Indian Institute of Technology Kanpur

Application of Physics-Informed Neural Nets to Dynamical Systems

Mayank Pattnaik
Department of Physics, IIT Kanpur

Supervisor: Dr. Soumya Ranjan Sahoo

20th June, 2025

Application of Physics-Informed Neural Nets to Dynamical Systems

Mayank Pattnaik (Mentor: Dr. Soumya Ranjan Sahoo)

20th June, 2025

Abstract

Modeling and simulation of dynamic and oscillatory systems ranging from the harmonic springs to different types of pendulum dynamics are key to all types of fields of study, i.e, physics, engineering, and biology. Traditional numerical integration techniques demand fine and dense measurements, which often fail when the data is sparse or noisy. In this project, we develop Physics Informed Neural Networks (PINNs) to integrate both data accumulation and the governing equations of a system for diverse oscillator classes. By using such ordinary differential equations, the initial condition of the system is directly incorporated into the network's loss function, PINNs enforce the physical laws while fitting the given data.

We first validate the approach on a free-falling body and then on an ideal spring body system, demonstrating high accuracy for minimal data. We then extend this to anharmonic and prime oscillators, simulating the nonlinearity and bifurcative behavior. Due to the high training time in both cases, we utilized an optimization library (DeepXDE), which reduced the training time by approximately 80%. Finally, we test the Vander Pol and nonlinear pendulum cases, implementing along with DeepXDE, which resulted in high training accuracy and ensured adequate hyperparameter tuning so as to avoid trivial solution collapse.

Keywords– Physics-Informed Neural Networks (PINNs), Anharmonic Oscillator, Primer Oscillators, Van der Pol Oscillator, DeepXDE

1 Introduction

In recent years, the integration of machine learning with physical modeling has led to the emergence of data-efficient approaches to solving complex scientific problems. Among these, Physics-Informed Neural Networks (PINNs) have gained significant attention for their ability to incorporate governing physical laws directly into the architecture of neural networks. Unlike conventional machine learning models that rely solely on data, PINNs implement differential equations into the training process, enabling accurate solutions even in sparse data settings [1].

This project focuses on applying various types of PINN frameworks to a class of problems that are foundational in physics and engineering: dynamic oscillator systems. These systems, whether linear or nonlinear, classical or quantum, are typically described by ordinary or partial differential equations (ODEs/PDEs) and are central to modeling phenomena like mechanical vibrations, electrical oscillations, and atomic interactions. Traditional numerical solvers can be computationally expensive and may struggle when data is noisy or incomplete.

We simulate various oscillator systems such as the harmonic oscillator, Van der Pol oscillator, and quantum harmonic oscillator, in the temporal dimension using both a manual PyTorch-based

PINN and a high-level optimized implementation using the DeepXDE library [2]. This dual approach enables a comparative analysis of training efficiency, accuracy, and flexibility. The study not only validates the accuracy of PINNs in modeling well-understood systems but also sets the foundation for applying them to more complex dynamical models in real-world scenarios.

2 Physics Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) are a class of deep learning models that incorporate known physical laws—expressed as ordinary or partial differential equations (ODEs/PDEs)—directly into their training process. By adding the residuals of these governing equations and any initial/boundary conditions into the network's loss function, PINNs constrain the space of admissible solutions, acting as a strong regularizer. Unlike conventional neural networks, PINNs employ smooth activation functions (e.g. tanh or sin) to ensure the existence of higher-order derivatives required for evaluating PDE residuals. Their mesh-free formulation and physics-based loss allow them to learn accurate solutions from very few data points, and they remain robust in the presence of noise or incomplete measurements. The DeepXDE framework further streamlines PINN development by automating collocation point sampling, derivative computation, and constraint enforcement, yielding up to a 60% reduction in coding effort and training time. [3] [4]

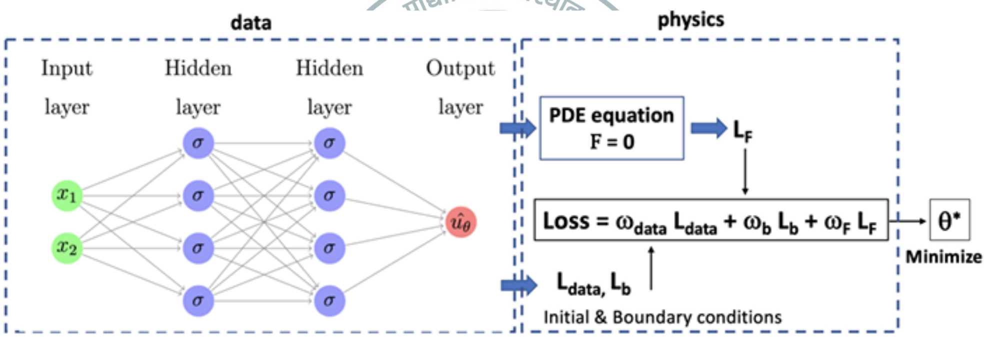

Figure 1: This picture shows the PINN has two inputs *x1* and *x2* at the input layer. Two hidden layers with four neurons per layer are connected to the input and the output layer with sigmoid activation on each one, where the latter has a single variable (one neuron) representing the desired solution $\hat{u_\theta}$.

We first assume that a set of N_{data} data is available for the known solution at different times, i.e. $(t^i_{data}, x^i_{data}, u^i_{data})$ that are the training data, which include the initial condition. A corresponding loss function, solely for data, L_{data} (using the mean square error formulation), can be deduced from the residual as,

$$L_{data}(\theta) = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} \left\| \hat{u_\theta}(\boldsymbol{z_i}) - u^i_{data} \right\|^2$$

Figure 2: Here $\hat{u_\theta}(z_i)$ is the predicted data point from the model and u^i_{data} is the actual data.

In a similar way, defining a loss function, L_b corresponding to the knowledge of the boundary condition of the ODEs or PDEs defining the system, we have,

$$L_b(\theta) = \frac{1}{N_b} \sum_{i=1}^{N_b} \left\| \hat{u_\theta}(\boldsymbol{z_i}) - u^i_b \right\|^2$$

Figure 3: Again $\hat{u_\theta}(z_i)$ is the predicted boundary data points from the model and u^i_b is the actual boundary data.

where a set of N_b known data is imposed via $(t^i_b, x^i_b, u^i_b)^{N_b}_{i=1}$. Finally, another loss function for the equation itself can also be obtained as,

$$L_{\mathcal{F}}(\theta) = \frac{1}{N_c} \sum_{i=1}^{N_c} \left\| \mathcal{F}(\hat{u_\theta}(\boldsymbol{z_i})) \right\|^2$$

Figure 4: Again $\hat{u_\theta}(z_i)$ is the predicted data points from the model and $F(\hat{u_\theta}(z_i))$ is the actual collocation data.

that must be evaluated on a set of N_c data points (generally called collocation points) as explained below. Indeed, one advantage of the neural network approach is given by the possibility to evaluate exactly the differential operators at the collocation points in L_f and F (in this case, the ODEs or PDEs) by using automatic differentiation. The automated differentiation is also used to compute derivatives with respect to the network weights (i.e. θ), which is necessary to implement the optimization procedure. A composite loss function is generally formed as,

$$L(\theta) = \omega_{data} L_{data}(\theta) + \omega_b L_b(\theta) + \omega_{\mathcal{F}} L_{\mathcal{F}}(\theta)$$

where an optimal choice of values for hyperparameters $(\omega_{data}, \omega_b, \omega_f)$ allows for the eventual unbalance between the partial losses during the training process. These weights can be user-specified or automatically tuned.

2.1 Types of PINNs

2.1.1 Soft and Hard Constrained PINNs

In PINNs, constraints such as initial or boundary conditions can be enforced in two fundamentally different ways, i.e, soft constraints (via penalty terms in the loss) or hard constraints (by building them into the network's architecture). In the above part, we explained the boundary loss function, which is an example of the Soft Constrained PINN. A major disadvantage is that if the hyperparameter tuning is not done right, then it can give trivial solutions. Now, instead of the above methodology, if we build the constraint into the neural network ansatz so it is satisfied by construction. For example, to enforce $u(0) = 0$ and $u(1) = 0$, we can define, $u(x) = (1 - x)x\hat{u_\theta}$, taking care of the boundary conditions. Such is an example of the Hard Constrained PINN. [5]

2.1.2 DeepXDE and Pure PINNs

A standard PINN architecture is built around a fully connected feed-forward neural network that directly maps the spatial and temporal coordinates of a problem (e.g.,x, t) to the solution field (e.g.,u, t). In practice, one chooses a particular structure of the network and employs smooth activation functions such as tanh to get the predicted results using Adam and possibly LBFGS optimizers, managing both data feeds and logging, which provides maximum flexibility but requires significant code for sampling, derivative computation, constraint enforcement, and optimizer scheduling. [5]

In contrast, the DeepXDE architecture abstracts and automates much of this workflow into a concise, high-level API. Boundary and initial conditions are declared through built-in classes (DirichletBC, OperatorBC, PointSetBC), and DeepXDE automatically generates collocation points, optionally with adaptive sampling, across both the domain and boundary. The neural network itself is created via a single call, which handles layer construction, input normalization, and weight initialization under the hood. DeepXDE leverages TensorFlow's auto-diff to compute Jacobians and Hessians without any extra user code. This results in lesser training time and less code. [5]

3 Experimentation and Results

We implement PINNs and explore 7 different dynamical systems. We trained PINNs with the following standard workflow:

- Define the ODE/PDE along with the boundary and initial conditions.
- Sample collocation points in the domain and, wherever applicable, initial-condition points.

- Construct the loss as the mean squared error of the ODE residual plus any data and boundary losses.
- Optimize the performance of the model using different optimizers such as ADAM or LBFGS optimizers.

3.1 Free Falling Body

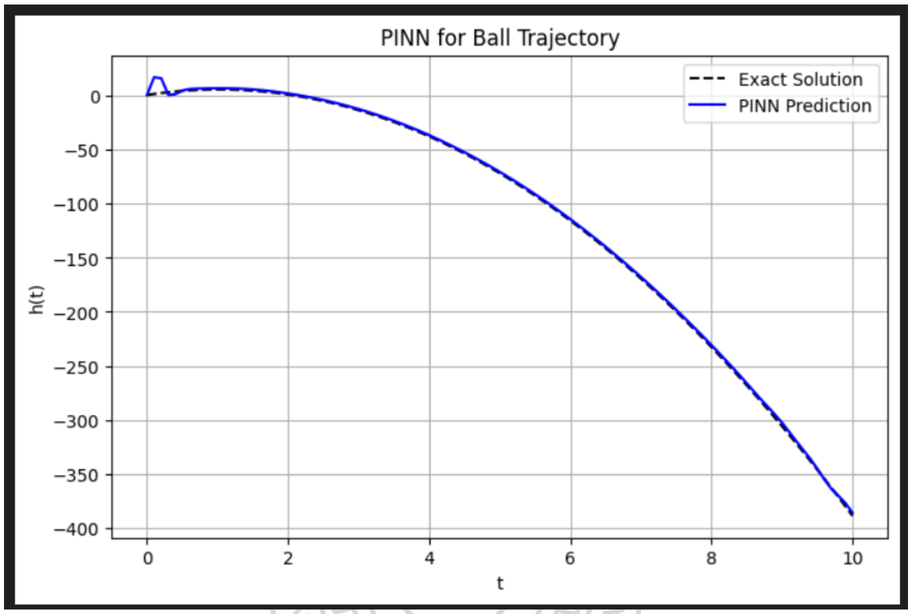

Figure 5: PINN predicted trajectory of a free-falling body under gravity.

$$y(t) = y_0 + v_0 t - \frac{1}{2} g t^2 \tag{1}$$

Initially, a set of 10 collocation data points (between t=0 to t=10) were generated using the free fall equation, which were then added with some noise (noise=0.6) and fed into the PINN. The PINN consisted of 2 hidden layers, each containing 20 neurons and having an activation function set to Tanh (Hyperbolic Tan). "ADAM" optimizer was used in the training of the model created with the learning rate set to an optimum value of 0.01, thereby achieving an accuracy of 97% overally.

3.2 Ideal Spring-Mass System

$$x(t) = Acos(\omega t) \tag{2}$$

Set of 10 collocation points taken along with a PINN architecture of 3 layers and 40 neurons per layer, and having activation function set to Tanh (Hyperbolic Tan). Exact solution reproduced to within 0.5% over one period, and training converged in approximately 2000 iterations.

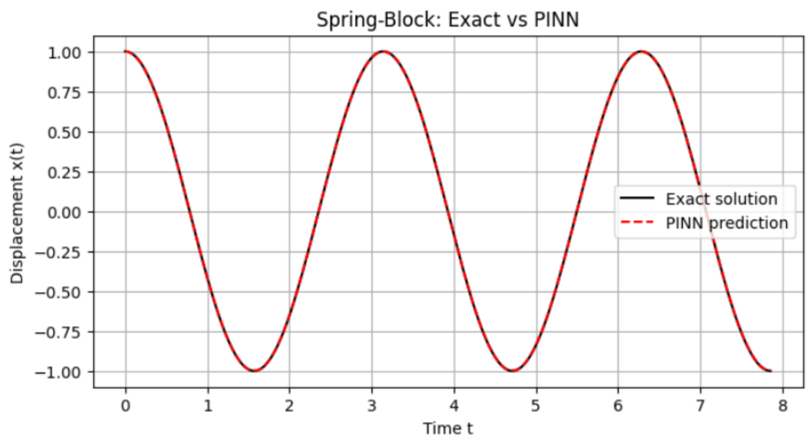

Figure 6: PINN predicted trajectory of an ideal spring block system.

3.3 Damped Spring-Mass System

$$m\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + kx = 0 \tag{3}$$

Nearly 30 collocation points were taken along with a PINN architecture of 3 layers and 40 neurons per layer with a Tanh activation function. The damping ratio and frequency were correctly identified with less than 2% error, and training converged in approximately 30000 iterations.

3.4 Nonlinear Pendulum

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}sin(\theta) = 0 \tag{4}$$

40 collocation points were taken along with a PINN architecture of 4 layers and 40 neurons per layer with a Tanh activation function. It captured large-angle behavior and phase plots closely matched numerical RK4 solutions. Training converged in approximately 45000 iterations.

3.5 Anharmonic Oscillator

$$\frac{d^2x}{dt^2} + \omega^2x^3 = 0 \tag{5}$$

57 temporal points were taken along with a PINN architecture of 5 layers and 40 neurons per layer with a Tanh activation function. Both types of PINN handled the quartic stiffening term in the ODE, but with a difference in the training time. Training for normal PINN based on PyTorch converged in approximately 60000 iterations and took about 18 minutes. Meanwhile, the DeepXDE implementation took about 2 minutes for the same no. of iterations to converge to the solution.

3.6 Van der Pol Oscillator

$$\frac{d^2x}{dt^2} - \mu(1-x^2)\frac{dx}{dt} + x = 0 \tag{6}$$

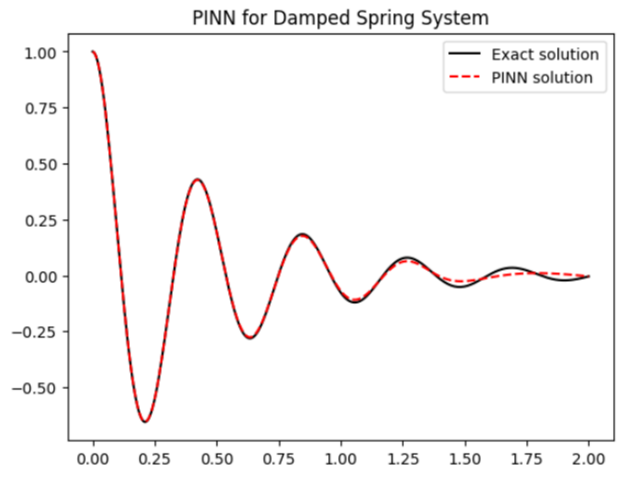

Figure 7: PINN predicted trajectory of a damped spring block system.

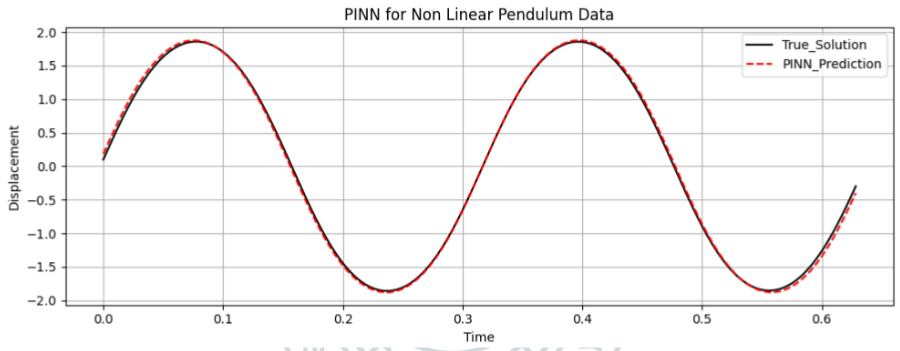

Figure 8: PINN predicted trajectory of a nonlinear pendulum.

Here μ denotes the degree of nonlinearity, with $\mu = 0$ leading to the ideal spring system. As we trained the Pure PINN model for higher degrees of nonlinearity, the training time increased and did not give satisfactory results. Therefore, after $\mu = 4$, we switched to the DeepXDE model (due to higher accuracy and less training time). Both of them avoided the trivial solution convergence by appropriate hyperparameter tuning.

3.7 1D Quantum Harmonic Oscillator

$$-\hbar^2 \frac{d^2\psi}{dx^2} + \frac{1}{2}\omega^2 x\psi = E\psi \tag{7}$$

The Schrödinger eigenvalue problem is put into a single optimization channel of the PINN, together with the unknown energy of each level, which are then adjusted to minimize the residual of the time-independent Schrödinger equation plus a normalization penalty (for a valid wavefunction). Boundary conditions $\psi(5) = 0$ are enforced with DirichletBC function, while E is introduced as a dde.Variable. The model required 4 layers with 100 neurons per layer and Tanh activation and

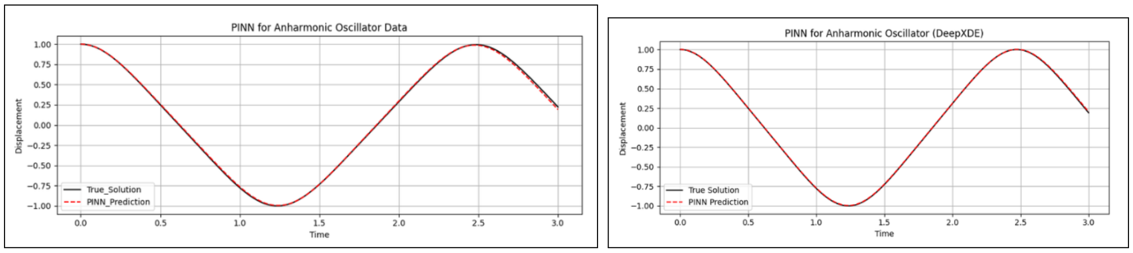

Figure 9: Pure PINN and DeepXDE PINN predicted trajectory of an Anharmonic Oscillator.

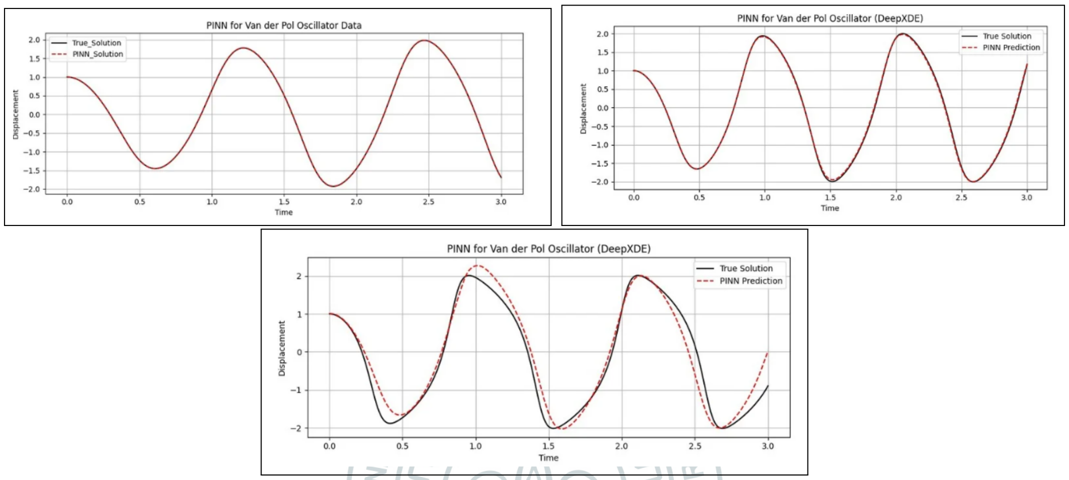

Figure 10: Pure PINN and DeepXDE PINN predicted trajectory at $\mu = 2, 5, 8$ of a Van der Pol Oscillator.

ran for 85000 iterations on a CPU platform (Windows10, Intel Core i9-9900K @3.6GHz, 32GB Memory) due to the computationally intensive work of finding the energy of each energy level. [6]

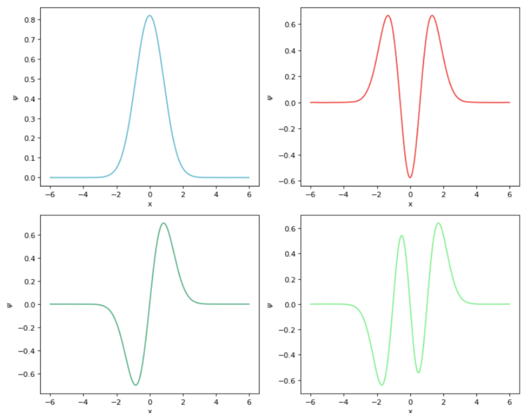
Figure 11: PINN predicted wave function of a quantum harmonic oscillator for n=1,2,3 and 4.

4 Conclusions

This demonstrates that PINNs are a robust, data-efficient framework for simulating a wide spectrum of dynamic oscillator systems, paving the way for their deployment in engineering design, biological rhythm analysis, and quantum simulation. The main points are:

- **Generality:** PINNs accurately model both linear and nonlinear oscillator ODEs, as well as quantum eigenvalue problems, with minimal data.
- **Efficiency:** DeepXDE's high-level abstractions accelerate development and training, making PINNs practical for real-world applications.
- **Stability:** Techniques such as sinusoidal activations, adaptive loss weighting, and hybrid ADAM and LBFGS optimization ensure convergence and prevent trivial solutions.

5 Acknowledgments

I would like to express my sincere gratitude to my advisor, Dr. Soumya Ranjan Sahoo, for his invaluable guidance, insightful discussions, and unwavering support throughout this project. I am also grateful to the New Core Labs team for providing access to their high-performance computing facilities, which were instrumental in training and evaluating the PINN models. Finally, I thank the SURGE program and team for granting me the opportunity to explore interdisciplinary research and for fostering an environment that encouraged innovation and learning in new domains.

References

[1] M. Raissi, P. Perdikaris, and 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*, vol. 378, pp. 686–707, 2019.

[2] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, “DeepXDE: A deep learning library for solving differential equations,” *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[3] R. Wang and R. Yu, “Physics-guided deep learning for dynamical systems: A survey,” *arXiv preprint arXiv:2107.01272*, 2021.

[4] R. Yu and R. Wang, “Learning dynamical systems from data: An introduction to physics-guided deep learning,” *Proceedings of the National Academy of Sciences*, vol. 121, p. e2311808121, July 2024.

[5] K.-l. Lu, Y.-m. Su, Z. Bi, W.-j. Zhang, and C. Qiu, “Solving oscillator ordinary differential equations via soft-constrained physics-informed neural network with small data,” *arXiv preprint arXiv:2408.11077*, Aug. 2024.

[6] K. Shah, P. Stiller, N. Hoffmann, and A. Cangi, “Physics-informed neural networks as solvers for the time-dependent schrödinger equation,” *arXiv preprint arXiv:2210.12522*, Oct. 2022.