Tunable complexity benchmarks for evaluating physics-informed neural networks on coupled ordinary differential equations

Alexander New, Benjamin Eng, Andrea C. Timm, Andrew S. Gearhart
Research and Exploratory Development Department
Johns Hopkins University Applied Physics Laboratory
Laurel, Maryland 21044
Corresponding author: alex.new@jhuapl.edu

Abstract—In this work, we assess the ability of physics-informed neural networks (PINNs) to solve increasingly-complex coupled ordinary differential equations (ODEs). We focus on a pair of benchmarks: discretized partial differential equations and harmonic oscillators, each of which has a tunable parameter that controls its complexity. Even by varying network architecture and applying a state-of-the-art training method that accounts for “difficult” training regions, we show that PINNs eventually fail to produce correct solutions to these benchmarks as their complexity—the number of equations and the size of time domain—increases. We identify several reasons why this may be the case, including insufficient network capacity, poor conditioning of the ODEs, and high local curvature, as measured by the Laplacian of the PINN loss.

I. INTRODUCTION

In recent years, there has been much interest in using machine learning (ML) models to approximate systems governed by mechanistic dynamics [1]. This includes rapid forward simulations and the ability to determine unknown components of dynamics via solution of inverse problems. Typically, these physics-based ML models incorporate a priori knowledge of the system to be modeled—via a black-box simulation environment or an explicit system of differential equations (DEs). In this work, we analyze a popular approach to the latter strategy: physics-informed neural networks (PINNs) [2]. PINNs incorporate DEs as physics-based regularizers in the objective function of a neural network (i.e., soft constraints) along with terms associated with initial and boundary conditions. This is a straightforward approach to incorporating known dynamics into neural networks and showed early success [3], [4], but recent results [3], [5]–[7] highlight difficulties that PINNs face due to their complex objectives. Our analysis builds upon these latter efforts, focusing on representing coupled systems of ordinary differential equations (ODEs).

ODE systems, or differential equations with a single independent variable and one or more dependent variables, arise in a number of use cases. These include modeling reaction-diffusion processes in synthetic biology [8] and chemical kinetics [9], as well as solving partial differential equations (PDEs) via the method of lines [10]. This leads, for example, to applications in fluid mechanics like Burgers’ equation [11] and condensed matter physics such as lattice dynamics [12].

Systems of ODEs vary in complexity based on domain size, number of equations being modeled, and problem stiffness. These challenges occur regularly—for example, when discretizing PDEs into coupled ODEs, the number of ODEs scales with discretization granularity. This is a computational bottleneck in higher-fidelity simulations. Due to the ubiquity of these systems, it is natural to explore how effectively PINNs solve them, especially as their complexities increase. Similar work was performed by [5], who use underlying PDE parameters such as convection coefficients as proxies for the complexity of the learning problem, and show that PINNs fail to solve these problems as the complexity increases.

In this paper, we review the formulation of PINNs (Section II-A and Section II-D) and introduce two problems that involve systems of coupled ODEs (Section II-C). Each problem is characterized by a tunable “complexity”. We then demonstrate that PINNs, even with the use of a state-of-the-art training method, fail to produce correct solutions to the system of ODEs (as indicated by a classical ODE solver) as the complexity of the problem increases (Section III). We then link the intuitive complexity of the problem with a quantitative characterization of the difficulty via the Laplacian of the PINN learning problem.

II. APPROACH

A. Physics-Informed Neural Networks

PINNs [2] are neural networks with weights trained such that the network satisfies a differential equation. The use of a neural network to satisfy a DE is motivated by the universal approximation theorem [13]; however, this theorem does not guarantee that a specific optimization procedure will yield a network satisfying the DE. The authors of [14] showed that, for certain categories of PDE, as the amount of available data increases, the PINN learning problem will converge to a solution to the PDE.
Due to the overhead of training, solving a DE with a PINN takes longer than using a classical numerical method. Unlike classical methods, however, PINNs are easily extended to inverse problems—i.e., given a partially-specified governing equation and some data, recover the full governing equation. Furthermore, PINNs do not require a mesh, which enables efficient application to new problems. The forward prediction setting, which we consider here, is a different use-case than many ML applications, in that the training signal uses a DE (or system of DEs), rather than a labelled dataset.

To define PINNs, consider a general ODE that determines function $u : \mathbb{R} \rightarrow \mathbb{R}^N$:

$$
\mathcal{N}(u)(t) = 0, \quad t \in (0, T),
$$

$$
\mathcal{I}(u)(t) = 0, \quad t = 0,
$$

where $\mathcal{N}$ is a differential operator, and $\mathcal{I}$ captures deviation from prespecified initial conditions. For the first-order DEs considered here, $\mathcal{I}(u)(t) = u(t) - u_0$, where $u_0$ is fixed. Then a PINN $\hat{u}_w : \mathbb{R} \rightarrow \mathbb{R}^N$ is a neural network parameterized by a weight vector $w \in \mathbb{R}^M$ that satisfies

$$
\min_w L(w) = \sum_{t_d \in T} ||\mathcal{N}(\hat{u}_w(t_d))||^2 + \nu_\mathcal{I}||\mathcal{I}(\hat{u}_w)(0)||^2, \quad (1)
$$

where $\nu_\mathcal{I} > 0$ weights the initial condition components and $T = \{t_1, \ldots, t_D\} \subseteq (0, T)$ is the set of training points. The PINN literature typically refers to the loss term involving the differential operator $\mathcal{N}$ as the residual loss.

The PINN loss $L(w)$ may be minimized with first-order methods like Adam [15]. Unlike many other ML settings, it includes derivatives of $\hat{u}_w$ with respect to its input $t$. These may be calculated using the same automatic differentiation procedures [16] used to calculate loss gradients with respect to network weights, but this means that the weight update of a PINN loss function contains second- and higher-order derivatives (a gradient calculation with respect to $w$, combined with derivatives with respect to $t$ based on the DE).

In practice, we often solve a modified formulation of Equation (2), introduced by [3]:

$$
\min_w \max_\lambda L(w, \lambda) \quad (2)
$$

where

$$
L(w, \lambda) = \frac{1}{|T|} \sum_{t_d \in T} \mu(\lambda_d)||\mathcal{N}(\hat{u}_w(t_d))||^2 + \mu(\lambda_0)\nu_\mathcal{I}||\mathcal{I}(\hat{u}_w)(0)||^2,
$$

where $\lambda \in \mathbb{R}^{D+1}_{>0}$ are attention weights associated with each point $t_d \in \{0\} \cup T$, and $\mu : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is a masking function required to be differentiable, non-negative, and strictly increasing. This problem can be solved with a joint gradient ascent (for attention weights $\lambda$) and descent (for PINN weights $w$) procedure.

The authors of [5] observe that the PINN learning problem can be ill-conditioned due to the differential operator $\mathcal{N}$ in $L$. We continue this line of analysis by considering the Laplacian $\Delta L_w$ of a component $c$ (residual loss or initial condition loss) of $L$ with respect to the network weights $w$:

$$
\Delta L_w(c) = \sum_p \frac{\partial w_p}{\partial w_p} L_w(w),
$$

where $L_w$ is the residual loss or the initial condition loss in Equation (1). The Laplacian’s magnitude measures the local size of the learning problem’s curvature. We evaluate the Laplacian by estimating the trace of the Hessian $\nabla^2 L_w(w)$ using Hutchinson’s method [17]:

$$
\Delta L_w(c) = \text{tr} \nabla^2 L_w(c) = \mathbb{E}_v \{v^T[(\nabla^2 L_w)(v)]v\},
$$

where $v \in \mathbb{R}^M$ has iid components sampled from a Rademacher distribution, and the Hessian-vector products do not require the full Hessian [18].

PINNs have primarily been used to solve and analyze PDEs rather than ODEs. This is partially because, unlike for PDEs, classical ODE solvers such as Runge-Kutta [19] are efficient for forward predictions of ODE systems. We focus here on forward prediction of ODEs because they represent the simplest-possible application domain of PINNs. PINN failures during forward prediction should be addressed to allow general use of PINNs in other settings like inverse problems and PDEs.

B. The pinn-jax library

PINN research has benefited from several open-source libraries, including DeepXDE [4] and NeuralPDE.jl [20]. DeepXDE in particular easily enables solving PINNs over irregular domain geometries. For this work, we implemented our PINNs in a new library, pinn-jax, which is built on the jax\(^1\) framework and uses flax\(^2\) for neural network layers and optax\(^3\) for optimization. The use of jax improves code performance and allows calculations to be run on CPUs/GPUs/TPUs as available, while preserving the flexibility to specify different ODEs and obtain their solutions via different PINN training strategies (e.g., different weighting schemes) and neural network architectures (e.g., multi-layer perceptrons (MLPs) and ResNets [21]).

As we consider scalar inputs for our networks, pinn-jax implements first-order loss derivatives with forward-mode automatic differentiation (the jacfwd function), and second-order loss derivatives with forward-over-forward-mode automatic differentiation. The vmap function enables efficient calculation of these derivatives over an entire batch of domain points. Calculation of Hessian-vector products to assess loss Laplacians $\Delta L_w$ is accomplished by composing the Jacobian-vector product function jvp with the gradient function grad. Although we focus on PINNs for ODEs, we note that pinn-jax also includes functionality for solving PDEs. In that component of the library, we use DeepXDE’s [4] geometry module, which enables specification of and sampling from complex domains.

---

1https://github.com/google/jax
2https://github.com/google/flax
3https://github.com/deepmind/optax
C. Benchmarks

After finding it difficult to train PINNs on a set of biological reaction equations [8], we decided to explore the broader problem of using PINNs to solve systems of ODEs. This resulted in the development of two benchmarks with parameterized complexity—one that scales the number of equations in the system (a discretized heat equation) and the other that defines complexity as a maximum simulation time (simple harmonic motion (SHM)). Each of our benchmarks has additional parameters (e.g., choice of initial conditions) that we do not vary in these results—our goal is to leave much fixed and explore the effect of increasing complexity on the ability of a PINN to solve these ODE systems.

1) Discretized heat equation: The heat equation, \( u_t = \Delta u \), is defined for a scalar field \( u \) on the domain \( (x,t) \in (0,1) \times (0,T) \). In particular, when sources are added to it, one obtains a broad family of reaction-diffusion systems applicable to lattice problems [12]. Although the heat equation may be solved as a PDE with a PINN, here we explore how effectively PINNs solve it after it is discretized into a set of coupled ODEs. In particular, we apply the method of lines [10] and discretize the spatial dimension into \( N \) evenly-spaced points \( x_1, \ldots , x_N \). where \( x_n = (n-1)\Delta x = (n-1)/(N-1) \), and then approximate the second derivative with a central finite difference. This yields the following coupled ODE, for a function \( u : \mathbb{R} \rightarrow \mathbb{R}^N \):

\[
\begin{align*}
\dot{u} & = Au + f, \quad t \in (0,T), \\
u_{t=0} & = [g(x_1), \ldots , g(x_N)]^T
\end{align*}
\]

where the matrix \( A \) is given by

\[
A_{n,n} = -2(N-1)^2, \quad A_{n,n+1} = A_{n+1,n} = (N-1)^2,
\]

and \( A_{n,n'} = 0 \) for other \( n,n' \). Further, \( f = (N-1)^2[u_L(t), 0, \ldots, 0, u_R(t)]^T \), \( g \) specifies the initial condition, and \( u_L \) and \( u_R \) specify the boundary conditions on the left and right side, respectively, of the spatial domain \( (0,1) \).

An example solution to this problem is shown in Figure 5(a) in Appendix C. In this discretized formulation, the complexity of the problem is determined by the size of the discretization \( N \). The discretized differential operator \( A \) couples each state variable \( u_n \) to its spatially-adjacent variables \( u_{n+1} \) and \( u_{n-1} \).

As the number of discretized space-points \( N \) increases, the condition number \( \kappa_N \) of the discretized differential operator \( A \) increases without bound [22]. This yields an explicit connection between the intuitive complexity of the problem \( (N) \) and the complexity of the underlying optimization problem \( (\kappa_N) \). Further details on this benchmark are found in Appendix A.

2) Long time-horizon simple harmonic motion: We begin by considering a classic coupled ODE: simple harmonic motion (SHM), and we increase its complexity by evaluating it over increasingly large time domains. This ODE, for a function \( u : \mathbb{R} \rightarrow \mathbb{R}^2 \), is specified by

\[
\begin{align*}
\dot{u} & = Au, \quad t \in (0,T) \\
u_{t=0} & = u_0,
\end{align*}
\]

where the matrix \( A \) is given by \( A_{1,2} = -\omega, A_{2,1} = \omega, \) and \( A_{1,1} = A_{2,2} = 0 \), for a frequency \( \omega > 0 \). This problem has a closed-form solution in which \( u_1 \) and \( u_2 \) are sinusoidal, and a representative solution is shown in Figure 5(b) in Section C. In this work, we use a fixed initial condition of \( u_0 = [0, \pi/2]^T \) and a frequency of \( \omega = 1 \).

When being solved by a PINN, the complexity of the ODE is determined by the size of the time domain \( T \). Although the solution is periodic, we will show that PINNs fail to solve this ODE as \( T \) increases in Section III-C.

D. Related work

PINNs were introduced in [2], and there has been a great deal of follow-on research. We focus on work that considers the PINN learning problem and recommend [1] as an overview of the broader space of physics-informed ML.

The authors of [6] showed that standard PINNs can fail to solve simple PDEs like the 2D Helmholtz equation with a known analytic solution. The PINN struggled with fitting this problem’s boundary condition, which the authors relate to underlying stiffness in the dynamics of the PINN training process. They suggest several heuristic strategies for mitigating this stiffness, including a learning rate annealing process and a novel network architecture. Similarly, [23] use gradient uncertainties to adjust learning rates for PINNs.

In follow-on work, [7] analyzed the dynamics of the PINN learning problem through a neural tangent kernel (NTK) [24] perspective and found that components of the PINN loss function had different convergence rates. Most recently, [25] proposed a time-weighted scheme for training PINNs that attempts to preserve physical causality in DEs, and [26] analyzed how PINNs can fail to propagate solutions across large time-domains.

The authors of [5] focused instead on the difficulties that follow from the presence of differential operators in the PINN loss function. This operator can lead to a poorly-conditioned loss function (as measured by local smoothness), which impairs the PINN’s ability to model simple PDEs like a reaction-diffusion model. The authors of [5] also used Hessian-based information to analyze PINNs—specifically, they perturbed the PINN loss in directions defined by the top two principal eigenvectors of the Hessian. Similarly, [6] tracked the principal eigenvalue of the Hessian to define a time scale of training. Outside of work with PINNs, [17] analyze Hessian traces and spectral densities of feed-forward neural networks for computer vision problems, and [27] proposes a training scheme that uses an approximated Hessian diagonal to account for single-objective loss function curvature.

Similarly to our work, [9] noted that the PINNs defined by [2] struggle when attempting to model stiff systems of ODEs for chemical kinetics models. They showed that the
quasi-steady-state assumption, used to reduce the stiffness of a problem by assuming a zero concentration change rate for certain variables, can improve the accuracy of PINNs when the reduced system is used as the PINN loss $L(w)$. This approach showed promise for stiff systems, but will likely not mitigate the general challenge of ODE system complexity with increased problem size and coupling.

One of our benchmarks (Section II-C2) shows that the accuracy of a PINN breaks down as the size of the time domain $T$ increases. Prior work has considered similar settings for PINNs [5], [28] and operator-learning approaches [29], albeit focusing more on complicated PDEs. We show here that the challenge of using PINNs for long time-horizon forecasting exists even in the simplest ODEs.

### III. Results

#### A. Evaluation procedure

In Table I in Appendix B, we show the network configurations and hyperparameters considered when training PINNs for each benchmark. All hidden layers had the same number of units (64 or 128) depending on the particular architecture, and ResNet models incorporate skip connections [21] between hidden layers. These network sizes are typical of those used in PINN literature (e.g., [3], [23], [25], which primarily consider networks with 4-5 layers and 20-100 hidden units).

We selected training points across the time domain as $\mathcal{T} = \{T/D, 2T/D, \ldots, (D - 1)T/D, T\}$, evaluation points as the midpoints of $\mathcal{T}$: $\mathcal{T}' = \{(3T)/(2D), \ldots, (2D - 1)T/(2D)\}$, and for each benchmark/complexity value pair we trained a total of 48 PINN configurations. All networks used tanh activation functions to ensure smoothness in the loss, and Table II in Appendix B includes benchmark-specific parameters.

#### B. Evaluation metrics

We evaluate PINN solutions via several methods. The primary approach is with the relative $L^2$ error of the PINN solution $\hat{u}_w$ at the end of training vs. a solution $u$ obtained with a classical ODE solver:

$$\text{RelError}_\mathcal{T}(u, \hat{u}_w) = \sqrt{\frac{\sum_{t' \in \mathcal{T'}} \|u(t') - \hat{u}_w(t')\|^2}{\sum_{t' \in \mathcal{T'}} \|u(t')\|^2}},$$

where $\mathcal{T}' \subseteq (0, T)$ is a set of evaluation collocation points distinct from the training points $\mathcal{T}$. Classical solver solutions are obtained with the `solve_ivp` function in `scipy.integrate` using `RK45`, the explicit Runge-Kutta method of order 5(4) (i.e., fourth-order accuracy with fifth-order local extrapolation) [19]. In addition, we consider relative initial condition error

$$\text{RelError}_0(u, \hat{u}_w) = \frac{\|u_0 - \hat{u}_w(0)\|}{\|u_0\|}.$$

As discussed in Section II-A, we track the complexity of the PINN learning problem with the Laplacian of components of the PINN loss function. We also demonstrate that our notion of benchmark “complexity” or “difficulty” correlates with this measure of complexity for some benchmarks.

#### C. Relative error analysis

In Figure 1, we show that PINNs fail to solve the discretized heat equation problem (Section II-C1) as the discretization size $N$ increases. This happens across variations in network architecture and PINN formulation. Despite our normalization of the residual loss by the norm of the discretized differential operator, the uniform PINN formulation (Equation (1)) fails to solve the problem, and the adaptive PINN (Equation (2)) performs worse.

In 2, we show that PINNs fail to solve the SHM problem (Section II-C2) as the maximum time $T$ of the domain increases. This happens despite our variations in network architecture and PINN formulation, as well as the fact that we increase the number of training points $D$ as the maximum time $T$ increases (Appendix B, Table II). All configurations are able to effectively learn the problem’s initial condition—in Figure 5 in Appendix C, we show that the average relative errors for the initial condition do not exceed 0.02. In this problem, the adaptive weighting scheme [3] attains the lowest relative error but is still dependent on other hyperparameters.

In Figure 2 (SHM) but not Figure 1 (heat equation), deeper networks were able to solve more complex problems—although the high-end of time domain size still results in a failed PINN. This is likely not an indication that increased network depth is a solution, as the underlying phenomena has a simple periodic structure with fixed complexity. Thus, the PINN is not discovering the periodic structure in SHM and can only learn the DE’s behavior with very high-capacity networks. This works in this SHM setting, but will not scale to complex long-time horizon phenomena.

#### D. Laplacian analysis

In Figure 4, we analyze the Laplacian $\Delta L_c$ of the components of the heat equation loss function. We seek trends in the Laplacian as the number of equations varies; thus, we normalize the Laplacian of the residual by the condition number of the discretized differential operator and the Laplacian of the initial condition error by the number of equations. Even with this normalization, there remains a positive correlation between the number and equations and the Laplacian of the residual—the PINN formulation cannot handle the ill-conditioning of the heat equation benchmark.

In Figure 3, we analyze the Laplacian $\Delta L_c$ of the components of the SHM loss function. For ResNet architectures, the Laplacian of the residual loss does increase as the maximum time $T$ also increases; however, this increase is not also found for MLP architectures.

### IV. Discussion

In this paper, we demonstrate that, in a pair of benchmarks, the relative error of a PINN falters as problem complexity increases. Our results serve as a cautionary tale in that the
Fig. 1. In the heat equation benchmark, all PINNs fail to solve the problem as dimensionality $N$ increases. Adaptive weighting in the domain [3] performs worse than uniform weighting—the weights $\lambda$ cannot mitigate the increase in ill-conditionedness of the problem. Interestingly, increasing network depth and width have little effect on the final relative error. Here we show relative error results for MLPs initiated with a learning rate of $10^{-3}$; results for other PINN configurations are comparable or worse.

Fig. 2. For the SHM benchmark and network depths of 2, 4, and 8, all PINNs increasingly fail to solve the problem as the maximum time $T$ increases. Depth reduces relative error, although error is still dependent on other hyperparameters—the high learning rate of 0.001 and a MLP perform the best. Here we show error results for models under the adaptive weighting PINN formulation with a width of 64 units; results for other PINN configurations are comparable or worse. Upper bounds on $T$ are of form $T_{\text{max}} \pi$ when actually training the PINNs. Precise values of maximum time are given in Table II in Section B.

Fig. 3. Analysis of the Laplacian $\Delta L_c$ of components of the SHM loss function. The Laplacian of the initial condition loss remains fairly constant as network configurations and maximum time $T$ change, reflecting how the initial condition error for this problem is easily satisfied. Interestingly, the Laplacian of PINNs with ResNet networks is consistently much larger than those with MLPs, which aligns with the findings of [17] but contradicts findings of [30]; both of whom consider computer vision problems. We show results for the uniform PINN loss formulation, initiated with a learning rate of $10^{-3}$; other results are comparable.

ability of modern ML to “reason” in the language of physical laws still remains challenging.

Our experiments can be expanded on in several ways. First, we only consider forward problems, in the assumption that inverse problems would require robust forward prediction capability. This seems necessary, but may not be sufficient for successful solutions to inverse problems. Second, our current results may benefit from a larger sweep of model hyperparameters—which could provide additional insight into PINN behavior and increase confidence in our results. Finally, our choice of two benchmarks is focused on PINN performance for ODEs—as we discuss below, a broader benchmark suite is likely needed.

The two benchmarks chosen for this paper are simple and representative of actual DE modeling problems encountered in scientific applications. We identify Hessian-based metrics as being sometimes representative of a PINN’s ability to solve a problem, but further work remains in a devising a more general classification of PINNs and DEs. This can assist in evaluating novel approaches for PINN architectures, loss functions, and training schemes. Our benchmarks can easily be made more extensive. For example, more than one
dimension of complexity can be varied, and a variety of initial conditions can be specified. In addition, similar benchmarks can be developed for PDEs. We believe that a robust suite of benchmarks for physics-informed ML models is needed to facilitate reproducibility and to aid validation of models for difficult problems—such as large, stiff, and multi-scale systems, as well as inverse problems.

REFERENCES

[1] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, “Physics-Informed Machine Learning,” Nature Reviews Physics, vol. 3, no. 6, pp. 422–440, 2021.

[2] 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,” J. Comp. Phys., vol. 379, pp. 686–707, 2019.

[3] L. McClenney and U. Braga-Neto, “Self-Adaptive Physics-Informed Neural Networks using a Soft Attention Mechanism,” 2020, doi:10.48550/ARXIV.2009.04544.

[4] 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.

[5] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney, “Characterizing Possible Failure Modes in Physics-Informed Neural Networks,” Advances Neural Inf. Process. Syst., vol. 34, 2021.

[6] S. Wang, Y. Teng, and P. Perdikaris, “Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks,” SIAM Journal on Scientific Computing, vol. 43, no. 5, pp. A3055–A3081, 2021.

[7] S. Wang, X. Yu, and P. Perdikaris, “When and Why PINNs Fail to Train: A Neural Tangent Kernel Perspective,” Journal of Computational Physics, vol. 449, p. 110768, 2022.

[8] V. Singhal, Z. A. Tuza, Z. Z. Sun, and R. M. Murray, “A MATLAB Toolbox for Modeling Genetic Circuits in Cell-Free Systems,” Synthetic Biology, vol. 6, no. 1, 02 2021.

[9] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng, “Stiff-PINN: Physics-Informed Neural Network for Stiff Chemical Kinetics,” The Journal of Physical Chemistry A, vol. 125, no. 36, pp. 8098–8106, 2021.

[10] J. Verwer and J. Sanz-Serna, “Convergence of Method of Lines Approximations to Partial Differential Equations,” Computing, vol. 33, no. 3–4, pp. 297–313, Jan. 1984.

[11] B. Zogheib, E. Tohid, H. M. Baskonus, and C. Cattani, “Method of Lines for Multi-Dimensional Coupled viscous Burgers’ Equations via Nodal Jacobi Spectral Collocation Method,” Physica Scripta, vol. 96, no. 12, p. 124011, Aug 2021.

[12] S.-N. Chow, J. Mallet-Paret, and E. S. Van Vleck, “Dynamics of Lattice Differential Equations,” Int. J. Bifurcation Chaos, vol. 06, no. 09, pp. 1605–1621, 1996.

[13] K. Hornik, M. Stinchcombe, and H. White, “Multilayer Feedforward Networks are Universal Approximators,” Neural Networks, vol. 2, no. 5, pp. 359–366, 1989.

[14] Y. Shin, “On the Convergence of Physics-Informed Neural Networks for Linear Second-Order Elliptic and Parabolic Type PDEs,” Comm. Comp. Phys., vol. 28, no. 5, pp. 2042–2074, Jun 2020.

[15] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” 2014, doi:10.48550/ARXIV.1412.6980.

[16] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, “Automatic Differentiation in Machine Learning: A Survey,” J. Mach. Learn. Res., vol. 18, no. 1, p. 5595–5637, Jan 2017.

[17] Z. Yao, A. Gholami, K. Keutzer, and M. W. Mahoney, “Pyhessian: Neural Networks Through the Lens of the Hessian,” in 2020 IEEE Int. Conf. Big Data (Big Data), IEEE, 2020, pp. 581–590.

[18] B. A. Pearlmutter, “Fast Exact Multiplication by the Hessian,” Neural Computation, vol. 6, no. 1, pp. 147–160, 01 1994.

[19] J. Dormand and P. Prince, “A Family of Embedded Runge-Kutta Formulas,” Journal of Computational and Applied Mathematics, vol. 6, no. 1, pp. 19–26, 1980.

[20] K. Zabov, Z. McCarthy, Y. Ma et al., “NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations,” 2021, doi: 10.48550/ARXIV.2107.09443.

[21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[22] S. Noschese, L. Pasquini, and L. Reichel, “Tridiagonal Toeplitz Matrices: Properties and Novel Applications,” Numerical Linear Algebra with Applications, vol. 20, no. 2, pp. 302–326, 2013.

[23] S. Maddu, D. Sturm, C. L. Müller, and I. F. Shalzarinzi, “Inverse Dirichlet Weighting Enables Reliable Training of Physics Informed Neural Networks,” Machine Learning: Science and Technology, vol. 3, no. 1, p. 015026, feb 2022.

[24] A. Jacot, F. Gabriel, and C. Hongler, “Neural Tangent Kernel: Convergence and Generalization in Neural Networks,” in Advances in Neural Information Processing Systems, vol. 31, 2018.

[25] S. Wang, S. Sankaran, and P. Perdikaris, “Respecting Causality is all you Need for Training Physics-Informed Neural Networks,” 2022, doi:10.48550/ARXIV.2203.07404.

[26] A. Daw, J. Bu, S. Wang, P. Perdikaris, and A. Karpatne, “Mitigating propagation failures in pinns using evolutionary sampling,” 2022, doi:10.48550/ARXIV.2207.02338.

[27] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney, “ADAHESIAN: An Adaptive Second Order Optimizer for Machine Learning,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 12, pp. 10665–10673, May 2021.

[28] X. Meng, Z. Li, D. Zhang, and G. E. Karniadakis, “PPINN: Parareal Physics-Informed Neural Network for Time-Dependent PDEs,” Comput. Methods Appl. Mechanics Eng., vol. 370, p. 113250, 2020.

[29] S. Wang and P. Perdikaris, “Long-time Integration of Parametric Evolution Equations with Physics-Informed DeepONets,” 2021, doi: 10.48550/ARXIV.2106.05384.

[30] C. Li, H. Farkhoor, R. Liu, and J. Yosinski, “Measuring the Gradient Flow Pathologies in Physics-Informed Neural Networks,” 2021, doi:10.48550/ARXIV.2207.02338.
APPENDIX A
DETAILS ON THE HEAT EQUATION

The continuous-space formulation of the heat equation is:

\[
\frac{\partial}{\partial t} u = \frac{\partial^2}{\partial x^2} u
\]
\[
\begin{align*}
  u(x, 0) &= g(x) \\
  u(0, t) &= u_L(t) \\
  u(1, t) &= u_R(t) \\
  (x, t) &\in (0, 1) \times (0, T),
\end{align*}
\]

where \(g\) specifies the initial condition, and \(u_L\) and \(u_R\) specify the boundary conditions on the left and right side, respectively, of the spatial domain \((0, 1)\). We use an initial condition of \(g(x) = \sin(2\pi x) + 1\) and boundary conditions of \(u_L(t) = u_R(t) = 1\).

The central finite difference approximation to \(\frac{\partial}{\partial x^2}\) is:

\[
\frac{\partial^2}{\partial x^2} u(x_n, t) = \frac{u(x_{n+1}, t) - 2u(x_n, t) + u(x_{n-1}, t)}{(\delta x)^2},
\]

where \(\delta x\) is the discretization in \(x\), yields the system of linear equations described in Section II-C1.

Because the discretized differential operator \(A\) is a symmetric tridiagonal Toeplitz matrix, it has a closed-form representation of its eigenvalues [22]:

\[
e_n = -2(N - 1)^2 (1 - \cos(n\pi/(N + 1))),
\]

for \(n = 1, \ldots, N\).

In this case, the condition number is given by:

\[
\kappa_N = \frac{|e_N|}{|e_1|} = \frac{1 - \cos(N\pi/(N + 1))}{1 - \cos(\pi/(N + 1))}
\]

Since the condition number increases without bound as the discretization size \(N\) increases, we have an explicit connection between an intuitive notion of complexity \((N)\) and a more formal notion of complexity (the condition number of the matrix \(A\)).

APPENDIX B
SUPPLEMENTAL TABLES

| Hyperparameter          | Value(s) used          | Other parameters |
|-------------------------|------------------------|------------------|
| # Layers                | 2, 4, 8                | \(N = 2\), \(\nu_T = 1\) |
| # Hidden units          | 64, 128                |                  |
| Initial learning rate   | \(10^{-3}\), \(10^{-4}\) |                  |
| Network type            | MLP, ResNet            |                  |
| PINN formulation        | Uniform, Adaptive      |                  |
| Activation function     | tanh                   |                  |
| Masking function \(\mu\) | Sigmoid                | \(N = 2\), \(\nu_T = 1\) |
| # Adam iterations       | 10, 241                |                  |
| # Training points \(D\) (SHM) | 256\(T/\pi\)          |                  |
| # Training points \(D\) (heat equation) | 1,024 |                  |

TABLE I

Hyperparameters searched over when solving PINN problems. For the PINN formulation, Uniform is the standard PINN formulation Equation (1), and Adaptive is the adaptive-weighted PINN formulation Equation (2). The heat equation benchmark uses a fixed number of training points, since their domains do not vary in size; the SHM benchmark scales the number of training points with the time horizon \(T\).

| Problem                | Complexity parameter | Values evaluated | Other parameters |
|------------------------|----------------------|------------------|------------------|
| Simple harmonic motion | Maximum time         | \(T = 2\pi, \ell = 0, 1, \ldots, 5\) | \(N = 2\), \(\nu_T = 1\) |
| Heat equation          | # discretized points | \(N = 4, 8, \ldots, 256, 512\) | \(\nu_T = \|\mathbf{A}\|_2, T = 0.1\) |

TABLE II

ODE system parameters evaluated. For Section II-C1, since the norm of \(A\) scales with the number of discretizations \(N\) and the initial condition remains constant, we scale the initial condition error by \(\|\mathbf{A}\|_2\) during training. For Section II-C1, we choose the maximum time \(T\) so that the solution has decayed to a constant value.
(a) An example solution to the heat equation (Section II-C1). At $t = 0$, the system has a sinusoidal shape; as $t$ increases, the solution decays to a constant value. As the discretization of the spatial dimension increases, the problem becomes more ill-conditioned.

(b) A solution to the SHM problem (Section II-C2); as the size of the time domain increases, the model becomes harder to solve with a PINN. The initial condition $u_0 = (0, \pi/2)$ was used with a maximum time of $T = 4\pi$ and a frequency of $\omega = 1$.

**APPENDIX C**

**SUPPLEMENTAL FIGURES**

Fig. 5. For the SHM problem, all PINNs are able to consistently minimize the relative initial condition error. Here we plot the final relative errors, averaged over all network configurations. Error bars indicate 95% confidence intervals for the estimate of the mean.

Fig. 6. Even though PINNs fail to solve the high-dimensional heat equation over the entire domain, the standard PINN formulation is able to learn the initial condition.