

Physics-informed neural networks for high-dimensional solutions and snaking bifurcations in nonlinear lattices

Muhammad Luthfi Shahab^a, Fidya Almira Suheri^b, Rudy Kusdiantara^b,
Hadi Susanto^a

^a*Department of Mathematics, Khalifa University of Science & Technology, Abu Dhabi, PO Box 127788, United Arab Emirates*

^b*Industrial and Financial Mathematics Research Group, Institut Teknologi Bandung, Bandung, 40132, Indonesia*

Abstract

This paper introduces a framework based on physics-informed neural networks (PINNs) for addressing key challenges in nonlinear lattices, including solution approximation, bifurcation diagram construction, and linear stability analysis. We first employ PINNs to approximate solutions of nonlinear systems arising from lattice models, using the Levenberg–Marquardt algorithm to optimize network weights for greater accuracy. To enhance computational efficiency in high-dimensional settings, we integrate a stochastic sampling strategy. We then extend the method by coupling PINNs with a continuation approach to compute snaking bifurcation diagrams, incorporating an auxiliary equation to effectively track successive solution branches. For linear stability analysis, we adapt PINNs to compute eigenvectors, introducing output constraints to enforce positivity, in line with Sturm–Liouville theory. Numerical experiments are conducted on the discrete Allen–Cahn equation with cubic and quintic nonlinearities in one to five spatial dimensions. The results demonstrate that the proposed approach achieves accuracy comparable to, or better than, traditional numerical methods, especially in high-dimensional regimes where computational resources are a limiting factor. These findings highlight the potential of neural networks as scalable and efficient tools for the study of complex nonlinear lattice systems.

Keywords: Physics-informed neural networks, Nonlinear lattices, Bifurcation, Linear stability, High dimensions

1. Introduction

Neural networks have emerged as powerful tools for solving partial differential equations (PDEs) [1], largely due to their capacity to approximate com-

plex functions with high accuracy [2]. Foundational studies by Dissanayake and Phan-Thien [3], and Lagaris et al. [4], demonstrated that PDEs can be reformulated as optimization problems and solved using neural network architectures, marking a significant shift in computational mathematics. More recently, physics-informed neural networks (PINNs) [5] have advanced the field by incorporating governing physical laws directly into the training process, enabling the resolution of both forward and inverse problems across a range of domains, including fluid dynamics [6] and optics [7]. A key strength of neural networks lies in their ability to overcome the curse of dimensionality. Unlike traditional numerical methods, which become computationally infeasible in high-dimensional settings, neural networks offer scalable and efficient alternatives through optimization-based formulations [8, 9], thereby enabling the solution of previously intractable problems.

Extending these capabilities, recent studies have applied neural network methodologies to nonlinear lattices—discrete systems that arise in many physical contexts. Zhu et al. [10] employed symmetry-preserving neural architectures to model nonlinear dynamical lattices, focusing on the Ablowitz–Ladik system. Lin and Chen [11] introduced a pseudo-grid-based, physics-informed convolutional-recurrent network for integrable lattice equations, while Geng et al. [12] developed a separable graph Hamiltonian network to capture lattice dynamics via graph-based deep learning. Zhou et al. [13] integrated symmetric difference data enhancement into a PINN framework to improve accuracy for discrete nonlinear equations.

In parallel, alternative paradigms have broadened the scope of neural approaches. Saqlain et al. [14] proposed PINNs for discovering governing equations in discrete systems. Opala et al. [15] applied reservoir computing to the complex Ginzburg–Landau lattice model, successfully capturing a wide range of nonlinear dynamics. Stokes et al. [16] introduced first-quantized deep neural networks for analyzing strongly coupled fermionic lattice systems, further expanding the application of neural networks in quantum many-body physics. Together, these efforts underscore the growing versatility of deep learning techniques in tackling the complexities of nonlinear lattice models.

Nonlinear lattices play a central role in modeling phenomena in optics, condensed matter, and biological systems [17]. However, they pose substantial analytical and computational challenges, particularly in bifurcation analysis and stability characterization—both essential for understanding qualitative system behavior. These difficulties are compounded by intricate bifurcation structures and bistability, common features in discrete settings [18, 19], and demand precise, scalable computational strategies.

This study addresses a key gap in the application of PINNs to nonlinear lattice systems, with emphasis on high-dimensional cases, bifurcation

structures, and linear stability analysis. While PINNs have been successfully applied to related PDE-based tasks [20, 21, 22, 23], their use in discrete lattice contexts remains limited. Here, we extend PINN methodologies to these systems, particularly in the context of bifurcation tracking and stability diagnostics.

We also consider high-dimensional problems, where conventional solvers often fail due to the curse of dimensionality. PINNs offer an efficient alternative capable of capturing complex dynamics in such settings. Our analysis centers on the discrete Allen–Cahn equation with cubic and quintic nonlinearities [18], a benchmark model known for its rich bifurcation landscape, including snaking patterns with multiple turning points. This system provides a rigorous testbed for evaluating the effectiveness of neural-network-based solvers in nonlinear lattice dynamics.

The remainder of this paper is organized as follows. Section 2 introduces the discrete Allen–Cahn equation. Section 3 discusses the pseudo-arc length continuation method. Section 4 presents the proposed PINN framework, including its integration with pseudo-arc length continuation, a stochastic approach for high-dimensional problems, and a linear stability analysis module. Section 5 provides numerical results from one-dimensional to five-dimensional cases, demonstrating the versatility and performance of the proposed approach. Finally, Section 6 summarizes the key findings and outlines directions for future research.

2. Discrete Allen–Cahn Equation

We consider the discrete Allen–Cahn equation with cubic and quintic nonlinearities and a parameter μ . This model arises from the continuous PDE

$$\frac{\partial u}{\partial t} = \mu u + \Delta u + 2u^3 - u^5, \quad (1)$$

by discretizing the spatial domain. Specifically, the continuous variable u is replaced by its discrete counterpart u_i defined on a one-dimensional lattice, $i = 1, \dots, n$, and the Laplacian Δu is approximated using the finite difference operator $c\Delta_1 u_i = c(u_{i+1} - 2u_i + u_{i-1})$, where $c > 0$ is a fixed coupling parameter. This yields the one-dimensional discrete Allen–Cahn equation:

$$\frac{du_i}{dt} = \mu u_i + c\Delta_1 u_i + 2u_i^3 - u_i^5, \quad i \in \{2, \dots, n-1\}, \quad (2)$$

with boundary conditions $u_1 = u_n = 0$.

The two-dimensional version is defined on a lattice (i, j) , where $i, j \in \{1, \dots, n\}$. The equation becomes

$$\frac{du_{i,j}}{dt} = \mu u_{i,j} + c\Delta_2 u_{i,j} + 2u_{i,j}^3 - u_{i,j}^5, \quad i, j \in \{2, \dots, n-1\}, \quad (3)$$

where

$$c\Delta_2 u_{i,j} = c(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}), \quad (4)$$

and boundary conditions are $u_{i,j} = 0$ for $i \in \{1, n\}$ or $j \in \{1, n\}$. Higher-dimensional versions (three to five dimensions) can be defined analogously by extending the discrete Laplacian accordingly.

In all cases, we are interested in the steady-state solutions obtained by setting the time derivatives to zero. This reduces the system to a set of nonlinear algebraic equations. For one- and two-dimensional cases, we additionally analyze the bifurcation structure and linear stability of the steady states. The discrete Allen–Cahn model is particularly well-suited for this purpose due to its characteristic snaking bifurcations with multiple turning points [18], making it an effective benchmark for continuation methods.

The steady states of interest are symmetric about the midpoint of the lattice [18]. To capture this, the lattice size is commonly chosen as $n = 2m-1$ for site-centered solutions, or $n = 2m$ for bond-centered solutions. These configurations allow for a detailed comparison of solution types and their respective bifurcation characteristics.

2.1. Eigenvalue Problem for Linear Stability

To determine the linear stability of a steady-state solution u to Eq. (2), we consider a small perturbation of the form

$$\tilde{u}_i(t) = u_i + \varepsilon w_i(t), \quad |\varepsilon| \ll 1. \quad (5)$$

Substituting this into Eq. (2) and linearizing in ε yields the equation for the perturbation:

$$\frac{dw_i}{dt} = \mu w_i + c\Delta_1 w_i + 6u_i^2 w_i - 5u_i^4 w_i, \quad i \in \{2, \dots, n-1\}. \quad (6)$$

Assuming a separable solution $w_i(t) = e^{\lambda t} v_i$, we arrive at the eigenvalue problem:

$$\lambda v_i = \mu v_i + c\Delta_1 v_i + 6u_i^2 v_i - 5u_i^4 v_i, \quad i \in \{2, \dots, n-1\}, \quad (7)$$

subject to boundary conditions $v_1 = v_n = 0$, where λ is the eigenvalue and $v = (v_i)$ is the corresponding eigenvector.

A steady-state solution is linearly stable if the largest eigenvalue satisfies $\max(\lambda) < 0$; otherwise, it is unstable. The formulation for higher dimensions follows analogously. Notably, Eq. (7) can be viewed as a discrete Sturm–Liouville problem. To determine $\max(\lambda)$, we identify the critical eigenvalue associated with an eigenvector that has no internal sign changes.

3. Pseudo-arc length Continuation

In parameter-dependent nonlinear systems, direct continuation by varying the parameter may fail to trace the full bifurcation structure, especially in the presence of turning points. At such points, the solution curve folds back on itself, and the parameter is no longer a monotonic or uniquely defining quantity for the solution. Standard continuation methods often fail here due to the breakdown of the one-to-one correspondence between the parameter and the solution.

Pseudo-arc length continuation overcomes this limitation by reparameterizing the solution curve using a pseudo-arc length measure, allowing the method to follow the solution branch through turning points. Instead of advancing the parameter directly, the method controls the step direction along the curve itself via an auxiliary constraint.

Consider a system of nonlinear equations $F(u, \mu) = 0$, where u is the state variable and μ is a bifurcation parameter. For simplicity, denote a solution as $(u^{[k]}, \mu^{[k]})$, where $u^{[k]}$ satisfies $F(u, \mu^{[k]}) = 0$. To construct the bifurcation diagram, we begin with two known solutions, $(u^{[1]}, \mu^{[1]})$ and $(u^{[2]}, \mu^{[2]})$, and generate subsequent solutions using a predictor-corrector scheme.

The predictor step uses the extrapolation:

$$(u^{[k]}, \mu^{[k]}) = 2(u^{[k-1]}, \mu^{[k-1]}) - (u^{[k-2]}, \mu^{[k-2]}), \quad (8)$$

which provides an initial guess for the k -th solution. To enable correction and enforce continuation along the solution branch, we introduce a constraint equation of the form:

$$\alpha \left(\sqrt{[\beta_1 (\|u^{[k]}\| - (\|u^{[k-1]}\| + \gamma))]^2 + [\beta_2 (\mu^{[k]} - \mu^{[k-1]})]^2} - \delta \right) = 0. \quad (9)$$

This auxiliary condition serves two main purposes, depending on the choice of parameters:

- **Norm-based continuation:** When $\beta_2 = \delta = 0$, $\beta_1 = 1$, and $\gamma \neq 0$, the continuation enforces $\|u^{[k]}\| = \|u^{[k-1]}\| + \gamma$. This approach is effective when each solution norm corresponds to a unique solution, as

seen in problems such as the Bratu equation [24], the Burgers equation [25, 22], and nonlinear boundary value problems [26, 27]. We adopt this formulation for the one-dimensional discrete Allen–Cahn equation.

- **Arclength-based continuation:** When $\beta_1, \beta_2 \neq 0$ and $\gamma = 0$, the constraint enforces a fixed pseudo-arclength δ between successive steps in the $(\|u\|, \mu)$ plane. This formulation is better suited for general bifurcation tracking when the norm and parameter operate on different scales. We apply this approach for the two-dimensional discrete Allen–Cahn equation.

Here, α is a weighting factor balancing the continuation constraint with the original system $F(u, \mu) = 0$. When $\delta = 1$, the square root in Eq. (9) may be omitted for computational simplicity. The effects of tuning parameters α , γ , and (β_1, β_2) are discussed in Appendices [Appendix A](#) and [Appendix B](#).

The continuation proceeds iteratively until a stopping condition is met (e.g., reaching $k = k_{\max}$ or covering the desired range of solutions). For improved convergence, the initial solutions $(u^{[1]}, \mu^{[1]})$ and $(u^{[2]}, \mu^{[2]})$ should approximately satisfy Eq. (9).

Each correction step requires solving the augmented system consisting of $F(u, \mu) = 0$ and Eq. (9). Newton’s method can be applied, with iterations given by [28]:

$$u^{(k+1)} = u^{(k)} - J(u^{(k)})^{-1}F(u^{(k)}), \quad (10)$$

where $J(u^{(k)})$ is the Jacobian of F evaluated at $u^{(k)}$. The dominant computational cost lies in solving the linear system involving $J(u^{(k)})$.

In this work, we employ the Levenberg–Marquardt algorithm [29, 30], a regularized variant of Newton’s method, given by:

$$u^{(k+1)} = u^{(k)} - (J^T J + \lambda^{(k)} I)^{-1} J^T F(u^{(k)}), \quad (11)$$

where $\lambda^{(k)}$ is a damping parameter ensuring that each iteration proceeds in a descent direction. When solving the combined system $F(u, \mu) = 0$ and Eq. (9), the parameter μ is treated as an additional variable and updated alongside u .

We use $u^{(k)}$ to denote the k -th iterate produced by the solver, and $u^{[k]}$ to represent the k -th converged solution that satisfies $F(u, \mu) = 0$. This notation distinguishes from u^k , which denotes exponentiation.

4. Proposed Methods

4.1. PINNs for Nonlinear Lattices

Instead of solving the nonlinear system directly using traditional numerical solvers, this study proposes the use of physics-informed neural networks

(PINNs) to approximate solutions. While PINNs have been extensively applied to continuous partial differential equations (PDEs), their adaptation to discrete nonlinear lattice problems remains comparatively underexplored.

We begin with a general nonlinear system of equations of the form

$$f_{\mathbf{i}}(u, \mu) = 0, \quad \mathbf{i} \in A, \quad (12)$$

where $u = \{u_{\mathbf{i}} \mid \mathbf{i} \in A\}$ denotes the set of unknowns, μ is a fixed parameter, and A is the index set corresponding to the discrete lattice points. The multi-index \mathbf{i} may represent one or more spatial indices, such as (i_1, i_2) or (i, j) , depending on the system dimension.

Unlike continuous PDE problems where the network input corresponds to spatial or temporal coordinates, nonlinear lattice systems lack explicit spatial variables. To overcome this, we reinterpret the lattice index \mathbf{i} as an input to the network and denote the network output as $u(\mathbf{i})$.

Given this formulation, a neural network is constructed to approximate the solution as $u(\mathbf{i}, W)$, where W represents the weights and biases of the network. The goal is to find W such that

$$f_{\mathbf{i}}(u(\mathbf{i}, W), \mu) = 0, \quad \mathbf{i} \in A. \quad (13)$$

This system can be interpreted as a root-finding problem in the space of network parameters. To measure the discrepancy, we define the mean squared error (MSE) as:

$$\text{MSE} = \frac{1}{|A|} \sum_{\mathbf{i} \in A} [f_{\mathbf{i}}(u(\mathbf{i}, W), \mu)]^2, \quad (14)$$

where $|A|$ denotes the number of lattice points in the system.

To determine the optimal weights W , two primary strategies can be employed: (1) minimize the loss function in Eq. (14) using optimization algorithms such as L-BFGS [31] or Adam [32], or (2) solve the system in Eq. (13) directly using a nonlinear solver. In this study, we adopt the second approach and apply the Levenberg–Marquardt algorithm [29, 30], which offers improved accuracy and convergence for small- to medium-sized networks.

The neural networks used in this work are shallow, comprising two hidden layers. Let W_1, B_1, W_2, B_2, W_3 , and B_3 represent the weights and biases of the network. Then the output can be expressed as:

$$u(\mathbf{i}, W) = \sigma(\sigma(\mathbf{i}W_1 + B_1)W_2 + B_2)W_3 + B_3, \quad (15)$$

where σ is the activation function. Following the success of previous work in nonlinear bifurcation problems [20], we adopt the Gaussian activation function $\sigma(x) = e^{-x^2/2}$ throughout this study.

4.2. Input Manipulation

In our implementation, we found that transforming the input index \mathbf{i} before feeding it into the neural network significantly improves convergence. Experimental results indicate that mapping discrete indices to a smaller, normalized domain enhances the training efficiency and stability of PINNs. This improvement is motivated by two key observations:

- Most activation functions perform optimally when their inputs lie within a bounded range. Large input indices can lead to extreme values when propagated through network layers, potentially causing vanishing gradients. Normalizing the input ensures that the network operates within the sensitive regions of its activation functions, which accelerates training and improves accuracy.
- PINNs are known to struggle with problems defined over large physical or index domains [33, 34, 35]. These difficulties stem from highly non-convex loss landscapes and gradient pathologies. Scaling the input domain helps mitigate these issues and supports more robust optimization.

To formalize this procedure, consider a two-dimensional nonlinear lattice where $\mathbf{i} = (i_1, i_2)$ or (i, j) and $i_1, i_2 \in \{1, \dots, n\}$. We apply the transformation:

$$\bar{i}_1 = (i_1 - 1)h, \quad \bar{i}_2 = (i_2 - 1)h, \quad (16)$$

where $h = 1/(n - 1)$, so that the original index range $[1, n]$ is mapped to the normalized interval $[0, 1]$. The PINN then approximates the solution as $u(\mathbf{i}, W)$ using the transformed inputs.

Further refinements are possible when specific structural properties of the solution are known. For example, solutions of the discrete Allen–Cahn equation exhibit reflection symmetry about the midpoint of the lattice [18]. To exploit this, we propose an additional symmetry-aware transformation:

$$\begin{aligned} \hat{i}_1 &= 0.5 - |\bar{i}_1 - 0.5|, & \hat{i}_2 &= 0.5 - |\bar{i}_2 - 0.5|, \\ (\tilde{i}_1, \tilde{i}_2) &= \text{sort}(\hat{i}_1, \hat{i}_2). \end{aligned} \quad (17)$$

Here, 0.5 represents the midpoint of the normalized domain. This transformation maps symmetrically equivalent inputs—those corresponding to equal solution values—to a canonical form within the region $0 \leq \tilde{i}_1 \leq \tilde{i}_2 \leq 0.5$. As a result, it reduces the complexity of the optimization process, leading to faster convergence during training.

This preprocessing step incurs no additional cost during inference, as the transformations can be computed and cached before training begins. Further details and illustrations of this approach are provided in [36].

4.3. Pseudo-arc length Continuation

When constructing bifurcation diagrams using pseudo-arc length continuation, the objective is to compute a sequence of solution-parameter pairs $(u^{[k]}, \mu^{[k]})$. However, since the solution u is represented implicitly by a PINN through its weights W , we instead compute a sequence $(W^{[k]}, \mu^{[k]})$, where each $W^{[k]}$ defines an approximate solution $u(\mathbf{i}, W^{[k]})$.

To integrate pseudo-arc length continuation within the PINN framework, we first determine two initial solutions, $(W^{[1]}, \mu^{[1]})$ and $(W^{[2]}, \mu^{[2]})$, that satisfy the nonlinear system in Eq. (13). For each subsequent step $k \geq 3$, the parameter $\mu^{[k]}$ is treated as an additional trainable variable alongside the neural network weights. The predictor step is initialized by extrapolation:

$$(W^{[k]}, \mu^{[k]}) = 2(W^{[k-1]}, \mu^{[k-1]}) - (W^{[k-2]}, \mu^{[k-2]}). \quad (18)$$

Using the predicted pair $(W^{[k]}, \mu^{[k]})$ as an initial guess, we retrain the network to satisfy both the nonlinear system and the continuation constraint:

$$\begin{aligned} f_{\mathbf{i}}(u(\mathbf{i}, W^{[k]}), \mu^{[k]}) &= 0, \quad \mathbf{i} \in A, \\ \alpha \left(\sqrt{[\beta_1 (\|u(\mathbf{i}, W^{[k]})\| - (\|u(\mathbf{i}, W^{[k-1]})\| + \gamma))]^2 + [\beta_2 (\mu^{[k]} - \mu^{[k-1]})]^2} - \delta \right) &= 0. \end{aligned} \quad (19)$$

Once the updated pair $(W^{[k]}, \mu^{[k]})$ is obtained, we evaluate $u(\mathbf{i}, W^{[k]})$ to plot the corresponding point on the bifurcation diagram. This process is iterated to trace the solution branch across turning points and parameter regimes of interest.

While the overall framework is conceptually straightforward, its implementation poses several technical challenges. These include:

- Treating μ as a trainable parameter integrated into the network architecture.
- Initializing network weights via Eq. (18), which is not standard in conventional machine learning workflows.
- Solving the nonlinear system using the Levenberg–Marquardt algorithm, as opposed to minimizing a typical loss function.

These requirements render standard machine learning libraries such as Scikit-learn [37] or high-level TensorFlow APIs [38] unsuitable. While functional (low-level) TensorFlow could support this approach, doing so would involve complex, non-standard manipulation of weights and control logic to

implement Eqs. (18) and (19) directly. For these reasons, we developed a custom implementation in MATLAB, which provides the flexibility necessary to accommodate the algorithmic structure and optimization routine specific to our method.

4.4. Stochastic Newton's Method

As discussed earlier, the PINN is trained to satisfy a nonlinear system using the Levenberg–Marquardt algorithm [29, 30], which is based on Newton's method. In this subsection, we introduce a simple yet effective modification inspired by the mini-batching technique in stochastic gradient descent (SGD). Rather than using the full system of equations at every iteration, we operate on randomly selected subsets of equations, significantly reducing computational cost. This stochastic strategy is particularly applied to the five-dimensional discrete Allen–Cahn equation.

In the standard Newton's method, the network weights W are iteratively updated as:

$$W^{(k+1)} = W^{(k)} - J(W^{(k)})^{-1}F(W^{(k)}), \quad (20)$$

where $J(W^{(k)})$ is the Jacobian of the nonlinear system F evaluated at $W^{(k)}$. However, in high-dimensional settings, the computation and inversion of the full Jacobian become prohibitively expensive.

To address this, we adopt a stochastic approach. At each iteration, we randomly select a subset S_k of the system equations and compute the corresponding Jacobian and residuals. This leads to the update rule:

$$W^{(k+1)} = W^{(k)} - J_{S_k}(W^{(k)})^{-1}F_{S_k}(W^{(k)}), \quad (21)$$

where F_{S_k} and J_{S_k} denote the restriction of F and J to the subset S_k . Although the approximation to the full Jacobian is less accurate, the reduced computational load per iteration enables faster convergence in practice.

This modification integrates naturally with neural networks since all weights contribute to computing F_{S_k} and J_{S_k} . Similar ideas have been studied in the context of optimization [39, 40], though most previous work focuses on loss minimization rather than solving nonlinear systems directly.

When using the Levenberg–Marquardt algorithm, the standard update becomes:

$$W^{(k+1)} = W^{(k)} - (J^T J + \lambda^{(k)} I)^{-1} J^T F(W^{(k)}), \quad (22)$$

which we adapt to the stochastic setting as:

$$W^{(k+1)} = W^{(k)} - (J_{S_k}^T J_{S_k} + \lambda^{(k)} I)^{-1} J_{S_k}^T F_{S_k}(W^{(k)}). \quad (23)$$

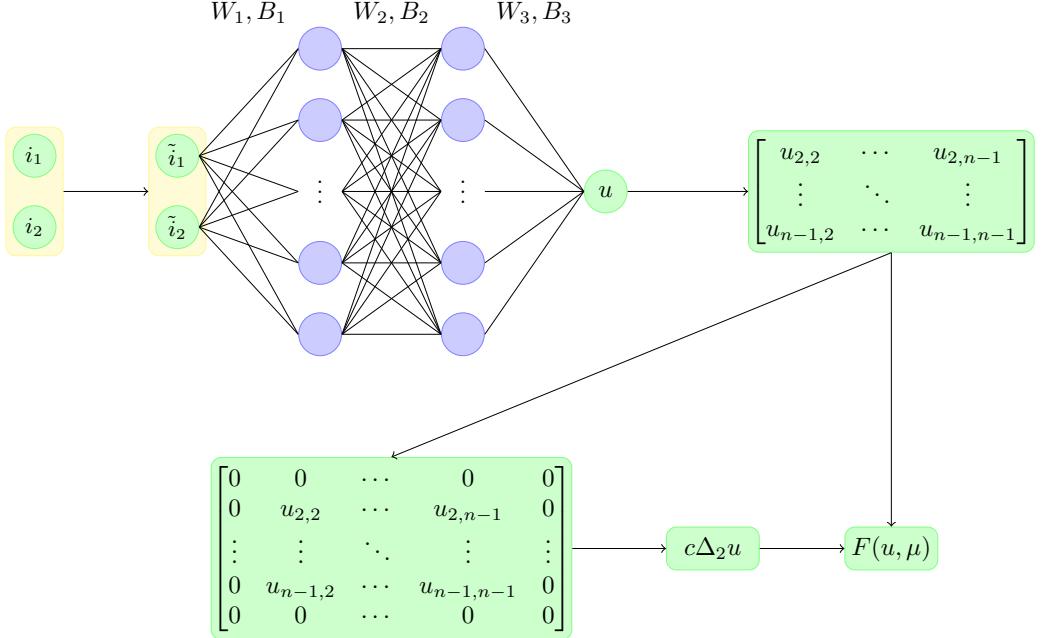


Figure 1: Schematic of the PINN architecture for solving nonlinear lattice systems. Although the diagram illustrates a two-dimensional lattice, the method generalizes to higher dimensions by extending the input space accordingly.

It is important to note that this stochastic method is particularly suited to neural-network-based formulations. In traditional finite-difference or finite-element contexts, functions in F are typically local, depending only on adjacent variables. Therefore, randomly sampling equations may fail to update all variables, limiting convergence. Neural networks, by contrast, maintain global parameter coupling, making the stochastic method viable for high-dimensional problems.

4.5. Computing the Nonlinear System and Treating Boundary Points

For one- to four-dimensional problems, we evaluate the full set of nonlinear equations in each iteration of the Levenberg–Marquardt algorithm. A d -dimensional matrix u is first constructed to represent the solution domain, where d is the number of spatial dimensions. Boundary conditions are enforced by explicitly setting $u_{\mathbf{i}} = 0$ at all boundary indices, eliminating the need for neural network evaluation at those locations. For interior points $\mathbf{i} \in A$, the PINN is used to compute $u_{\mathbf{i}} = u(\mathbf{i}, W)$. This approach allows the nonlinear system to be efficiently evaluated using matrix operations.

In the five-dimensional case, due to memory and computational constraints, we employ the stochastic Newton’s method described above. At

each iteration, only a subset S_k of the nonlinear equations is evaluated. Consequently, we avoid forming a full five-dimensional matrix. Instead, we construct an input vector corresponding to the points in S_k , along with 10 additional vectors representing the neighboring lattice sites. The PINN then computes the solution values $u(\mathbf{i}, W)$ for all sampled and adjacent points.

Some of these indices may lie on the boundary. To enforce zero boundary conditions in this case, we apply a multiplicative mask to the neural network output using:

$$u(\mathbf{i}, W) \leftarrow u(\mathbf{i}, W) \cdot \prod_{k=1}^5 \sin(\pi i_k), \quad (24)$$

for $\mathbf{i} = (i_1, i_2, i_3, i_4, i_5)$. This transformation ensures that the output vanishes on the boundary while preserving flexibility in the interior. The nonlinear system is evaluated using vectorized operations for scalability and computational efficiency.

Figure 1 illustrates the general structure of the PINN used in our framework. While the diagram focuses on a two-dimensional lattice for clarity, the extension to higher dimensions is straightforward: the number of input coordinates increases, and the internal matrix representation is adapted accordingly.

4.6. Largest Eigenvalue

Consider the general eigenvalue problem

$$H(v, u, \mu) = \lambda v, \quad (25)$$

where u and μ are fixed, λ is an eigenvalue, and v is the corresponding eigenvector. This equation can be reformulated as a nonlinear system:

$$G(v, \lambda, u, \mu) = H(v, u, \mu) - \lambda v = 0. \quad (26)$$

To solve this system, we adopt a PINN-based approach similar to that described in Section 4.1. Our objective is to compute the *largest* eigenvalue, which is critical for linear stability analysis as noted in Section 2.1. According to Sturm–Liouville theory, the dominant eigenvalue is associated with an eigenvector that does not change sign. To enforce this non-negativity, we apply an absolute value transformation to the neural network output. Using the same notation for weights and biases, the neural network is defined as:

$$v(\mathbf{i}, W) = |\sigma(\sigma(\mathbf{i}W_1 + B_1)W_2 + B_2)W_3 + B_3|, \quad (27)$$

where σ denotes the activation function.

If the system G consists of multiple component equations $g_{\mathbf{i}}$, indexed over a set $\mathbf{i} \in A$, the PINN is trained to satisfy:

$$\begin{aligned} g_{\mathbf{i}}(v(\mathbf{i}, W), \lambda, u, \mu) &= 0, & \mathbf{i} \in A, \\ \|v\|_2 &= 1, \end{aligned} \tag{28}$$

where the normalization constraint $\|v\|_2 = 1$ ensures uniqueness of the eigenvector. In this setup, λ is treated as an additional trainable parameter, jointly optimized with the network weights W .

Upon convergence, the solution λ obtained from Eq. (28) corresponds to the largest eigenvalue, consistent with the Sturm–Liouville theory. This PINN-based formulation offers a general and flexible framework for computing dominant eigenvalues in high-dimensional, structured systems.

5. Experimental Results and Discussion

We use the Levenberg–Marquardt algorithm [29, 30], both for direct solution of the nonlinear system and via the PINN-based formulations described in Section 4. Solutions obtained through direct numerical methods serve as ground truth for benchmarking the accuracy of the PINN approximations. All simulations were performed using MATLAB R2024a on a laptop equipped with an Intel Core i7-1185G7 processor.

5.1. One-dimensional Discrete Allen–Cahn Equation

We begin with the one-dimensional discrete Allen–Cahn equation, defined in Eq. (2). It is well established that this equation exhibits characteristic snaking bifurcation structures. For our experiments, we set $m = 10$, resulting in lattice sizes of $n = 19$ and $n = 20$ for the site- and bond-centered configurations, respectively. Excluding boundary points, the Jacobian matrices for direct solution have dimensions 17×17 and 18×18 .

For the PINN approach, we use a network architecture with one input, two hidden layers of four neurons each, and one output. This yields a total of 33 trainable parameters, including biases. The Jacobian matrices for this configuration expand to 17×33 and 18×33 for the site- and bond-centered cases, respectively. Reducing the network size adversely affects accuracy. Notably, the relative increase in Jacobian size occurs only in low-dimensional settings with small lattice sizes; in higher-dimensional problems, the PINN Jacobians are often more compact relative to the original system size.

Bifurcation diagrams are constructed using the norm-based continuation with the following parameters: $\alpha = 10$, $\beta_1 = 1$, $\beta_2 = 0$, $\gamma = 10/1000$, and

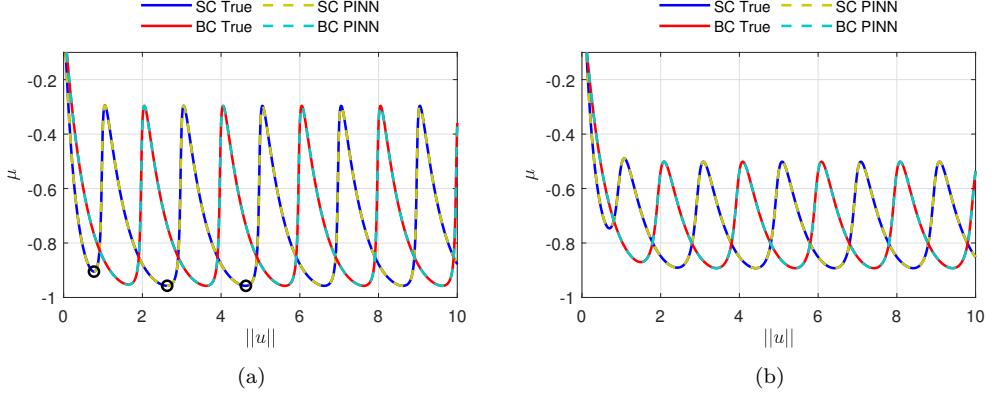


Figure 2: Snaking bifurcation diagrams of the one-dimensional discrete Allen–Cahn equation for (a) $c = 0.05$ and (b) $c = 0.15$. SC: site-centered, BC: bond-centered. Black circles mark the solutions shown in Figure 3.

$\delta = 0$. The solution norm is defined as

$$\|u\| = \frac{1}{1 + \sqrt{1 + \mu}} \sum_{i=1}^n u_i^2, \quad (29)$$

and diagrams are computed up to $\|u\| = 10$ for both solution types. Sensitivity to α and γ is explored in Appendix [Appendix A](#).

The initial solutions at $\mu = -0.1$ and $\mu = -0.12$ serve as starting points for continuation. Figure 2 shows the resulting bifurcation diagrams for $c = 0.05$ and $c = 0.15$. The PINN-generated curves closely match the ground-truth solutions, confirming the method’s ability to capture the underlying bifurcation structure accurately across both site- and bond-centered configurations.

Figure 3 presents three representative solutions for $c = 0.05$, corresponding to the marked locations in Figure 2. Subfigures (a), (c), and (e) show the true and PINN solutions, while (b), (d), and (f) display the absolute differences. In all cases, the error remains below 5×10^{-12} , highlighting the exceptional accuracy of the proposed method.

Figure 4 displays the largest eigenvalue as a function of $\|u\|$ for both cases. The eigenvalues were computed using the PINN-based Sturm–Liouville approach described in Section 4.6. The transitions between stability and instability are clearly observed. Finally, Figure 5 presents the stability classification of the bifurcation branches. Derived from the eigenvalue analysis in Figure 4, stable and unstable regions are depicted using thick and thin lines, respectively. The alternating pattern of stability along the branches is consistent with the known behavior of snaking bifurcations.

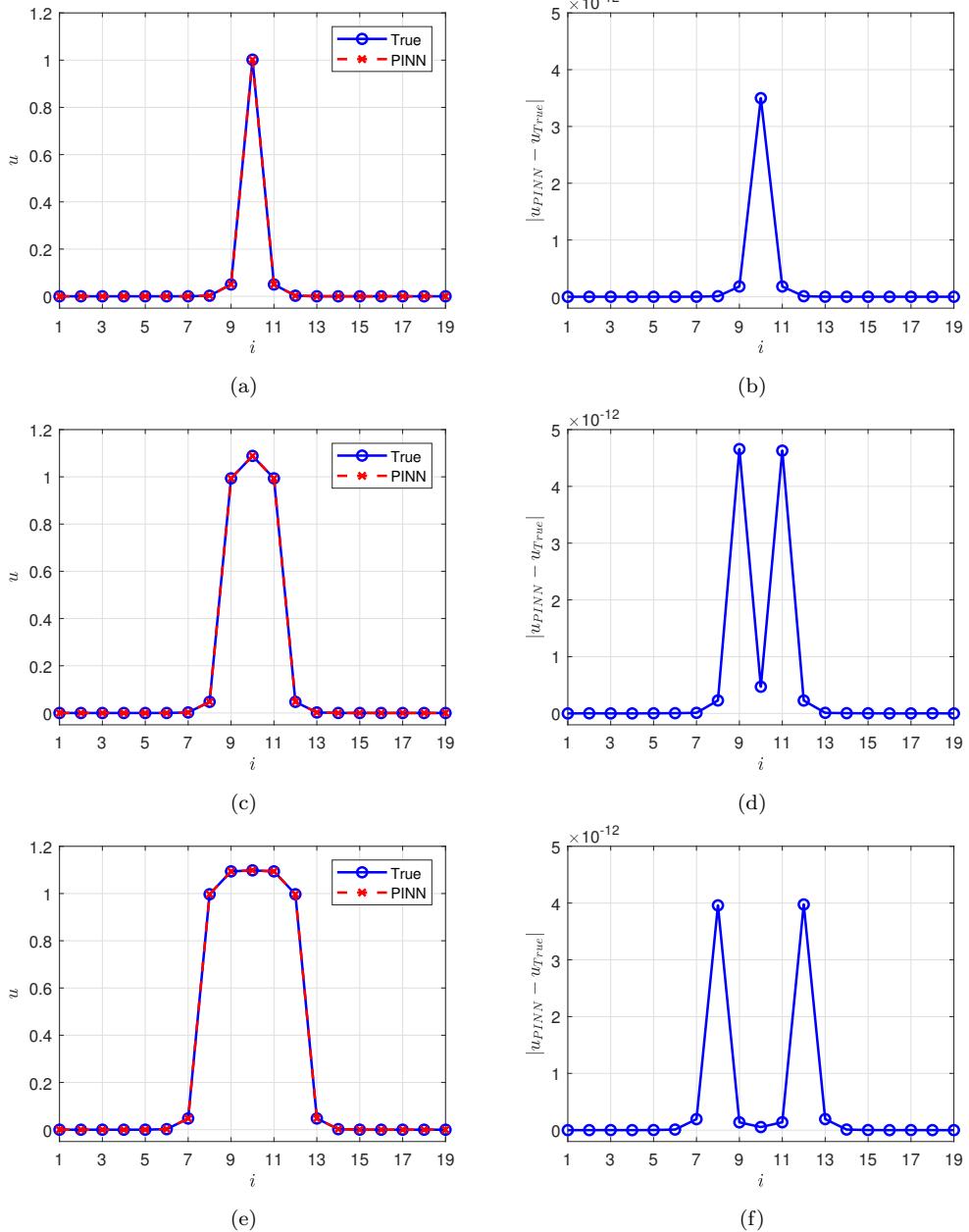


Figure 3: (a), (c), (e): True and PINN solutions corresponding to the marked points in Figure 2 for $c = 0.05$. (b), (d), (f): Absolute errors between true and PINN solutions.

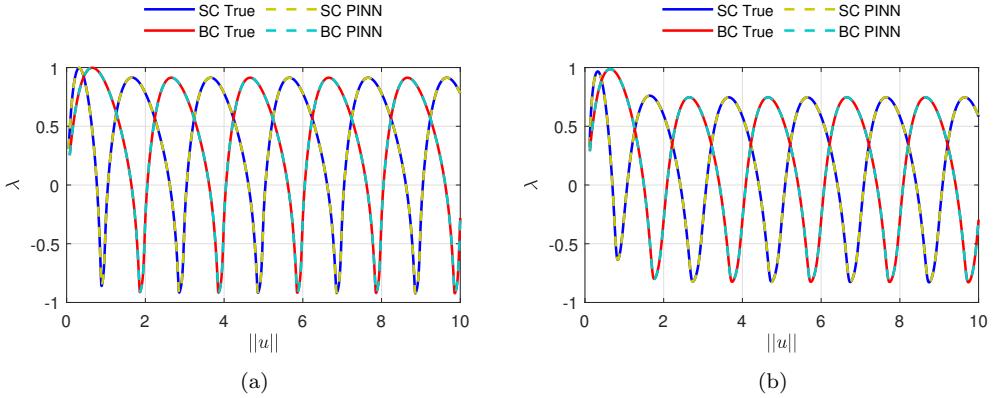


Figure 4: Largest eigenvalues versus $\|u\|$ for the one-dimensional discrete Allen–Cahn equation: (a) $c = 0.05$, (b) $c = 0.15$.

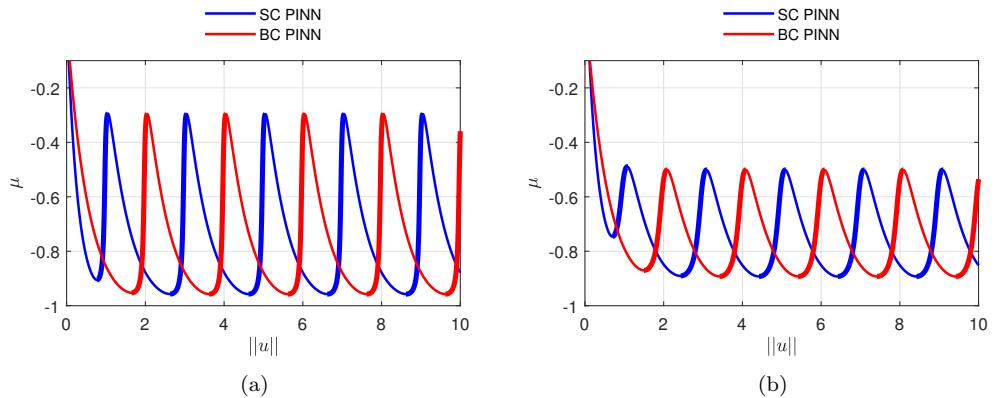


Figure 5: Bifurcation diagrams with linear stability indicated: (a) $c = 0.05$, (b) $c = 0.15$. Thick lines: stable branches; thin lines: unstable branches.

5.2. Two-dimensional Discrete Allen–Cahn Equation

We now extend our analysis to the two-dimensional discrete Allen–Cahn equation, defined in Eq. (3). Compared to the one-dimensional case, this system exhibits more intricate and irregular snaking bifurcations, as observed in [18]. For the simulation, we set $m = 8$, resulting in lattice sizes of 15×15 and 16×16 for the site- and bond-centered cases, respectively. Excluding boundary points, the corresponding Jacobian matrices for direct solution have dimensions 169×169 and 196×196 .

We employ a neural network with two inputs, two hidden layers containing seven neurons each, and one output. This architecture comprises 85 trainable parameters, including biases. With PINNs, the Jacobian matrix dimensions are reduced to 169×85 and 196×85 for the site- and bond-centered cases,

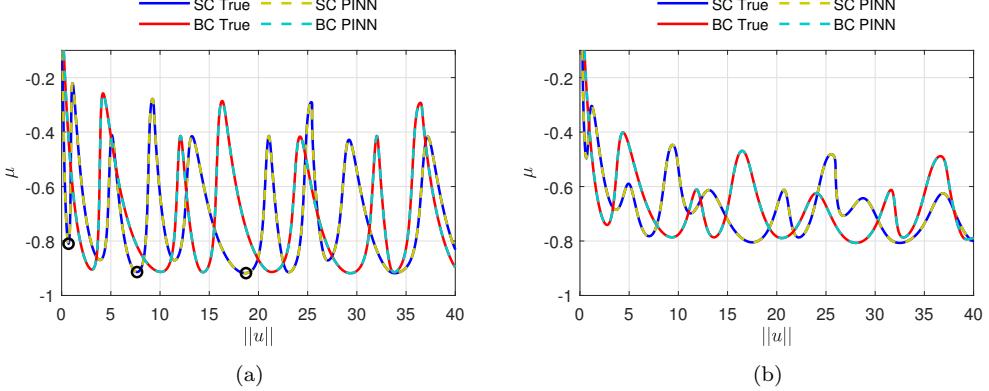


Figure 6: Snaking bifurcation diagrams of the two-dimensional discrete Allen–Cahn equation for (a) $c = 0.05$ and (b) $c = 0.15$. SC: site-centered, BC: bond-centered. Black circles mark the solutions shown in Figure 7.

respectively. Moreover, in the Levenberg–Marquardt algorithm, the matrix $(J^T J + \lambda I)$ further reduces to size 85×85 , regardless of configuration.

To construct the bifurcation diagrams, we use the arclength-based continuation with the following parameters: $\alpha = 10$, $\beta_1 = 1000/40$, $\beta_2 = 100$, $\gamma = 0$, and $\delta = 1$. The solution norm is defined as

$$\|u\| = \frac{1}{1 + \sqrt{1 + \mu}} \sum_{i=1}^n \sum_{j=1}^n u_{i,j}^2 \quad (30)$$

and the diagrams are computed up to $\|u\| = 40$ for both solution types. Sensitivity analysis for β_1 and β_2 is provided in Appendix [Appendix B](#).

The diagrams are initiated using two solutions at $\mu = -0.1$ and $\mu = -0.1 - 1/\beta_2$. These serve as starting points for the pseudo-arclength continuation. Figure 6 presents the resulting bifurcation diagrams for $c = 0.05$ and $c = 0.15$. In both site- and bond-centered cases, the PINN-generated results are nearly indistinguishable from those of the direct method, demonstrating the model’s effectiveness. Unlike the one-dimensional case, the snaking patterns in two dimensions are more complex and irregular, underscoring the challenge of accurate bifurcation tracking in higher dimensions.

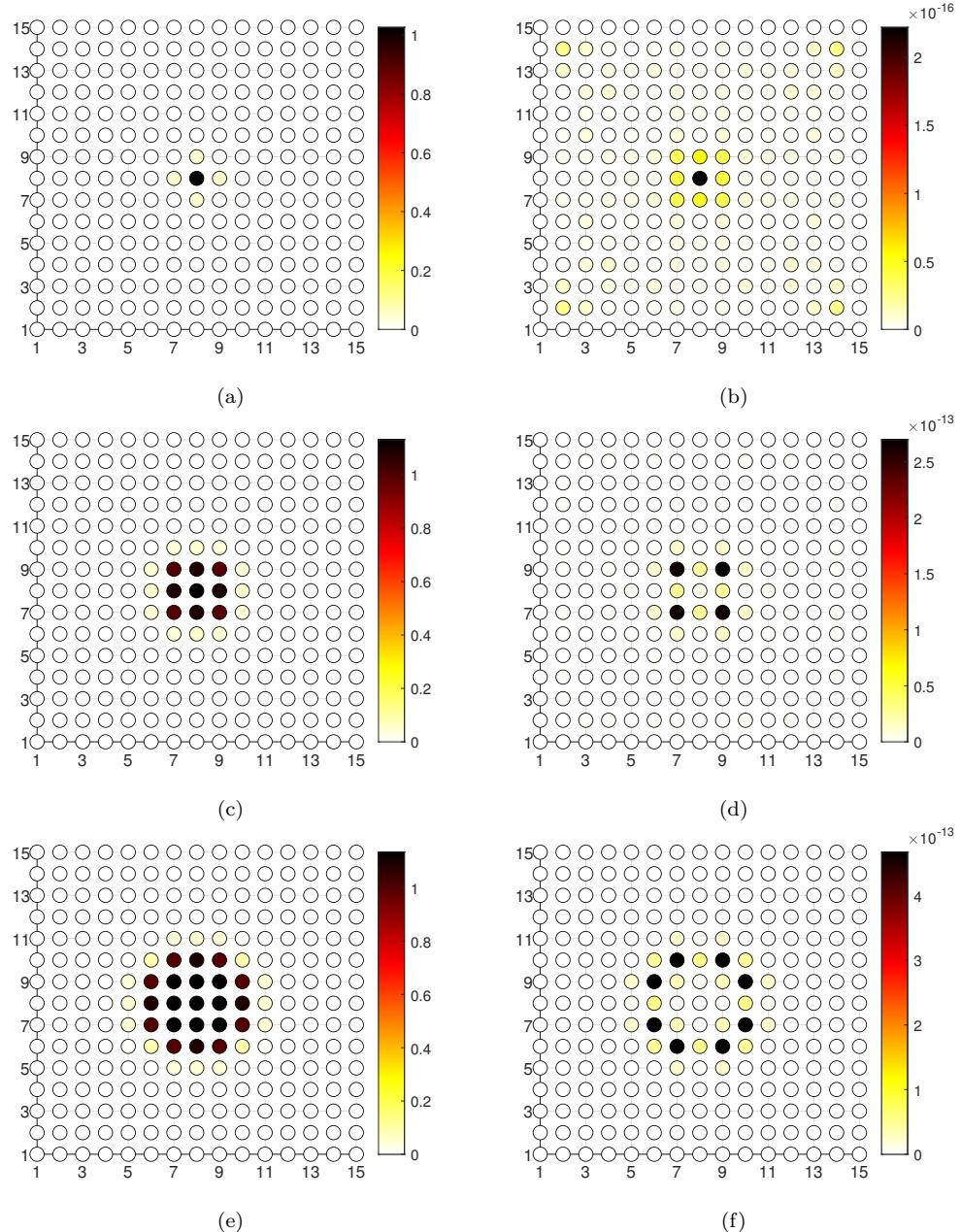


Figure 7: (a), (c), (e): PINN solutions corresponding to black circles in Figure 6 for $c = 0.05$. (b), (d), (f): Absolute difference between true and PINN solutions.

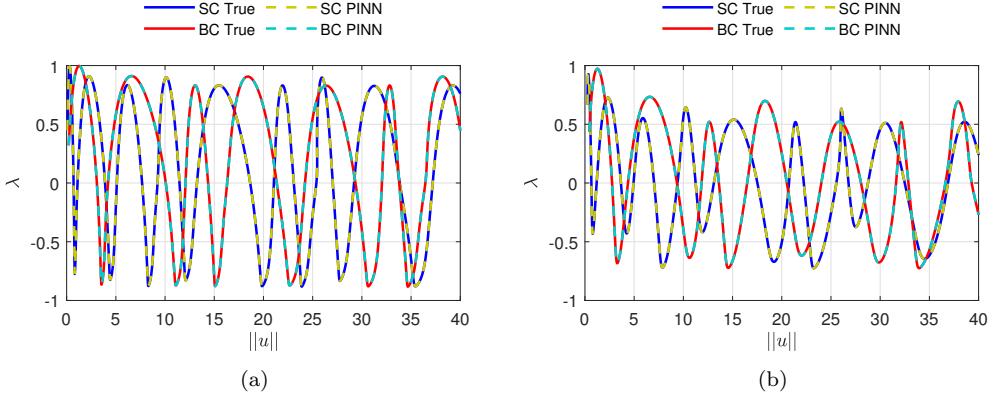


Figure 8: Largest eigenvalue versus $\|u\|$ for the two-dimensional discrete Allen–Cahn equation: (a) $c = 0.05$, (b) $c = 0.15$. SC: site-centered, BC: bond-centered.

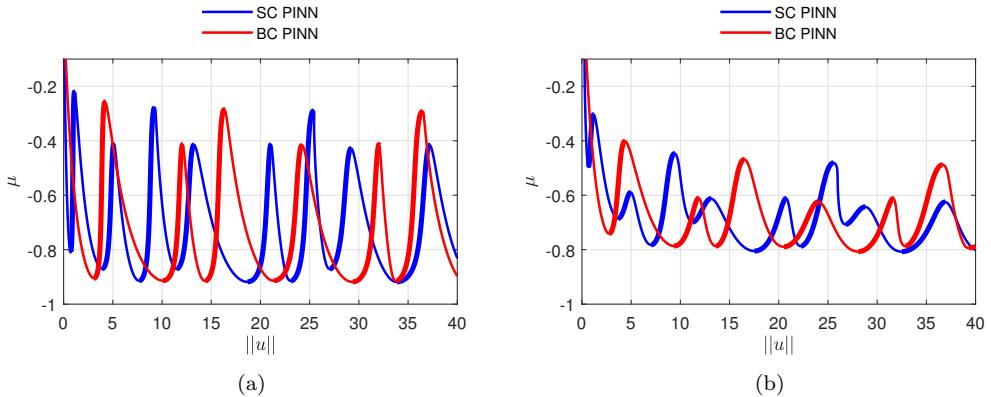


Figure 9: Snaking bifurcation diagrams with linear stability indicated for (a) $c = 0.05$ and (b) $c = 0.15$. SC: site-centered, BC: bond-centered. Thick lines: stable branches; thin lines: unstable branches.

Figure 7 shows three representative solutions at selected values of $\|u\|$ for $c = 0.05$, marked in Figure 6. The left column displays the PINN approximations, while the right column presents the absolute errors compared to the true solutions. In all instances, the absolute error remains below 5×10^{-13} , confirming the exceptional precision of the PINN method.

Figure 8 displays the largest eigenvalue as a function of $\|u\|$, computed using the PINN-based Sturm–Liouville approach. As expected, the eigenvalues fluctuate with the bifurcation branches, reflecting the stability characteristics of the solutions. Finally, Figure 9 shows the linear stability classification derived from the eigenvalue analysis in Figure 8. As in the one-dimensional case, stability alternates along the bifurcation curve, with thick lines de-

noting stable branches and thin lines indicating unstable ones. The results reaffirm the PINN framework’s ability to accurately resolve both the bifurcation structure and the associated stability transitions, even in more complex two-dimensional systems.

5.3. Three-dimensional Discrete Allen–Cahn Equation

We now extend our study to higher-dimensional versions of the discrete Allen–Cahn equation, beginning with the three-dimensional case. Due to the computational cost of constructing complete bifurcation diagrams in higher dimensions, we restrict our analysis to a single parameter value in the site-centered case.

We consider the three-dimensional problem at a fixed parameter $\mu = -0.5$ with $c = 0.05$. Choosing $m = 8$ results in a lattice with 15^3 points. Excluding the boundary points, the system consists of $13^3 = 2197$ nonlinear equations and variables. Directly solving the system requires computing a Jacobian matrix of size 2197×2197 .

To approximate the solution using PINNs, we employ a neural network with three inputs, two hidden layers each containing 10 neurons, and one output, resulting in 161 trainable parameters including biases. The corresponding Jacobian matrix has dimensions 2197×161 , significantly smaller than in the direct method. Figure 10a shows a one-dimensional slice of the solution, $u_{i,8,8}$, for visualization. The training convergence, shown in Figure 10b, confirms that the PINN accurately solves the three-dimensional problem, achieving an MSE on the order of 10^{-13} .

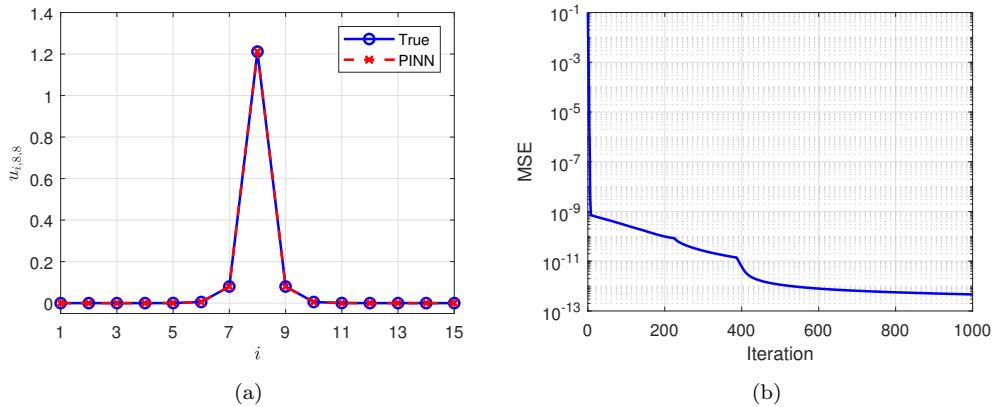


Figure 10: (a) True and PINN solutions for the three-dimensional discrete Allen–Cahn equation, plotted as $u_{i,8,8}$ for $i = 1, \dots, 15$. (b) Mean squared error (MSE) during training.

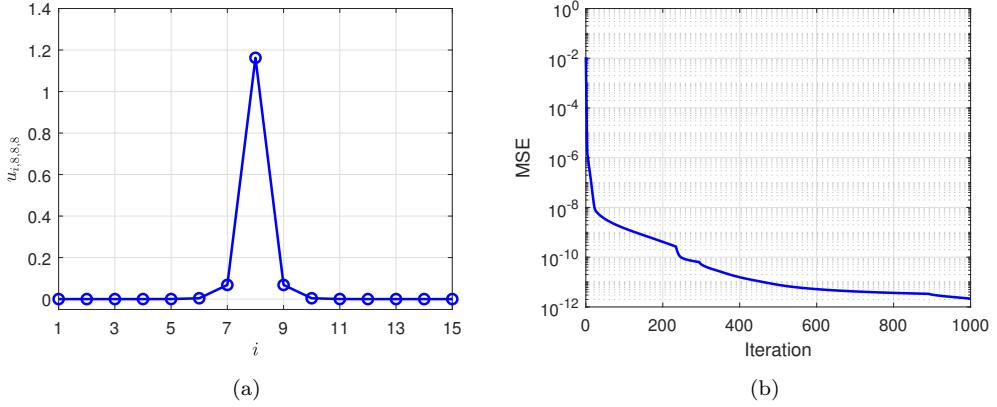


Figure 11: (a) PINN solution for the four-dimensional Allen–Cahn equation shown as $u_{i,8,8,8}$. (b) MSE during training.

5.4. Four-dimensional Discrete Allen–Cahn Equation

We next consider the four-dimensional case at fixed parameter $\mu = -0.5$ and $c = 0.05$. Choosing $m = 8$ yields a lattice with 15^4 points and $13^4 = 28561$ internal equations and variables. Solving the system directly would require manipulating a 28561×28561 Jacobian, which exceeds typical memory capacities.

Using a PINN with four inputs, two hidden layers of 10 neurons each, and one output, the number of weights is reduced to 171. The resulting Jacobian matrix has dimensions 28561×171 , a significant reduction that makes the computation feasible. Figure 11a presents a one-dimensional slice of the solution, while Figure 11b shows that the MSE reaches the order of 10^{-12} , confirming the high accuracy of the PINN approach in four dimensions.

5.5. Five-dimensional Discrete Allen–Cahn Equation

Finally, we address the five-dimensional case using the same fixed parameters: $\mu = -0.5$ and $c = 0.05$. With $m = 8$, the site-centered lattice includes 15^5 points and $13^5 = 371293$ internal variables and equations.

We use a PINN with five inputs, two hidden layers of 10 neurons each, and one output, resulting in 181 trainable parameters. The Jacobian matrix would nominally be 371293×181 , but due to its size, we adopt the stochastic approach described in Section 4.4. In each iteration, we randomly sample 1000 equations and always include the central function at $\mathbf{i} = (8, 8, 8, 8, 8)$ to increase accuracy. This reduces the Jacobian size to 1001×181 , and the Levenberg–Marquardt update involves matrices of size 181×181 and 181×1 , substantially improving computational efficiency.

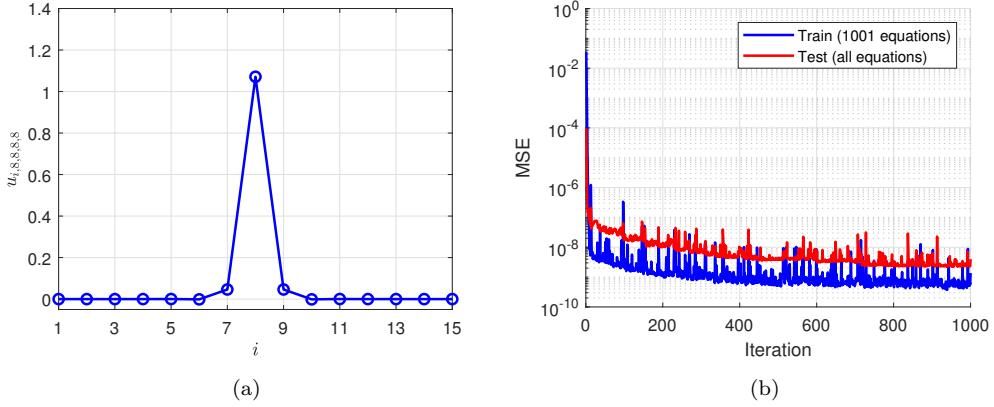


Figure 12: (a) PINN solution for the five-dimensional Allen–Cahn equation shown as $u_{i,8,8,8,8}$. (b) MSE during training: blue for training set (1001 equations), red for full system (371293 equations).

Table 1: Effect of training set size $s(S_k)$ on test MSE (full system) and computation time for the five-dimensional case over 1000 iterations.

$s(S_k)$	n_{iter}	Test MSE	Time (s)
101	1000	4.1×10^{-8}	88
1001	1000	4.0×10^{-9}	108
10001	1000	4.4×10^{-10}	301

Figure 12a shows the PINN solution slice, and Figure 12b presents the MSE for both training and test sets. Despite the stochastic updates, the final test MSE is approximately 10^{-9} , indicating that the PINN yields a sufficiently accurate solution. The non-monotonic MSE behavior arises from the stochastic nature of training, unlike the smooth convergence seen in lower dimensions where full Jacobians are used. Furthermore, the effect of varying the number of equations used during training is reported in Table 1.

Tables 2 and 3 summarize the architectural, numerical, and performance aspects of the PINNs across dimensions. The results indicate that the proposed framework scales well with dimensionality and provides accurate solutions, even in five-dimensional systems where traditional methods are computationally prohibitive.

6. Conclusion

In this work, we proposed the use of physics-informed neural networks (PINNs) to address nonlinear systems arising from discrete nonlinear lattices. The scope of the study encompassed three core tasks: obtaining steady-state

Table 2: Summary of PINN architectures from one to five dimensions.

d	PINN	n_W	$s(J)$	$s(J^T J + \lambda I)$	n_{iter}	MSE	Time (s)
1	(1,4,4,1)	33	17×33	33×33	1000	2.1×10^{-32}	0.1
2	(2,7,7,1)	85	169×85	85×85	1000	1.0×10^{-13}	10
3	(3,10,10,1)	161	2197×161	161×161	1000	4.6×10^{-13}	33
4	(4,10,10,1)	171	28561×171	171×171	1000	2.1×10^{-12}	552
5	(5,10,10,1)	181	1001×181	181×181	1000	4.0×10^{-9}	108

Table 3: Impact of hidden layer size on MSE and training time for the three-dimensional case.

d	PINN	n_W	$s(J)$	$s(J^T J + \lambda I)$	n_{iter}	MSE	Time (s)
3	(3,5,5,1)	56	2197×56	56×56	1000	1.1×10^{-12}	11
3	(3,10,5,1)	101	2197×101	101×101	1000	1.1×10^{-12}	22
3	(3,10,10,1)	161	2197×161	161×161	1000	4.6×10^{-13}	33
3	(3,15,10,1)	231	2197×231	231×231	1000	8.5×10^{-13}	53
3	(3,15,15,1)	316	2197×316	316×316	1000	2.2×10^{-13}	81
3	(3,20,15,1)	411	2197×411	411×411	1000	9.0×10^{-13}	128
3	(3,20,20,1)	521	2197×521	521×521	1000	1.9×10^{-12}	210

solutions, constructing bifurcation diagrams, and performing linear stability analysis. To that end, we formulated a PINN-based framework that processes lattice-based systems as input and solves the resulting nonlinear equations using the Levenberg–Marquardt algorithm. For high-dimensional settings, where conventional methods struggle due to memory constraints and computational cost, we introduced a stochastic optimization strategy to improve scalability and efficiency.

In the one- and two-dimensional cases, we demonstrated the efficacy of PINNs in capturing the snaking bifurcation structure and computing the associated linear stability. The continuation method was implemented via pseudo-arc length continuation, which was adapted to the PINN framework through the inclusion of an auxiliary constraint. This allowed for a sequence of neural network weights to be computed iteratively, thereby enabling the construction of full bifurcation diagrams. Depending on the nature of the solution curve, the method was flexibly applied in two variants—either assuming or not assuming a bijective relation between the norm and the solution.

For stability analysis, we proposed a PINN formulation to compute the principal eigenvalue of the linearized system. By enforcing positivity in the network output, we leveraged Sturm–Liouville theory to isolate the largest eigenvalue associated with a non-sign-changing eigenvector. Numerical re-

sults showed excellent agreement between the PINN predictions and reference solutions, affirming the robustness and accuracy of the approach.

In three-, four-, and five-dimensional settings, we extended the framework by employing compact neural networks with only modest growth in the number of trainable parameters, despite the exponential increase in system size. Each additional spatial dimension was accommodated by augmenting the input layer, while the number of hidden neurons remained fixed. This design choice helped preserve computational efficiency and kept the size of the Jacobian matrix tractable. Our experiments confirmed that even in high-dimensional spaces, PINNs can produce accurate solutions with considerably lower memory usage than traditional methods. In the five-dimensional case, we further applied a stochastic Levenberg–Marquardt algorithm, evaluating only a subset of equations per iteration, which significantly accelerated training and reduced computational overhead.

Looking ahead, several promising avenues can be explored to extend this work. One direction involves applying the proposed PINN methodology to other nonlinear lattice models that exhibit complex bifurcation phenomena. Notable examples include the discrete Swift–Hohenberg equation [19, 41] and the discrete nonlinear Schrödinger equation, which models quantum droplets and bubbles [42]. Similarly, adapting the framework to non-standard lattice geometries, such as Lieb [43], honeycomb, and triangular lattices [44], offers further opportunities for exploration.

Beyond lattice-based models, the techniques developed here may be generalized to a broader class of nonlinear systems where high dimensionality poses a challenge. The inherent flexibility of neural network architectures makes them particularly well-suited for capturing solution manifolds in such settings. Furthermore, in problems where analytical solutions are known, as in the integrable Ablowitz–Ladik lattice [10], PINNs may serve not only as numerical solvers but also as tools for symbolic discovery. Recent studies have demonstrated that neural networks can be trained to approximate and even recover closed-form solutions [45, 46, 47, 48].

In summary, this work highlights the potential of PINNs as an effective and scalable tool for solving nonlinear systems on lattices, particularly in high-dimensional and bifurcation-sensitive regimes. The combination of accuracy, adaptability, and computational efficiency makes them a compelling alternative to traditional numerical methods, especially for problems where conventional solvers become impractical.

CRediT authorship contribution statement

Muhammad Luthfi Shahab: Conceptualization, Formal Analysis, Software, Visualization, Writing - original draft, **Fidya Almira Suheri:** Conceptualization, Software, **Rudy Kusdiantara:** Conceptualization, Writing - review & editing, **Hadi Susanto:** Conceptualization, Supervision, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used Grammarly and ChatGPT in order to improve language and readability. After using these tools/services, the authors reviewed and edited the content as needed and took full responsibility for the content of the publication.

Acknowledgement

MLS is supported by a four-year Doctoral Research and Teaching Scholarship (DRTS) from Khalifa University. RK acknowledges Riset Utama PPMI FMIPA 2024 (617I/IT1.C02/KU/2024). HS acknowledged support by Khalifa University through a Competitive Internal Research Awards Grant (No. 8474000413/CIRA-2021-065) and Research & Innovation Grants (No. 8474000617/RIG-S-2023-031 and No. 8474000789/RIG-S-2024-070).

References

- [1] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics-informed neural networks: Where we are and what's next, *Journal of Scientific Computing* 92 (3) (2022) 88.
- [2] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* 2 (5) (1989) 359–366.

- [3] M. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, *Communications in Numerical Methods in Engineering* 10 (3) (1994) 195–201.
- [4] I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE Transactions on Neural Networks* 9 (5) (1998) 987–1000.
- [5] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *Journal of Computational Physics* 378 (2019) 686–707.
- [6] S. Cai, Z. Mao, Z. Wang, M. Yin, G. E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: A review, *Acta Mechanica Sinica* 37 (12) (2021) 1727–1738.
- [7] Y. Chen, L. Lu, G. E. Karniadakis, L. Dal Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, *Optics Express* 28 (8) (2020) 11618–11633.
- [8] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, *Proceedings of the National Academy of Sciences* 115 (34) (2018) 8505–8510.
- [9] E. R. Putri, M. L. Shahab, M. Iqbal, I. Mukhlash, A. Hakam, L. Mar-dianto, H. Susanto, A deep-genetic algorithm (deep-GA) approach for high-dimensional nonlinear parabolic partial differential equations, *Computers & Mathematics with Applications* 154 (2024) 120–127.
- [10] W. Zhu, W. Khademi, E. G. Charalampidis, P. G. Kevrekidis, Neural networks enforcing physical symmetries in nonlinear dynamical lattices: The case example of the Ablowitz-Ladik model, *Physica D: Nonlinear Phenomena* 434 (2022) 133264.
- [11] Z. Lin, Y. Chen, Pseudo grid-based physics-informed convolutional-recurrent network solving the integrable nonlinear lattice equations, *Physica D: Nonlinear Phenomena* 468 (2024) 134304.
- [12] R. Geng, J. Zu, Y. Gao, H.-K. Zhang, Separable graph Hamiltonian network: A graph deep learning model for lattice systems, *Physical Review Research* 6 (1) (2024) 013176.

- [13] J.-C. Zhou, X.-Y. Wen, M.-J. Guo, Symmetric difference data enhancement physics-informed neural network for solving discrete nonlinear lattice equations, *Communications in Theoretical Physics* (2024).
- [14] S. Saqlain, W. Zhu, E. G. Charalampidis, P. G. Kevrekidis, Discovering governing equations in discrete systems using PINNs, *Communications in Nonlinear Science and Numerical Simulation* 126 (2023) 107498.
- [15] A. Opala, S. Ghosh, T. C. Liew, M. Matuszewski, Neuromorphic computing in Ginzburg-Landau polariton-lattice systems, *Physical Review Applied* 11 (6) (2019) 064029.
- [16] J. Stokes, J. R. Moreno, E. A. Pnevmatikakis, G. Carleo, Phases of two-dimensional spinless lattice fermions with first-quantized deep neural-network quantum states, *Physical Review B* 102 (20) (2020) 205122.
- [17] Y. V. Kartashov, B. A. Malomed, L. Torner, Solitons in nonlinear lattices, *Reviews of Modern Physics* 83 (1) (2011) 247–305.
- [18] C. Taylor, J. H. Dawes, Snaking and isolas of localised states in bistable discrete lattices, *Physics Letters A* 375 (1) (2010) 14–22.
- [19] R. Kusdiantara, H. Susanto, Homoclinic snaking in the discrete Swift-Hohenberg equation, *Physical Review E* 96 (6) (2017) 062214.
- [20] M. L. Shahab, H. Susanto, Neural networks for bifurcation and linear stability analysis of steady states in partial differential equations, *Applied Mathematics and Computation* 483 (2024) 128985.
- [21] M. L. Shahab, H. Susanto, Corrigendum to "Neural networks for bifurcation and linear stability analysis of steady states in partial differential equations" [Appl. Math. Comput. 483 (2024) 128985], *Applied Mathematics and Computation* 495 (2025) 129319.
- [22] G. Fabiani, F. Calabrò, L. Russo, C. Siettos, Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines, *Journal of Scientific Computing* 89 (2) (2021) 1–35.
- [23] E. Galaris, G. Fabiani, I. Gallos, I. Kevrekidis, C. Siettos, Numerical bifurcation analysis of PDEs from Lattice Boltzmann model simulations: a parsimonious machine learning approach, *Journal of Scientific Computing* 92 (2) (2022) 34.

- [24] D. D. Joseph, T. S. Lundgren, Quasilinear Dirichlet problems driven by positive sources, *Archive for Rational Mechanics and Analysis* 49 (1973) 241–269.
- [25] E. J. Allen, J. A. Burns, D. S. Gilliam, Numerical approximations of the dynamical system generated by Burgers' equation with Neumann-Dirichlet boundary conditions, *ESAIM: Mathematical Modelling and Numerical Analysis* 47 (5) (2013) 1465–1492.
- [26] J. R. Graef, C. Qian, B. Yang, A three point boundary value problem for nonlinear fourth order differential equations, *Journal of Mathematical Analysis and Applications* 287 (1) (2003) 217–233.
- [27] S. Liao, Homotopy analysis method in non-linear differential equations (2012).
- [28] E. K. Chong, S. H. Zak, *An Introduction to Optimization*, Vol. 75, John Wiley & Sons, 2013.
- [29] K. Levenberg, A method for the solution of certain non-linear problems in least squares, *Quarterly of Applied Mathematics* 2 (2) (1944) 164–168.
- [30] D. W. Marquardt, An algorithm for least-squares estimation of non-linear parameters, *Journal of the Society for Industrial and Applied Mathematics* 11 (2) (1963) 431–441.
- [31] D. C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, *Mathematical Programming* 45 (1) (1989) 503–528.
- [32] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
- [33] B. Moseley, A. Markham, T. Nissen-Meyer, Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, *Advances in Computational Mathematics* 49 (4) (2023) 62.
- [34] A. D. Jagtap, G. E. Karniadakis, Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, *Communications in Computational Physics* 28 (5) (2020).

- [35] A. Heinlein, A. Klawonn, M. Lanser, J. Weber, Combining machine learning and domain decomposition methods for the solution of partial differential equations-a review, *GAMM-Mitteilungen* 44 (1) (2021) e202100001.
- [36] M. L. Shahab, H. Susanto, H. Hatzikirou, A finite difference method with symmetry properties for the high-dimensional Bratu equation, *Applied Mathematics and Computation* 489 (2025) 129136.
- [37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, *The Journal of Machine Learning Research* 12 (2011) 2825–2830.
- [38] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., TensorFlow: a system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.
- [39] R. H. Byrd, S. L. Hansen, J. Nocedal, Y. Singer, A stochastic quasi-Newton method for large-scale optimization, *SIAM Journal on Optimization* 26 (2) (2016) 1008–1031.
- [40] D. Kovalev, K. Mishchenko, P. Richtárik, Stochastic Newton and cubic Newton methods with simple local linear-quadratic rates, arXiv preprint arXiv:1912.01597 (2019).
- [41] J. Burke, E. Knobloch, Localized states in the generalized Swift-Hohenberg equation, *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 73 (5) (2006) 056211.
- [42] R. Kusdiantara, H. Susanto, F. Adriano, N. Karjanto, Analysis of multistability in discrete quantum droplets and bubbles, *Chaos, Solitons & Fractals* 187 (2024) 115410.
- [43] R. Kusdiantara, F. T. Akbar, N. Nuraini, B. E. Gunara, H. Susanto, Snakes on lieb lattice, *Journal of Nonlinear Science* 32 (4) (2022) 59.
- [44] R. Kusdiantara, H. Susanto, Snakes in square, honeycomb and triangular lattices, *Nonlinearity* 32 (12) (2019) 5170.
- [45] H. D. Mazraeh, K. Parand, Approximate symbolic solutions to differential equations using a novel combination of Monte Carlo tree search and physics-informed neural networks approach, *Engineering with Computers* (2025) 1–29.

- [46] H. D. Mazraeh, K. Parand, An innovative combination of deep Q-networks and context-free grammars for symbolic solutions to differential equations, *Engineering Applications of Artificial Intelligence* 142 (2025) 109733.
- [47] H. D. Mazraeh, K. Parand, GEPINN: An innovative hybrid method for a symbolic solution to the Lane–Emden type equation based on grammatical evolution and physics-informed neural networks, *Astronomy and Computing* 48 (2024) 100846.
- [48] H. D. Mazraeh, K. Parand, A three-stage framework combining neural networks and Monte Carlo tree search for approximating analytical solutions to the Thomas–Fermi equation, *Journal of Computational Science* 87 (2025) 102582.

Appendix A. Effect of Varying α and γ

The parameter α governs the relative weighting of the continuation equation in Eq. (9) with respect to the original nonlinear system $F(u, \mu) = 0$. When α is too small, the continuation constraint is weak and may be violated; when too large, it compromises the accuracy of the solution to the original system. Figure A.13a illustrates the effect of varying α across several orders of magnitude, from 10^{-5} to 10^5 . The results correspond to the site-centered case with $c = 0.05$ in the one-dimensional Allen-Cahn model. Here, the RMSE of the nonlinear system (blue line) and the scaled error in the continuation equation, $|\text{continuation}/\alpha|$, are plotted. The value $\alpha = 10$ achieves a desirable trade-off between these competing errors.

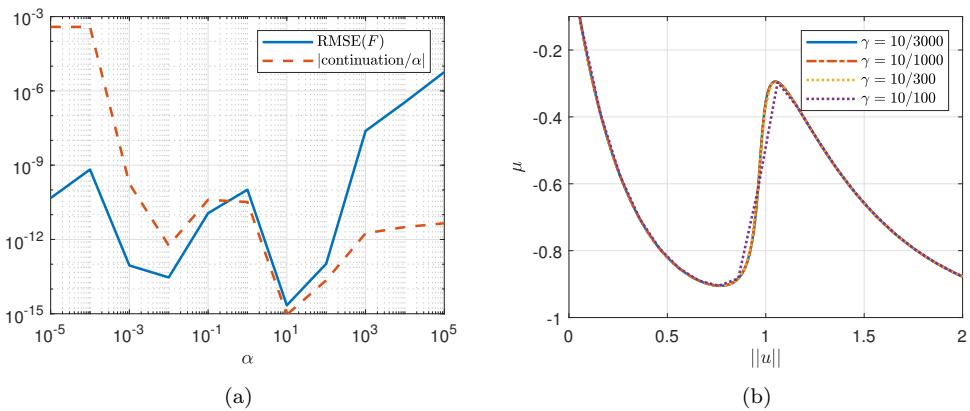


Figure A.13: (a) Influence of α and (b) γ on the one-dimensional Allen-Cahn equation.

The parameter γ defines the step size in the norm of the solution between successive continuation steps. For example, setting $\gamma = 10/1000$ implies roughly 1000 steps to reach $\|u\| = 10$. A small γ improves resolution near sharp turning points but increases computational cost, while a large γ may compromise accuracy. Figure A.13b demonstrates the consequences of varying γ for the one-dimensional Allen-Cahn equation, where bifurcation diagrams are computed up to $\|u\| = 2$. The diagram for $\gamma = 10/100$ fails to resolve the second turning point accurately, and even $\gamma = 10/300$ exhibits reduced smoothness. In contrast, both $\gamma = 10/1000$ and $\gamma = 10/3000$ provide smooth and consistent results. Computational times (in seconds) for each setting are 4.3250, 4.4447, 1.1761, and 9.0523, respectively, with $\gamma = 10/1000$ offering the most efficient balance between accuracy and speed.

Appendix B. Effect of Varying β_1 and β_2

The parameters β_1 and β_2 determine the relative step sizes in the norm of the solution and the parameter μ , respectively. In the two-dimensional case, we use $\beta_1 = 1000/40$ and $\beta_2 = 100/1$, which correspond to approximately 1000 steps to reach $\|u\| = 40$ and 100 steps to traverse one unit in μ . Figure B.14 shows the impact of varying these parameters for the site-centered case with $c = 0.05$. Smaller values of β_1 or β_2 reduce the number of continuation steps but may fail to resolve sharp turning points accurately. This is evident in the figure, where insufficient resolution leads to distorted or incomplete bifurcation structures. Conversely, increasing β_1 or β_2 improves accuracy at the cost of additional computation. It is also worth noting that the parameter δ in Eq. (19) functions in tandem with β_1 and β_2 to control the arclength step. In practice, varying δ independently is unnecessary, as its effect is implicitly governed by appropriate choices of β_1 and β_2 .

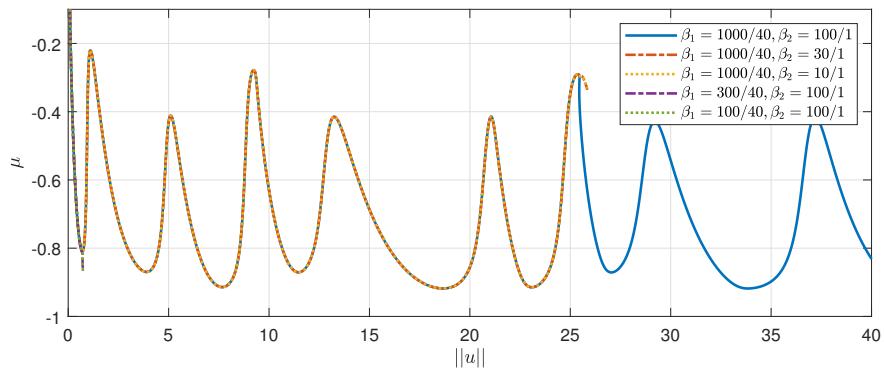


Figure B.14: Effect of β_1 and β_2 on the two-dimensional Allen-Cahn equation.