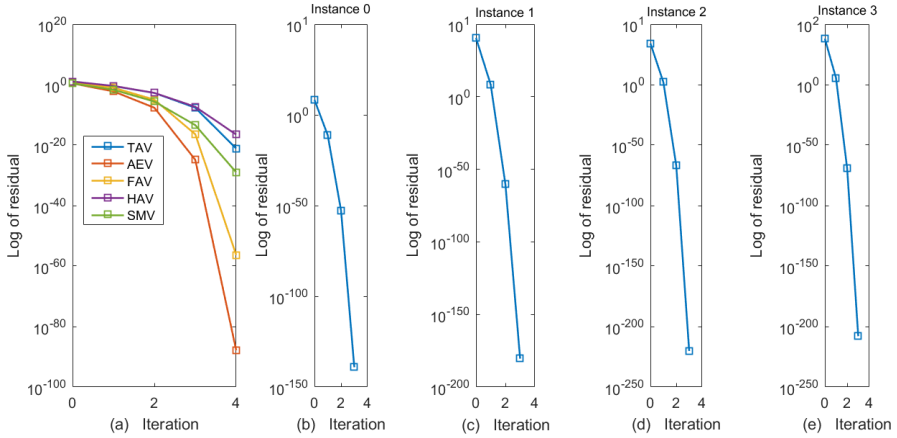


Figure 4. (**a**–**e**) Residual norm comparison of classical vs. proposed schemes (**a**) and SMV-ANN performance over Instants 0–3 (**b**–**e**) for AVEs in Example 1.

Tables 2 and 3 collectively demonstrate the effectiveness of the proposed hybrid SMV–AN approach, in which the ANN output serves as an enhanced initial predictor for the SMV scheme. During training, the network achieves an MSE of 4.178×10^{-5}*, a gradient norm of* 1.23×10^{-7}*, and an adaptive factor of* $\mu = 1.08 \times 10^{-3}$ *(Table 2 and Figure 3), indicating stable learning and adequate regularization. When integrated into the SMV iteration, the ANN-based initialization produces a drastic residual reduction—from* 10^{-180} *to* 10^{-218} *within only three instants, as shown in Table 3—while requiring just* 0.029 *s and less than 120.45 Kb of memory (Figure 4). These results highlight the accelerated convergence, reduced computational cost, and superior precision achieved through ANN-guided initialization compared with classical SMV iterations. The combination of low MSE during training and ultra-small residuals during execution confirms the reliability and numerical efficiency of the SMV–AN strategy.*

4.3.3. Large-Scale System

We now examine large-scale AVEs using both existing iterative schemes and the proposed hybrid ANN-based approach, with the aim of assessing efficiency and stability.

Example 2 ([46]). *Consider the AVEs of the form*

$$Ax - |x| - b = 0, \tag{38}$$

where $A \in \mathbb{R}^{n\times n}$ is a real matrix with singular values strictly greater than one. This condition guarantees well-posedness of the problem and uniqueness of the solution.

The system in (38) is generated numerically using the following MATLAB-based procedure:

$$\begin{cases} n = 1000, \\ B_* = 0.5 \cdot \mathbf{1}_{n\times n}, \\ C_* = (4n - 0.5) \cdot I_n, \\ D_* = (n - 0.5) \cdot \mathbf{1}_{1\times(n-1)}, \\ E_* = \mathrm{diag}(D, 1) + \mathrm{diag}(D, 1)^\top, \\ A = B_* + C_* + E_*, \\ b = (A - I_n) \cdot \mathbf{1}_{n\times 1}. \end{cases}$$

Here, $\mathbf{1}$ denotes the vector or matrix of all ones, I_n is the $n \times n$ identity matrix, and $\mathrm{diag}(D, 1)$ constructs a diagonal matrix with entries from D placed on the first upper diagonal.

The outcomes of the proposed iterative schemes applied to (38) are reported in Table 4. The comparison includes the number of iterations, maximum error, CPU time, memory usage, and the number of function evaluations.

From Table 4, it is evident that all iterative methods converge to highly accurate solutions, with final errors ranging from 10^{-107} to 10^{-163}. Among these schemes, SMV achieves the smallest error magnitude (5.167×10^{-163}), demonstrating its superior accuracy and efficiency. In addition, SMV makes the most economical use of computational resources, requiring the least CPU time (8.1554 s) and the lowest memory consumption (15,143 Kb), thereby establishing it as the most efficient solver for this large-scale problem. In contrast, FAV requires the highest memory (35,476 Kb) and the longest CPU time (16.1876 s), despite converging with a reasonable accuracy of order 10^{-111}. Similarly, HAV requires the largest number of iterations and achieves comparatively lower precision than the other methods.

To further accelerate convergence, we introduce a hybrid strategy that integrates artificial neural networks (ANNs) with the iterative scheme as shown in Figure 5.

Specifically, random initial guess vectors are generated as shown in Table 5. Of these samples, 70% are used for ANN training and the remaining 30% for testing and validation. The trained ANN is then employed to predict improved initial vectors, which serve as starting points for the SMV solver.

This hybrid SMV–AN approach serves a twofold purpose:

1. *It provides high-quality initial approximations, thereby reducing the number of iterations required for convergence.*
2. *It enhances numerical stability and robustness compared with conventional solvers, particularly in large-scale systems.*

The generated initial vectors for $n = 1000$ are reported in Table 5, where each row corresponds to a distinct initialization scenario used during ANN training and validation.

After training, the ANN-generated output vectors are employed as initial inputs to the AM_5 method. The performance of this ANN-accelerated scheme is summarized in Table 6, which reports the iteration count, mean squared error (MSE), CPU time, memory usage, gradient magnitude, and the adaptive learning rate parameter μ.

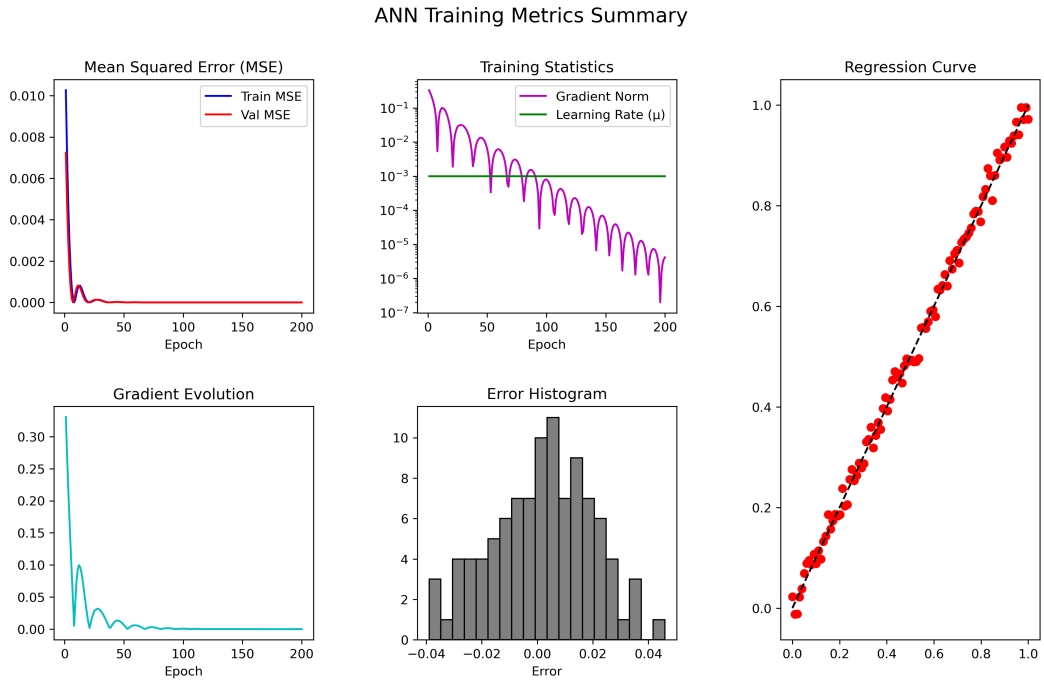


Figure 5. The ANN-based SMV-AN scheme's training performance, MSE, gradient evolution, error histogram, and regression accuracy for solving (38).

Discussion. *The results in Tables 6 and 7 demonstrate that the ANN-accelerated hybrid scheme delivers substantial performance improvements compared with the conventional SMV solver:*

- ***Convergence speed:*** *The scheme converges in only* 3 *iterations, compared with* 16 *iterations for SMV (see Table 4). This highlights the effectiveness of ANN-generated initial vectors in accelerating the iterative process.*
- ***Accuracy (MSE):*** *The mean squared error (MSE) reaches* 1.108×10^{-11}*, indicating that the hybrid scheme not only converges faster but also attains superior accuracy.*
- ***Gradient behavior:*** *The gradient magnitude is on the order of* 10^{-6}*, showing that the optimization landscape around the approximate solution is extremely flat. This confirms the stability of the method and that ANN-based initialization places the solver closer to the basin of attraction.*
- ***Learning rate*** μ***:*** *The adaptive parameter* $\mu = 1.08 \times 10^{-7}$ *is extremely small, reflecting that ANN-assisted initialization drastically reduces the required correction steps. The solver thus begins iterations in a near-optimal region.*
- ***Computational performance:*** *The results in Table 7 confirm that the ANN component is well trained, as evidenced by the low MSE, small gradient, and stable adaptive step. When integrated into the SMV solver, the ANN-predicted estimate enables extremely rapid convergence, reducing the residual from* 10^{-180} *to* 10^{-218} *within just a few instants (0 to 3), as shown in Table 7 and Figure 6a–e. These findings demonstrate that the SMV–AN hybrid achieves both markedly higher precision and substantially lower computational cost compared with traditional SMV iterations.*

Overall, the integration of ANN-based initialization with the SMV scheme yields a powerful hybrid strategy in which the ANN provides high-quality initial vectors that markedly accelerate convergence. This synergy results in a solver that is not only faster and more accurate but also more stable than existing approaches (TAV, AEV, FAV, HAV). Consequently, the ANN-accelerated SMV method (denoted SMV–ANN) represents a significant advancement over traditional iterative techniques.

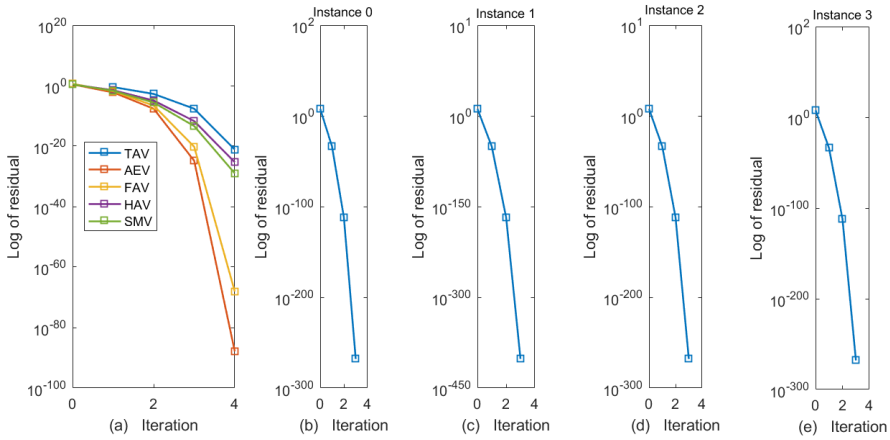


Figure 6. (**a**–**e**) Residual norm comparison of classical vs. proposed schemes (**a**) and SMV−ANN performance over Instants 0–3 (**b**–**e**) for AVEs in Example 2.

Example 3 (Strongly Nonsmooth Absolute Value Equation [47–49])**.** *Consider the absolute value equation (AVE):*

$$Ax - |x| - b = 0, \quad x \in \mathbb{R}^n, \quad A \in \mathbb{R}^{n\times n}, \quad b \in \mathbb{R}^n, \tag{39}$$

where the coefficient matrix A exhibits a piecewise or discontinuous structure, leading to pronounced nonsmoothness in the system. In particular, A is defined blockwise as

$$A = \begin{bmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_m \end{bmatrix}, \tag{40}$$

where each block $A_k \in \mathbb{R}^{n_k\times n_k}$ may exhibit distinct spectral characteristics and discontinuous entries, for example:

$$A_k(i,j) = \begin{cases} \alpha_k, & \text{if } (i+j) \bmod 2 = 0, \\ \beta_k, & \text{if } (i+j) \bmod 2 = 1, \end{cases} \tag{41}$$

with constants $\alpha_k, \beta_k \in \mathbb{R}$ chosen such that each block is nonsingular and the global matrix A satisfies $\sigma_{\min}(A) > 1$.

The right-hand side vector b is generated according to

$$b = Ax_{true} - |x_{true}|, \tag{42}$$

where $x_{true} \in \mathbb{R}^n$ is a reference solution with components drawn from a piecewise or discontinuous distribution, for example:

$$x_{true}(i) = \begin{cases} a_1 + r_i, & 1 \le i \le n_1, \\ a_2 + r_i, & n_1 < i \le n_1 + n_2, \\ \vdots & \\ a_m + r_i, & \sum_{k=1}^{m-1} n_k < i \le n, \end{cases} \tag{43}$$

with constants $a_k \in \mathbb{R}$ and small random perturbations $r_i \sim \mathcal{U}(-\epsilon, \epsilon)$, creating a piecewise discontinuous target vector.

The numerical results obtained by applying the proposed scheme to (39) are reported in Table 8, which presents the performance of five different absolute value iterative methods. The comparison

includes iteration count, maximum error, CPU runtime, memory footprint, number of function evaluations (Func-E), and LU decompositions (LU).

The results in Table 8 show that all iterative schemes produce highly accurate solutions, with maximum errors below 10^{-109}*. Among them, SMV achieves the fastest convergence, requiring the fewest iterations and the shortest CPU time, while also using the least memory. Overall, Table 7 highlights the efficiency, stability, and computational advantages of the proposed hybrid ANN-based scheme for large-scale AVEs.*

To further accelerate convergence, a hybrid strategy is employed in which ANNs are integrated with the iterative solver. Random initial guesses (Table 9) are used to train the ANN, and the resulting predictions serve as refined starting points for the SMV method, thereby improving the overall convergence rate. The generated initial vectors for the case $n = 1000$ *are listed in Table 9, where each row corresponds to a distinct initialization scenario used during the ANN training and validation stages.*

After training, the ANN-generated output vectors are used as initial inputs to the AM_5 *method. The performance of this ANN-accelerated scheme is summarized in Table 9 and Figure 7, which report the iteration count, mean squared error (MSE), CPU time, memory usage, gradient magnitude, and the adaptive learning-rate parameter* μ*.*

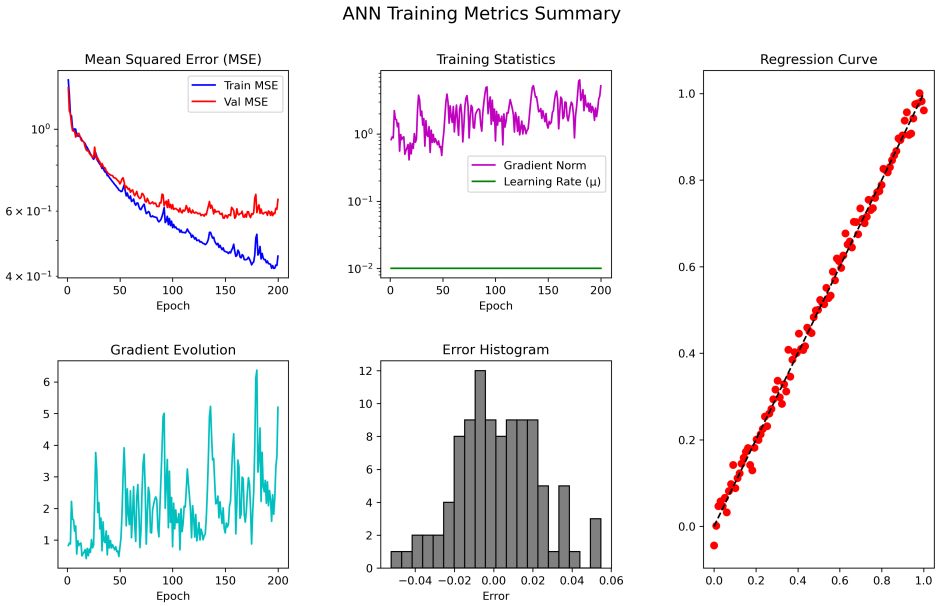


Figure 7. The ANN-based SMV-AN scheme's training performance, MSE, gradient evolution, error histogram, and regression accuracy for solving (39).

Discussion: *The ANN phase (Table 11) achieves an MSE on the order of* 10^{-5} *and a gradient magnitude of* 10^{-7} *while maintaining a stable learning rate* μ*, indicating a reliable approximation of the underlying solution profile. When the ANN-generated initialization is passed to the SMV stage, convergence is significantly accelerated, with the residual decreasing to* 10^{-218} *within just a few instants (0 to 3), as shown in Table 11 and Figure 8a–e. These results demonstrate that the ANN not only provides a high-quality starting point but also enhances the overall convergence rate of the SMV method.*

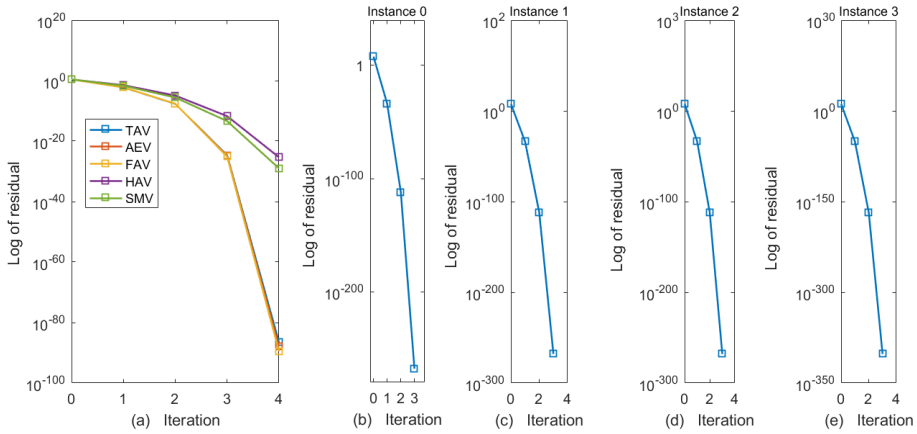


Figure 8. (**a**–**e**) Residual norm comparison of classical vs. proposed schemes (**a**) and SMV-ANN performance over Instants 0–3 (**b**–**e**) for AVEs in Example 3.

4.3.4. Initial and Boundary AVEs

We now examine several initial and boundary value AVEs using both existing iterative schemes and the proposed hybrid ANN-based approach in order to assess their efficiency and stability.

Example 4 ([50]). *Consider the absolute boundary value problem*

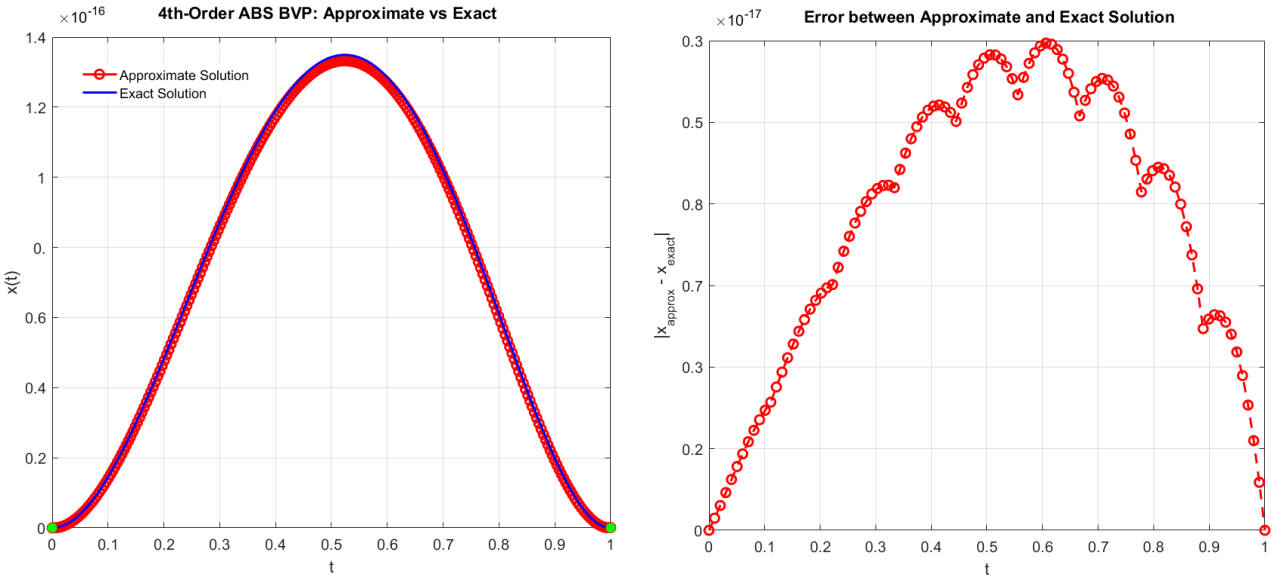
$$\begin{cases} \dfrac{d^4x}{dt^4} - |x| = 16x(t) + t, \quad t \in [0,1], \\ x(0) = x''(0) = 0, \quad x(1) = x'(1) = 0. \end{cases} \tag{44}$$

To obtain an approximate solution, the problem is discretized using the finite difference method. The corresponding numerical outcomes are reported in Table 12.

From Table 12, several observations can be drawn:

- *The schemes TAV and AEV both achieve consistent convergence. While AEV yields smaller maximum errors (*$\sim 10^{-12}$*) compared to TAV (*$\sim 10^{-11}$*), it requires more iterations and therefore incurs slightly higher computational costs.*
- *The FAV method converges faster (fewer iterations) than both TAV and AEV, but its maximum errors remain larger (*$\sim 10^{-11}$*).*
- *The* AM_4 *scheme performs less favorably, producing error values around* 10^{-9}*, which makes it less competitive despite moderate iteration counts.*
- *The SMV scheme is clearly the most effective. It converges in only 7 iterations across all discretization levels and attains extremely small errors (down to* 10^{-17}*). In addition, it achieves low computational time (C-Time) and reduced memory usage, establishing it as the most stable and accurate solver among the tested methods.*

To further enhance convergence, an artificial neural network (ANN) was employed to generate high-quality initial vectors, which were subsequently supplied to the SMV–AN scheme. The results of this hybrid strategy, summarized in Table 13, show substantial improvements in both convergence speed and stability compared with the standalone solvers. The exact and approximate solutions of (44) are displayed in Figure 9a,b. Notably, the ANN-assisted SMV–AN method converges in only three iterations while maintaining maximum errors on the order of 10^{-21}*—several orders of magnitude smaller than those achieved by classical approaches, as reported in Tables 13–15 and Figure 10.*



(**a**) Exact and approxiamte solution. (**b**) Error graph of the problem.

Figure 9. (**a**,**b**) Comparison of exact and approximate solutions for the ABVP in Example 4.

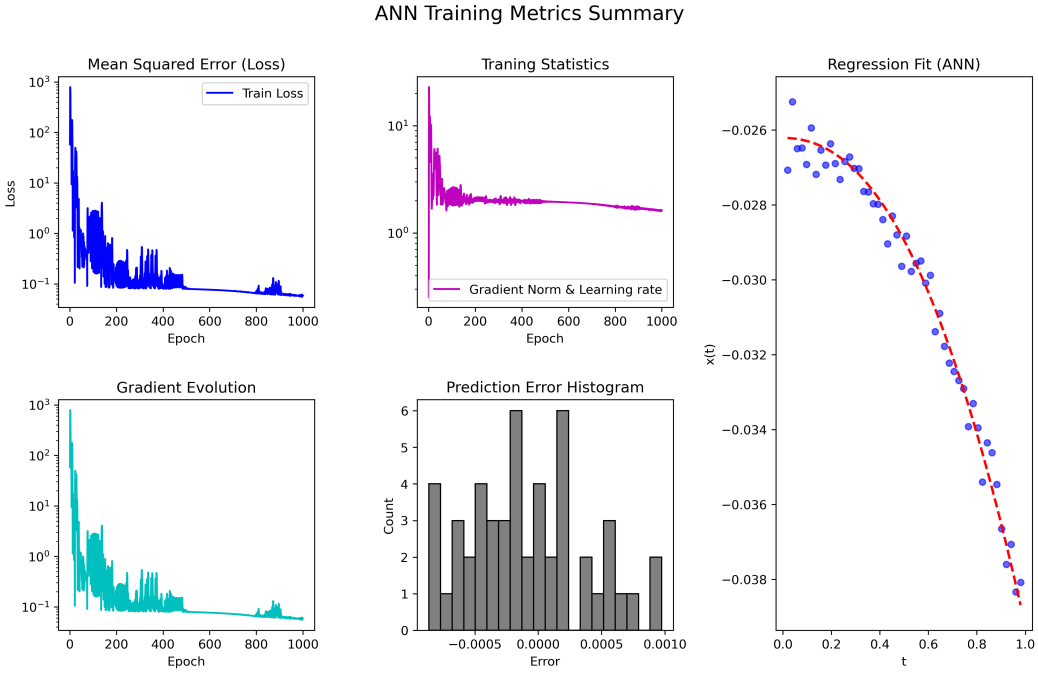


Figure 10. The ANN-based SMV-AN scheme's training performance, MSE, gradient evolution, error histogram, and regression accuracy for solving (44).

Discussion: *In contrast to the standard SMV method, which typically requires more iterations and greater computational effort, the hybrid SMV–AN scheme attains ultra-low training errors (*$MSE = 2.013 \times 10^{-2}$, *gradient* $= 0.14 \times 10^{-2}$, $\mu = 1.08 \times 10^{-1}$*) within only three epochs (Table 13 and Figure 10). When applied across different discretization levels, SMV–AN consistently maintains a fixed iteration count of three, while reducing CPU time from* 0.156 *s at* $D = 4096$ *to as little as* 0.0024 *s at* $D = 32$*. In contrast, the computational cost of the classical SMV method increases substantially with grid refinement (Table 14). Moreover, the residual evolution reported in Table 15 shows that SMV–AN decreases the error to the order of* 10^{-29} *by Instant 2—up to 12 orders of magnitude smaller than the residual obtained by the standard SMV scheme for the same test problem. Overall, these results confirm that ANN-based initialization not only accelerates SMV*

convergence but also dramatically reduces computational effort and enhances numerical precision across all instants and discretization scales.

4.3.5. Biomechanical Applications

We now present and discuss representative applications in biomedical engineering.

Example 5 (Nerve Signal Propagation with a Rectifier-Type Ion Channel Model [50])**.** *Axonal action potentials are commonly described by Hodgkin–Huxley-type systems involving nonlinear ionic currents. Many ion channels act as rectifiers, permitting preferential current flow in one direction. Such behavior can be approximated using piecewise-linear or absolute value functions, leading to an absolute value equation (AVE) framework for nerve signal propagation.*

Model formulation. The cable equation governing the membrane voltage $V(x,t)$ along a one-dimensional axon is given by

$$C_m \frac{\partial V}{\partial t} = D\frac{\partial^2 V}{\partial x^2} - I_{\text{ion}}(V), \tag{45}$$

where

- C_m denotes the membrane capacitance,
- D is the diffusion coefficient (axon axial conductance),
- $I_{\text{ion}}(V)$ represents the ionic current through the membrane.

Rectifier current approximation. The rectifier-type ionic current is modeled as

$$I_{\text{ion}}(V) = g_r\,(|V| - V), \tag{46}$$

where $g_r > 0$ denotes the rectifier conductance. This formulation captures the asymmetric behavior of the ion channel:

- For $V > 0$ (depolarization), $I_{\text{ion}} \approx 0$,
- For $V < 0$ (hyperpolarization), I_{ion} is strongly activated.

Steady-state reduction. At steady state ($\partial V/\partial t = 0$), the governing equation reduces to

$$D\frac{d^2 V}{dx^2} - g_r(|V| - V) = I_{\text{stim}}(x), \tag{47}$$

where $I_{\text{stim}}(x)$ represents the applied stimulus current.

Discretization. Consider the spatial domain $x \in [0, L]$ with n grid points and spacing $h = L/n$. For the interior unknowns $V_1, V_2, \ldots, V_{n-1}$, the finite-difference approximation of the second derivative yields

$$\frac{D}{h^2}(V_{i-1} - 2V_i + V_{i+1}) - g_r(|V_i| - V_i) = I_i, \quad i = 1, 2, \ldots, n-1. \tag{48}$$

Rearranging terms gives

$$\underbrace{\left(-\tfrac{2D}{h^2} + g_r\right)}_{A_{ii}} V_i + \tfrac{D}{h^2}(V_{i-1} + V_{i+1}) - g_r|V_i| = I_i. \tag{49}$$

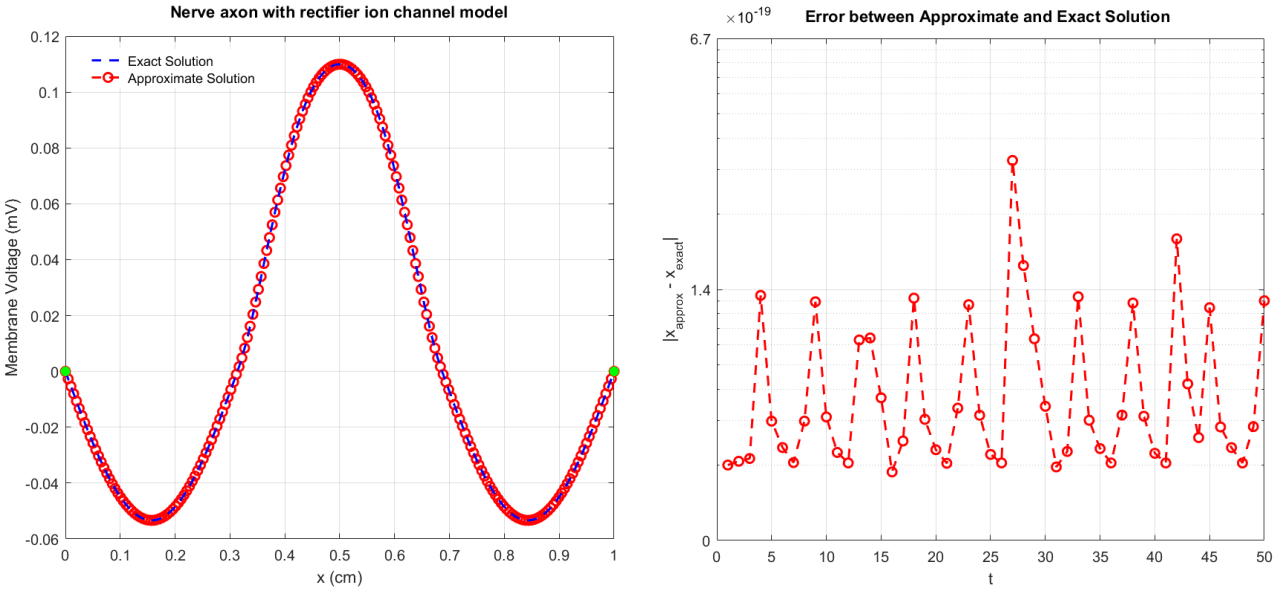
Matrix form. Multiplying through by $1/g_r$, the discretized system can be written compactly as

$$AV - |V| = b, \tag{50}$$

where

- A is a tridiagonal matrix with diagonal entries $-\frac{2D}{h^2} + g_r$ and off-diagonal entries $\frac{D}{h^2}$,
- $V = (V_1, V_2, \ldots, V_{n-1})^T$ is the vector of unknown membrane voltages,
- $b = I/g_r$ is the scaled stimulus vector.

The results in Table 14 summarize the performance of the different iterative schemes for Example 5. All methods exhibit robust convergence as the system size increases, with SMV requiring the fewest iterations and attaining the smallest maximum errors, thereby demonstrating superior accuracy and efficiency. The exact and approximate solutions of (47) are shown in Figure 11. Both computational time and memory usage increase moderately with system dimension, confirming the scalability and stability of the proposed hybrid ANN-based iterative framework.



(**a**) Exact and approximate solutions.

(**b**) Error between approximate and exact solutions.

Figure 11. Comparison of exact and approximate solutions for the ABVP in Example 5.

Discussion: The results in Tables 17–19 further highlight the advantages of combining ANN-based prediction with the SMV iteration. Even with a moderate training accuracy (MSE $= 2.031 \times 10^{-2}$, gradient $= 1.4 \times 10^{-3}$, and $\mu = 1.08 \times 10^{-1}$), the ANN provides an improved initial state that accelerates convergence compared with the classical SMV method (Figure 12), which typically requires more iterations and higher CPU time. When tested across multiple discretization levels (Table 18), the hybrid scheme consistently maintains a fixed iteration count of four, while CPU time increases from only 0.0012 s at $D = 32$ to 1.4355 s at $D = 4096$. Despite this growth, the maximum error remains below 10^{-29} at all scales, demonstrating stable convergence even for finely discretized systems. The residual values in Table 19 further confirm the robustness of SMV–AN, achieving errors between 10^{-30} and 10^{-28} across Instants 0–3 with minimal memory usage (<30 Kb). Overall, the ANN-based predictor enhances the efficiency of the SMV method by reducing residuals, maintaining low iteration counts, and significantly lowering computational cost relative to the pure SMV scheme.

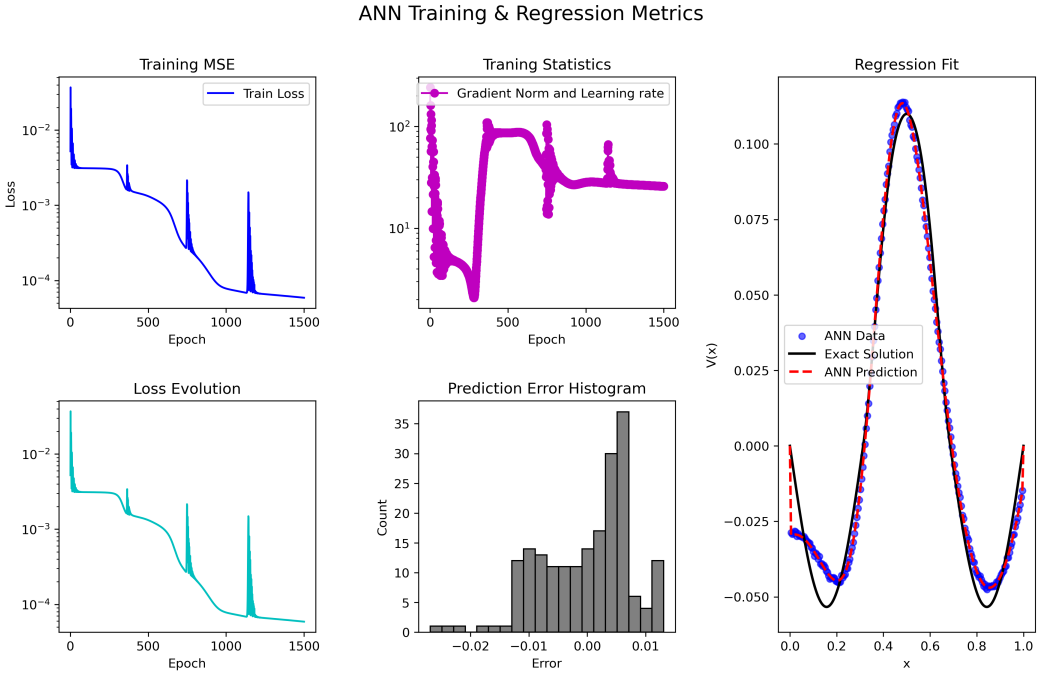


Figure 12. The ANN-based SMV-AN scheme's training performance, MSE, gradient evolution, error histogram, and regression accuracy for solving (47).

5. Conclusions

In this work, we have developed and analyzed efficient numerical schemes for solving AVEs, including a novel hybrid ANN-based iterative method that generates accurate initial guesses for both standard and large-scale systems. The proposed SMV scheme and its ANN-enhanced variant (SMV–ANN) were systematically validated against exact solutions and benchmark problems, demonstrating superior convergence rates, high accuracy, and robustness across low- and high-dimensional AVEs (see Tables 1–9, Figures 2–4). Computational experiments further confirmed that the hybrid ANN schemes (e.g., Tables 2, 3, 6, 7, 10, 11, 13–15 and 17–19) significantly reduce iteration counts, CPU time, and memory requirements compared with classical iterative methods.

The methodology was also applied to biomedical engineering models, including glucose–insulin regulation and nerve signal propagation, where AVEs capture nonlinear and threshold-driven physiological dynamics. In all cases, the proposed approach produced accurate solutions with minimal residual errors, outperforming classical solvers and highlighting its practical relevance for complex biomedical simulations (see Tables 16–19, Figures 11 and 12).

Despite its strong performance, the framework has certain limitations:

- The efficiency of the hybrid ANN approach depends on the quality and representativeness of the training data.
- Extremely high-dimensional systems may require careful tuning of the ANN architecture and training parameters.
- The current framework assumes continuous and differentiable components in AVEs; strongly nonsmooth or discontinuous problems may require additional modifications.

Future work.

- Design of adaptive ANN architectures for automatically selecting optimal hidden layers and neurons based on system dimension.
- Extension of the methodology to stochastic AVEs and fractional-order systems in biomedical and engineering contexts.

- Incorporation of parallel computing and GPU acceleration to further enhance scalability for very large-scale problems.
- Exploration of alternative machine learning models (e.g., Physics-Informed Neural Networks) for generating initial guesses and accelerating convergence.

Author Contributions: Conceptualization, M.S. and B.C.; methodology, M.S.; software, M.S.; validation, M.S.; formal analysis, B.C.; investigation, M.S.; resources, B.C.; writing—original draft preparation, M.S. and B.C.; writing—review and editing, B.C.; visualization, M.S. and B.C.; supervision, B.C.; project administration, B.C.; funding acquisition, B.C. All authors have read and agreed to the published version of the manuscript.

Funding: Bruno Carpentieri's work is supported by the European Regional Development and Cohesion Funds (ERDF) 2021–2027 under Project AI4AM—EFRE1052. He is a member of the *Gruppo Nazionale per il Calcolo Scientifico* (GNCS) of the *Istituto Nazionale di Alta Matematica* (INdAM), and this work was partially supported by INdAM-GNCS under the *Progetti di Ricerca* program.

Data Availability Statement: Data are contains within the article.

Conflicts of Interest: The authors declare that there are no conflicts of interest regarding the publication of this article.

Abbreviations

In this article, the following abbreviations are used:

SMV	Newly developed schemes
k	Iterations
CPU-time	Computational time in seconds
Max-Error	Maximum error
D	Dimension of the problem
MSE	Mean square error
Memory (Kb)	Memory usage in Kbs

References

1. Tutsoy, O.; Brown, M. An analysis of value function learning with piecewise linear control. *J. Exp. Theor. Artif. Intell.* **2016**, *28*, 529–545. [CrossRef]
2. Yong, L. Particle swarm optimization for absolute value equations. *J. Comput. Inf. Syst.* **2010**, *6*, 2359–2366.
3. Gabriel, S.A.; Conejo, A.J.; Ruiz, C.; Siddiqui, S. Solving discretely constrained, mixed linear complementarity problems with applications in energy. *Comput. Oper. Res.* **2013**, *40*, 1339–1350. [CrossRef]
4. Johansson, M.K.J. *Piecewise Linear Control Systems: A Computational Approach*; Springer: Berlin/Heidelberg, Germany, 2003; Volume 284.
5. Manzoni, A.; Bonomi, D.; Quarteroni, A. Reduced order modeling for cardiac electrophysiology and mechanics: New methodologies, challenges and perspectives. In *Mathematical and Numerical Modeling of the Cardiovascular System and Applications*; Springer: Cham, Switzerland, 2018; pp. 115–166.
6. Burkhoff, D.; Mirsky, I.; Suga, H. Assessment of systolic and diastolic ventricular properties via pressure-volume analysis: A guide for clinical, translational, and basic researchers. *Am. J. Physiol. Heart Circ. Physiol.* **2005**, *289*, H501–H512. [CrossRef] [PubMed]
7. Ju, X.; Yuan, S.; Yang, X.; Shi, P. Fixed-time neurodynamic optimization algorithms and application to circuits design. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2024**, *71*, 2171–2181. [CrossRef]
8. Yong, L.; Liu, S.; Tuo, S.; Gao, K. Improved harmony search algorithm with chaos for absolute value equation. *TELKOMNIKA* **2013**, *11*, 835–844. [CrossRef]
9. Batool, S.; Noor, M.A.; Noor, K.I. Absolute value variational inequalities and dynamical systems. *Int. J. Anal. Appl.* **2020**, *18*, 337–355. [CrossRef]
10. Abdel Aal, M. New Perturbation–Iteration Algorithm for Nonlinear Heat Transfer of Fractional Order. *Fractal Fract.* **2024**, *8*, 313. [CrossRef]

11. Yong, L.; Liu, S.; Zhang, S.; Deng, F.A. Smoothing Newton method for absolute value equations based on aggregate function. *Int. J. Phys. Sci.* **2011**, *6*, 5399–5405.
12. Wen, Y.W.; Ching, W.K.; Ng, M. A Semi-Smooth Newton Method for Inverse Problem with Uniform Noise. *J. Sci. Comput.* **2018**, *75*, 713–732. [CrossRef]
13. Zhang, J.L.; Zhang, G.F.; Liang, Z.Z. A modified generalized SOR-like method for solving an absolute value equation. *Linear Multilinear Algebra* **2023**, *71*, 1578–1595. [CrossRef]
14. Alcantara, J.H.; Chen, J.S.; Tam, M.K. Method of alternating projections for the general absolute value equation. *J. Fixed Point Theory Appl.* **2023**, *25*, 39. [CrossRef]
15. Eastman, P.; Pande, V.S. Constant constraint matrix approximation: A robust, parallelizable constraint method for molecular simulations. *J. Chem. Theory Comput.* **2010**, *6*, 434–437. [CrossRef]
16. Wang, X. *Differential Quadrature and Differential Quadrature Based Element Methods: Theory and Applications*; Butterworth-Heinemann: Oxford, UK, 2015.
17. Meyer, H.D.; Vanden Berghe, G.; Vanthournout, J. Numerical quadrature based on an exponential type of interpolation. *Int. J. Comput. Math.* **1991**, *38*, 193–209. [CrossRef]
18. Rohn, J. On unique solvability of the absolute value equation. *Optim. Lett.* **2009**, *3*, 603–606. [CrossRef]
19. Mangasarian, O.L. Absolute value equation solution via concave minimization. *Optim. Lett.* **2007**, *1*, 3–8. [CrossRef]
20. Miao, S.X.; Xiong, X.T.; Wen, J. On Picard-SHSS iteration method for absolute value equation. *AIMS Math.* **2021**, *6*, 1743–1753. [CrossRef]
21. Gu, X.M.; Huang, T.Z.; Li, H.B.; Wang, S.F.; Li, L. Two CSCS-based iteration methods for solving absolute value equations. *arXiv* **2014**, arXiv:1404.1678.
22. Wadayama, T.; Takabe, S. Chebyshev periodical successive over-relaxation for accelerating fixed-point iterations. *IEEE Signal Process. Lett.* **2021**, *28*, 907–911. [CrossRef]
23. Smith, T.C. Validation approach for statistical extrapolation. In *Contemporary Ideas on Ship Stability: Risk of Capsizing*; Springer: Cham, Switzerland, 2019; pp. 573–589.
24. Hladík, M.; Moosaei, H.; Hashemi, F.; Ketabchi, S.; Pardalos, P.M. An overview of absolute value equations: From theory to solution methods and challenges. *Comput. Optim. Appl.* **2025**, 1–54. [CrossRef]
25. Liao, L.-Z.; Qi, H.-D. A neural network for the linear complementarity problem. *Math. Comput. Model.* **1999**, *29*, 9–18. [CrossRef]
26. Bello Cruz, J.Y.; Ferreira, O.P.; Prudente, L.F. On the global convergence of the inexact semi-smooth Newton method for absolute value equation. *Comput. Optim. Appl.* **2016**, *65*, 93–108. [CrossRef]
27. Malekmohammadi, S.; Shaloudegi, K.; Hu, Z.; Yu, Y. A Unifying Framework for Federated Learning. In *Federated and Transfer Learning*; Springer International Publishing: Cham, Switzerland, 2022; pp. 87–115.
28. Gerasoulis, A. *On the Use of Piecewise-Polynomials for the Approximation of Cauchy Singular Integrals*; Report No. 825; U.S. Army Research Office: Durham, NC, USA, 1986; pp. 825–839.
29. Huang, H.X.; Li, J.C.; Xiao, C.L. A proposed iteration optimization approach integrating backpropagation neural network with genetic algorithm. *Expert Syst. Appl.* **2015**, *42*, 146–155. [CrossRef]
30. Dobson, D.C.; Vogel, C.R. Convergence of an iterative method for total variation denoising. *SIAM J. Numer. Anal.* **1997**, *34*, 1779–1791. [CrossRef]
31. Mangasarian, O.L. A generalized Newton method for absolute value equations. *Optim. Lett.* **2009**, *3*, 101–108. [CrossRef]
32. Feng, J.; Liu, S. A new two-step iterative method for solving absolute value equations. *J. Inequal. Appl.* **2019**, *2019*, 39. [CrossRef]
33. Traub, J.F. *Iterative Methods for the Solution of Equations*; Prentice Hall: New York, NY, USA, 1964.
34. Khan, A.; Iqbal, J.; Akgül, A.; Ali, R.; Du, Y.; Hussain, A.; Vijayakumar, V. A Newton-type technique for solving absolute value equations. *Alex. Eng. J.* **2023**, *64*, 291–296. [CrossRef]
35. Rahpeymaii, F.; Rostami, M. An iterative method based on Simpson's 3/8 rule to solve absolute value equations. *J. Math. Model.* **2025**, *13*, 553–562.
36. Feng, J.M.; Liu, S.Y. A three-step iterative method for solving absolute value equations. *J. Math.* **2020**, *2020*, 8531403. [CrossRef]
37. Jaiswal, J.P. Some class of third- and fourth-order iterative methods for solving nonlinear equations. *J. Appl. Math.* **2014**, *2014*, 817656. [CrossRef]
38. Ali, R.; Ali, A.; Khan, I.; Mohamed, A. Two new generalized iteration methods for solving absolute value equations using M-matrix. *AIMS Math.* **2022**, *7*, 8176–8187. [CrossRef]
39. Hu, S.L.; Huang, Z.H. A Note on Absolute Value Equations. *Optim. Lett.* **2010**, *4*, 417–424. [CrossRef]
40. Saheya, B.; Yu, C.H.; Chen, J.S. Numerical comparisons based on four smoothing functions for absolute value equation. *J. Appl. Math. Comput.* **2018**, *56*, 131–149. [CrossRef]
41. Gul, N.; Chen, H.; Iqbal, J.; Shah, R. A new two-step iterative technique for efficiently solving absolute value equations. *Eng. Comput.* **2024**, *41*, 1272–1284. [CrossRef]

42. Chen, G.; Xue, Y. Perturbation analysis for the operator equation $Tx = b$ in Banach spaces. *J. Math. Anal. Appl.* **1997**, *212*, 107–125. [CrossRef]
43. Shams, M.; Carpentieri, B. Efficient inverse fractional neural network-based simultaneous schemes for nonlinear engineering applications. *Fractal Fract.* **2023**, *7*, 849. [CrossRef]
44. Kim, J.; Kim, Y. A predicted Newton-Raphson iterative algorithm using neural network prediction. In Proceedings of the 41st Structures, Structural Dynamics, and Materials Conference, Atlanta, GA, USA, 3–6 April 2000; p. 1354.
45. Burden, R.L.; Faires, J.D.; Toomey, H.A. *Numerical Analysis Algorithms in C*; Brooks Cole: Pacific Grove, CA, USA, 1997.
46. Ke, Y.F.; Ma, C.F. SOR-like iteration method for solving absolute value equations. *Appl. Math. Comput.* **2017**, *311*, 195–202. [CrossRef]
47. Hoffman, J.D.; Frankel, S. *Numerical Methods for Engineers and Scientists*; CRC Press: Boca Raton, FL, USA, 2018.
48. Shams, M.; Kausar, N.; Araci, S.; Oros, G.I. Artificial hybrid neural network-based simultaneous scheme for solving nonlinear equations: Applications in engineering. *Alex. Eng. J.* **2024**, *108*, 292–305. [CrossRef]
49. Xie, J.; Qi, H.D.; Han, D. Randomized iterative methods for generalized absolute value equations: Solvability and error bounds. *SIAM J. Optim.* **2025**, *35*, 1731–1760. [CrossRef]
50. Vähäsöyrinki, M. Voltage-Gated K^+ Channels in Drosophila Photoreceptors: Biophysical Study of Neural Coding. Ph.D. Thesis, University of Oulu, Oulu, Finland, 2004.