Residual-Based Error Bound for Physics-Informed Neural Networks

Shuheng Liu[1] **Xiyue Huang**[2] **Pavlos Protopapas**[3]

[1, 3] Institute for Applied Computational Science, Harvard University, Cambridge, Massachusetts, USA
[2] Data Science Institute, Columbia University, New York, New York, USA

Abstract

Neural networks are universal approximators and are studied for their use in solving differential equations. However, a major criticism is the lack of error bounds for obtained solutions. This paper proposes a technique to rigorously evaluate the error bound of Physics-Informed Neural Networks (PINNs) on most linear ordinary differential equations (ODEs), certain nonlinear ODEs, and first-order linear partial differential equations (PDEs). The error bound is based purely on equation structure and residual information and does not depend on assumptions of how well the networks are trained. We propose algorithms that bound the error efficiently. Some proposed algorithms provide tighter bounds than others at the cost of longer run time.

1 INTRODUCTION

Differential equations (DEs) are a useful mathematical tool for describing various phenomena in natural sciences, engineering, and humanity studies. As universal approximators, neural networks are powerful in approximating unknown functions. With back-propagation and modern computing devices, neural networks are convenient to differentiate, making them an ideal choice for solving differential equations.

However, a major criticism of neural network solutions to DEs is the lack of error bound. Traditional numerical methods, such as the finite difference method (FDM) and the finite element method (FEM), compute numerical solutions with known error bounds. Unlike traditional methods, the error bounds of neural network solutions are not well-studied. Therefore, solving DEs with neural networks requires ad hoc customization and empirical hyperparameter finetuning. If the error of *any* given network can be bounded, we can train neural networks until the error falls below a specified tolerance threshold.

Our contribution is that we propose rigorous error-bounding algorithms for any neural network solution to certain classes of equations, including linear ODEs, certain nonlinear ODEs, and first-order linear PDEs. These algorithms can also be extended to bound the error of other classes of equations as well. The proposed algorithms only use residual information and equation structure as inputs and do not rely on assumptions of finetuning.

Section 2 introduces the symbols and notations adopted in this paper. Section 3 reviews the literature on emerging areas of research that are relevant to solving DEs with neural networks. Section 4 explains the existing effort to bounding the error of neural network DE solutions. Sections 5 and 6 propose various algorithms for the error bound of ODEs and PDEs, respectively. Section 7 uses the method of manufactured solution to verify the validity of each error-bounding algorithm and provides visualization of the tightness of the bounds.

2 SYMBOLS AND NOTATIONS

DEs in this paper are posed w.r.t. unknown function v,

$$\mathcal{D}v = f,$$

where $\mathcal{D}$ is a possibly nonlinear differential operator and f is some forcing function. Unlike the exact solution $v(\cdot)$, a neural network solution $u(\cdot)$ does not strictly satisfy the equation. Instead, it incurs an additional residual term, r, which the network aims to minimize, to the equation,

$$\mathcal{D}u = f + r.$$

The input to v, u, f, and r is time t for ODEs and spatial coordinates (x, y) for PDEs. We limit our reasoning to 2-dimensional PDEs in this work. In cases with multiple unknown functions, we use vector notations $\mathbf{v}$, $\mathbf{u}$, and $\mathbf{r}$ instead of the scalar notations v, u, and r.

The loss function of the network solution is defined as the L^2 norm of residual r over the domain of interest,

$$\mathrm{Loss}(u) := \frac{1}{|I|}\int_I \|r\|^2 \mathrm{d}I = \frac{1}{|I|}\int_I \|\mathcal{D}u - f\|^2 \mathrm{d}I, \quad (1)$$

where a spatial domain Ω is substituted for the temporal domain I in the case of a PDE.

2.1 INITIAL AND BOUNDARY CONDITIONS

For a neural network to satisfy initial or boundary conditions, we apply a technique called *parametrization*. As an intuitive example, the parametrization $u(t) = (1-e^{-t})\mathrm{Net}(t)+v(0)$ guarantees that $u(t)$ satisfies the initial condition $u(0) = v(0)$ regardless of the network $\mathrm{Net}(\cdot)$. This does not affect the capability of $\mathrm{Net}(\cdot)$ to learn any solution.

The parametrization is more complicated for higher-order ODEs and most PDEs and has been extensively studied by Lagaris et al. [1998], Lagaris et al. [2000], McFall and Mahan [2009], Lagari et al. [2020], and Sukumar and Srivastava [2021]. In this work, we assume all initial and boundary conditions are exactly satisfied.

2.2 ERROR AND ERROR BOUND

The error of a network solution u is defined as

$$\eta := u - v. \quad (2)$$

We are interested in *bounding* the error with a scalar function $\mathcal{B}$ such that

$$\|\eta(t)\| \le \mathcal{B}(t) \quad \text{or} \quad \|\eta(x,y)\| \le \mathcal{B}(x,y) \quad (3)$$

where $\|\eta\| = \|u - v\|$ is the *absolute error*. If $\mathcal{B}$ takes on the same value $B \in \mathbb{R}^+$ over the domain, it can be replaced with a constant B.

Notice that multiple bounds $\mathcal{B}$ exist for the same network solution u. For example, $|\eta(t)| \le \mathcal{B}^{(1)}(t) \le \mathcal{B}^{(2)}(t) \le \cdots \le B$ are bounds in decreasing order of tightness. Tighter bounds incur a higher computational cost, and looser bounds (such as constant B) are faster to compute.

A summary of the applicability, restraints, run-time complexity, and relative tightness of all proposed algorithms is listed in Table 1.

3 LITERATURE REVIEW

Hornik et al. [1989] showed that neural networks are universal function approximators. Lagaris et al. [1998] first studied the application of neural networks in solving DEs. The term *physics-informed neural networks*, or PINNs, was first introduced by Raissi et al. [2019] to name neural networks that satisfy DEs while fitting observed data points. Although we train PINNs only to solve DEs without any observed data in this work, the error-bounding algorithms we propose work for any given neural network, regardless of the training process.

Flamant et al. [2020] and Desai et al. [2021] showed that one main advantage of neural networks over traditional numerical methods, such as FDM and FEM, is that neural networks can potentially learn the structure of the solution space and give a bundle of solutions $u(\mathbf{x};\Theta)$ for different equation setup and initial/boundary conditions parameterized by Θ. For traditional methods, a new solution must be recomputed for any slight changes in equation setup or initial/boundary conditions.

Some effort has been made to redefine the objective loss function. Yu et al. [2017] applied the Ritz method to a particular class of variational problems. Mattheakis et al. [2020] incorporated an additional constraint to force the network to learn solutions with energy conservation. Parwani and Protopapas [2021] used an adversarial network for sampling in particular areas of the domain where the residual is large.

There are also works that study the failure modes of PINNs and quantify the error of PINN solutions in recent years. Graf et al. [2021] worked on quantifying the uncertainty of PINNs using the Bayesian framework. Krishnapriyan et al. [2021] characterized possible failure modes of PINNs by studying the performance of PINNs on simple problems and analyzing their loss landscape. Krishnapriyan et al. [2021] also concluded that optimization difficulty is the essential cause of failure.

Our work uncovers the mathematical relationship between residual information and the error of PINNs on several classes of ODEs and PDEs. We propose different algorithms for various classes of equations and experimentally validate these algorithms.

4 EXISTING WORK

Sirignano and Spiliopoulos [2018] showed that for a class of quasi-linear parabolic PDEs, a neural network with a single hidden layer and sufficiently many hidden units could arbitrarily approximate the exact solutions. Guo and Haghighat [2022] proposed an energy-based *constitutive relation error* bound for elasticity problems.

De Ryck and Mishra [2022a] derived an error bound for ReLU networks on parametric hyperbolic conservation laws. De Ryck and Mishra [2022b] showed that there exists some PINN with arbitrarily small residual for Kolmogorov PDEs. De Ryck and Mishra [2022c] derived an error bound for operator learning with PINNs. The works of De Ryck and Mishra mentioned above did not bound the error of every given network. Instead, they mathematically proved the existence of a network with errors below a specified bound,

Table 1: **Overview of Proposed Algorithms.** The symbols in run-time analysis are defined and explained in detail in Sections 5 and 6, with the exception of K, which is the number of steps used in each numerical integration.

Algorithm	Applicable to	Restraint	Run-Time	Comment
Algorithm 1	Linear ODE	Semi-stable	$O(L)$	Looser than Alg 2
Algorithm 2	Linear ODE		$O(nL)$	Tighter than Alg 1
Algorithm 3	Linear ODE System		$O(n^3L)$	Norm and elementwise bounds
Algorithm 4	Nonlinear ODE	Nonlinear term is εv^k	$O(JnL)$	Bounded solution for family of DEs
Algorithm 5	Linear 1st-Order PDE	Coeff. $c \neq 0$ over domain	$O(\text{mesh})$	Constant bound; Looser than Alg 6
Algorithm 6	Linear 1st-Order PDE	Solvable characteristics	$O(KL)$	Tigher than Alg 5 if computable

under certain assumptions of network architecture, including width, depth, and activation functions. The question remaining to be answered is how to overcome optimization difficulties and find such a neural network.

Our work differs from the above in that we bound the error of *any* neural network regardless of finetuning, even networks with randomly initialized weights. Our algorithms only depend on inputs of residual information r, often used as training loss, and equations structure $\mathcal{D}v = f$. The output is a (possibly constant) function that guarantees to bound the error at any point in domain.

5 ERROR BOUND FOR ODE

This section considers both linear and nonlinear ODEs over the temporal domain $I = [0, T]$. Initial conditions are imposed on $\frac{\mathrm{d}^k}{\mathrm{d}t^k} v(t = 0)$ for $k = 0, \ldots, (n-1)$, where n is the highest order of derivative terms in ODE.

5.1 ERROR BOUND FOR LINEAR ODE

Consider the linear ODE $\mathcal{L}v(t) = f(t)$, where $\mathcal{L}$ is a linear differential operator. Its neural network solution u satisfies $\mathcal{L}u(t) = f(t) + r(t)$. Since error $\eta := u - v$, there is

$$\mathcal{L}\eta(t) = r(t). \tag{4}$$

With the assumption in Section 2.1 that u satisfies the initial conditions at $t = 0$, there is

$$\eta(0) = 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}\eta(0) = 0, \quad \frac{\mathrm{d}^2}{\mathrm{d}t^2}\eta(0) = 0, \quad \ldots \tag{5}$$

With initial conditions 5 known, a unique inverse transform $\mathcal{L}^{-1}$to $\mathcal{L}$ exists. Applying $\mathcal{L}^{-1}$ to Eq. 4, there is

$$\eta(t) = \mathcal{L}^{-1} r(t). \tag{6}$$

Hence, bounding the absolute error $|\eta|$ is equivalent to bounding $\left|\mathcal{L}^{-1}r\right|$. Notice that only a) the equation structure $\mathcal{L}$ and b) the residual information r are relevant to estimating the error bound. All other factors, including parameters of the neural network u, forcing function f, and initial conditions, do not affect the error bound at all.

5.1.1 Single Linear ODE with Constant Coefficients

Consider the case where $\mathcal{L} = \frac{\mathrm{d}^n}{\mathrm{d}t^n} + \sum_{j=0}^{n-1} a_j \frac{\mathrm{d}^j}{\mathrm{d}t^j}$ consists of only constant coefficients $a_0, a_1, \ldots, \in \mathbb{R}$. Its characteristic polynomial (defined below) can be factorized into

$$\lambda^n + a_{n-1}\lambda^{n-1} + \cdots + a_0 = \prod_{j=1}^{n} (\lambda - \lambda_j), \tag{7}$$

where $\lambda_1, \ldots, \lambda_n \in \mathbb{C}$ are the characteristic roots.

It can be shown that, for a semi-stable system ($\mathcal{R}e\,(\lambda_j) \leq 0$ for all λ_j), an error bound can be formulated as

$$|\eta(t)| \leq \mathcal{B}_{loose}(t) := C_{\lambda_{1:n}}\, R_{\max}\, t^Z, \tag{8}$$

where $0 \leq Z \leq n$ is the number of λ_j whose real part is 0, $C_{\lambda_{1:n}} := \frac{1}{Z!} \prod_{j=1;\lambda_j \neq 0}^{n} \frac{1}{\mathcal{R}e(-\lambda_j)}$ is a constant coefficient, and $R_{\max} := \max_{t \in I} |r(t)|$ is the maximum absolute residual. Knowing bound 8 is sufficient to qualitatively estimate the error for applications where only the order of error is concerned. See Alg. 1 for reference.

An issue with Eq. 8 and Alg. 1 is that they assume $\mathcal{R}e\,(\lambda_j) \leq 0$ for all characteristic roots λ_j. To address this issue, we propose an alternative error-bounding Alg. 2, which requires more computation but does not require the system to be semi-stable and provides a tighter bound.

Notice that the bounds of η in Eq. 6 can be estimated if the inverse operator $\mathcal{L}^{-1}$ is known. Let Eq. 7 be the factorization of characteristic polynomial of $\mathcal{L}$. Define operator $\mathcal{I}_\lambda$ as [1]

$$\mathcal{I}_\lambda \psi(t) := e^{\lambda t} \int_0^t e^{-\lambda\tau} \psi(\tau) \mathrm{d}\tau, \quad \forall \psi : I \to \mathbb{C}. \tag{9}$$

We show in supplementary material that $\mathcal{L}^{-1} = \mathcal{I}_{\lambda_n} \circ \mathcal{I}_{\lambda_{n-1}} \circ \cdots \circ \mathcal{I}_{\lambda_1}$ and that $|\mathcal{I}_\lambda \psi| \leq \mathcal{I}_{\mathcal{R}e(\lambda)} |\psi|$ for any $\lambda \in \mathbb{C}$ and function ψ. Hence, another error bound can be formulated as

$$\mathcal{B}_{tight}(t) := \left(\mathcal{I}_{\mathcal{R}e(\lambda_n)} \circ \cdots \circ \mathcal{I}_{\mathcal{R}e(\lambda_1)}\right) |r(t)|. \tag{10}$$

[1]This paper assumes the network solution exactly satisfies the initial conditions as discussed in Section 2.1. However, all our algorithms can be extended to cases where the network solution differs from the exact solution by some value. This is achieved by replacing $\mathcal{I}_\lambda \psi(t)$ with $\mathcal{I}_{\lambda,\delta} \psi(t) = \mathcal{I}_\lambda \psi(t) + \delta e^{\lambda t}$ in Eq. 9.

Algorithm 1 Loose Error Bound Estimation for Linear ODE with Constant Coefficients (Requires Semi-Stability)

```
Input: Coefficients $\{a_j\}_{j=0}^{n-1}$ for operator $\mathcal{L}$, residual information $r(\cdot)$, domain of interest $I = [0, T]$, and a sequence of time points $\{t_\ell\}_{\ell=1}^{L}$ where error bound is to be evaluated
Output: Error bound at given time points $\{\mathcal{B}(t_\ell)\}_{\ell=1}^{L}$
Require: $\mathcal{L}$ is semi-stable, and $t_\ell \in I$ for all $\ell$
Ensure: $|\eta(t_\ell)| \le \mathcal{B}(t_\ell)$ for all $\ell$
  $\{\lambda_j\}_{j=1}^{n} \leftarrow$ numerical roots of $\lambda^n + a_{n-1}\lambda^{n-1} + \cdots = 0$
  assert $\lambda_j \le 0$ for $1 \le j \le n$
  $Z, C \leftarrow 0, 1$
  for $j \leftarrow 1 \ldots n$ do
    if $\mathcal{R}e\,(\lambda_j) = 0$ then
      $Z \leftarrow Z + 1$
    else
      $C \leftarrow C / \mathcal{R}e\,(-\lambda_j)$
    end if
  end for
  $R_{\max} \leftarrow \max_{\tau \in I} |r(\tau)|$   ▷ Use linspace with mini-steps
  $\{\mathcal{B}(t_\ell)\}_{\ell=1}^{L} \leftarrow \{\frac{C}{Z!} R_{\max}\, t_\ell^{Z}\}_{\ell=1}^{L}$
  return $\{\mathcal{B}(t_\ell)\}_{\ell=1}^{L}$
```

Note: Jenkins and Traub [1970] solves polynomial roots.

It is also proven in supplementary material that $\mathcal{B}_{tight}$ is tighter than $\mathcal{B}_{loose}$ when $\mathcal{B}_{loose}$ is applicable,

$$|\eta(t)| \le \mathcal{B}_{tight}(t) \le \mathcal{B}_{loose}(t) \quad \forall t \in I. \tag{11}$$

Based on Eq. 10, we propose Alg. 2, which computes $\mathcal{B}_{tight}$ by repeatedly evaluating integrals in 9 using the cumulative trapezoidal rule.

Before moving on to the next section, we discuss the effect of numerical error in Alg. 2 and all pursuant algorithms which involve numerical integration. Empirically, the error introduced by numerical integration in negligible for most cases. To ensure the accuracy of the error bound, we recommend using a slightly modified trapezoidal rule that, instead of directly using the node value, takes the maximum of the node and its two adjacent nodes. This modification leads to a slighltly looser bound. Yet, this influence is almost negligible in practice.

5.1.2 Single Linear ODE of the General Form

In general, the coefficients for $\mathcal{L}$ can be functions of t. Namely, $\mathcal{L} = \frac{\mathrm{d}^n}{\mathrm{d}t^n} + \sum_{j=0}^{n-1} a_j(t) \frac{\mathrm{d}^j}{\mathrm{d}t^j}$. Similar to Eq. 7, $\mathcal{L}$ have characteristic roots $\{\lambda_j(t)\}_{j=1}^{n}$ as functions of t,

$$\lambda^n + a_{n-1}(t)\lambda^{n-1} + \cdots + a_0(t) = \prod_{j=1}^{n} (\lambda - \lambda_j(t)).$$

We can replace constant λ_j with functions $\lambda_j(t)$ in Eq. 9 and compute bound $\mathcal{B}_{tight}$ as in Eq. 10. However, the factorization in Eq. 5.1.2 is hard to implement in practice except

Algorithm 2 Tighter Error Bound Estimation for Linear ODE with Constant Coefficients (Stable and Unstable)

```
Input & Output: Same as Alg. 1
Require: Same as Alg. 1, except $\mathcal{L}$ can be unstable
Ensure: Same as Alg. 1
  $\{\lambda_j\}_{j=1}^{n} \leftarrow$ numerical roots of $\lambda^n + a_{n-1}\lambda^{n-1} + \cdots = 0$
  $\{t_k\}_{k=0}^{K} \leftarrow$ linspace(0, $T$, sufficient steps)
  $\{\mathcal{B}(t_k)\}_{k=0}^{K} \leftarrow \{|r(t_k)|\}_{k=0}^{K}$
  for $j \leftarrow 1 \ldots n$ do
    $\text{intgr}_{k=0}^{K} \leftarrow \text{CumTrap}(\{e^{-\lambda_j t_k}\mathcal{B}(t_k)\}_{k=0}^{K}, \{t_k\}_{k=0}^{K})$
    $\{\mathcal{B}(t_k)\}_{k=0}^{K} \leftarrow \{e^{\lambda_j t_k} \cdot \text{intgr}_k\}_{k=0}^{K}$
  end for
  $\{\mathcal{B}(t_\ell)\}_{\ell=1}^{L} \leftarrow \text{Interp}(\{\mathcal{B}(t_k)\}_{k=0}^{K}, \{t_k\}_{k=0}^{K}, \{t_\ell\}_{\ell=0}^{L})$
  return $\{\mathcal{B}(t_\ell)\}_{\ell=1}^{L}$
```

Note: CumTrap($\{y_k\}_{k=1}^{K}$, $\{x_k\}_{k=1}^{K}$) computes cumulative integral $\int_0^x y(x)\mathrm{d}x$ at discrete points $\{x_k\}_{k=1}^{K}$ using trapezoidal rule.

Note: Interp($\{y_k\}_{k=1}^{K}$, $\{x_k\}_{k=1}^{K}$, $\{x_\ell\}_{\ell=1}^{L}$) computes interpolant to a function with given discrete data points $\{(x_k, y_k)\}_{k=1}^{K}$ evaluated at $\{x_\ell\}_{\ell=1}^{L}$.

for the first-order case where $\mathcal{L}v = \frac{\mathrm{d}v}{\mathrm{d}t} + a_0(t)v$. Cases of second order and higher are out of the scope of this paper.

5.1.3 System of Linear ODEs with Constant Coefficients

Consider a system of linear ODEs with constant coefficients

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v} + A\mathbf{v} = \mathbf{f}(t) \tag{12}$$

where $\mathbf{v}$ and $\mathbf{f}$ are $\mathbb{R}^n$ vectors and A is a $n \times n$ matrix. Denote the Jordan canonical form of A as,

$$J = P^{-1}AP = \begin{pmatrix} J_1 & & \\ & \ddots & \\ & & J_K \end{pmatrix} \text{ where } J_k = \begin{pmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_k \end{pmatrix}. \tag{13}$$

Let n_k be the size of Jordan block J_k, we construct an operator matrix $\boldsymbol{\mathcal{I}} = \text{diag}(\mathbf{I}_1, \mathbf{I}_2, \ldots)$, where

$$\mathbf{I}_k = \begin{pmatrix} \mathcal{I}_{-\mathcal{R}e(\lambda_k)} & \mathcal{I}^2_{-\mathcal{R}e(\lambda_k)} & \ldots \mathcal{I}^{n_k}_{-\mathcal{R}e(\lambda_k)} \\ 0 & \mathcal{I}_{-\mathcal{R}e(\lambda_k)} & \ldots \mathcal{I}^{n_k-1}_{-\mathcal{R}e(\lambda_k)} \\ \vdots & \vdots & \ddots \quad \vdots \\ 0 & 0 & \ldots \mathcal{I}_{-\mathcal{R}e(\lambda_k)} \end{pmatrix}. \tag{14}$$

An *elementwise bound* (vector) $\boldsymbol{\mathcal{B}}(t)$ can be formulated as

$$\boldsymbol{\eta}^{|\cdot|}(t) \preceq \boldsymbol{\mathcal{B}}(t) := P^{|\cdot|}\boldsymbol{\mathcal{I}}\left[(P^{-1})^{|\cdot|}\,\mathbf{r}^{|\cdot|}\right](t), \tag{15}$$

where superscript $|\cdot|$ denotes taking elementwise absolute value and symbol $\preceq$ denotes elementwise inequality. In the meantime, a *norm bound* (scalar) $\mathcal{B}(t)$ also exists

$$\|\boldsymbol{\eta}(t)\| \leq \mathcal{B}(t) := \operatorname{cond}(P) \left\|\boldsymbol{\mathcal{I}}\big[\|\mathbf{r}\|\mathbf{1}\big](t)\right\| \tag{16}$$

where $\operatorname{cond}(P)$ is the conditional number of P w.r.t. induced matrix norm and $\mathbf{1}$ is an $n \times 1$ column vector of 1s. Proof of Eq. 15 and Eq. 16 can be found in in supplementary material. See Alg. 3 for implementation.

Algorithm 3 ODE System Bound (norm and elementwise)

```
Input: Coefficient matrix $A \in \mathbb{R}^{n\times n}$, residual vector $\mathbf{r}(t)$, and a sequence of points $\{t_\ell\}_{\ell=1}^L$ where error is to be bounded
Output: Norm bound (scalar) $\{\mathcal{B}(t_\ell)\}_{\ell=1}^L$ and componentwise bound (vector) $\{\boldsymbol{\mathcal{B}}(t_\ell)\}_{\ell=1}^L$ at given time points
Ensure: $\|\eta(t_\ell)\| \leq \mathcal{B}(t_\ell)$ and $\eta(t_\ell) \preceq \boldsymbol{\mathcal{B}}(t_\ell)$ for all $\ell$
  $J, P \leftarrow$ Jordan canonicalization of $A = PJP^{-1}$
  for each Jordan block $J_k$ of shape $n_k \times n_k$ do
      $\mathbf{I}_k \leftarrow$ construct operator block using Eq. 14
  end for
  $\boldsymbol{\mathcal{I}} \leftarrow \operatorname{diag}(\mathbf{I}_1, \mathbf{I}_2, \dots)$
  $\{\boldsymbol{\mathcal{B}}(t_\ell)\}_{\ell=1}^L \leftarrow \{P^{|\cdot|}\boldsymbol{\mathcal{I}}\big[(P^{-1})^{|\cdot|}\mathbf{r}^{|\cdot|}\big](t_\ell)\}_{\ell=1}^L$
  $\{\mathcal{B}(t_\ell)\}_{\ell=1}^L \leftarrow \{\operatorname{cond}(P)\left\|\boldsymbol{\mathcal{I}}\big[\|\mathbf{r}\|\mathbf{1}\big](t_\ell)\right\|\}_{\ell=1}^L$
  return $\{\mathcal{B}(t_\ell)\}_{\ell=1}^L$, $\{\boldsymbol{\mathcal{B}}(t_\ell)\}_{\ell=1}^L$
```

5.2 NONLINEAR ODE

Nonlinear ODEs are hard to solve in general. In this work, we only deal with nonlinear ODEs with a single nonlinear term of the form $\varepsilon v^k(t)$, where $\varepsilon \in \mathbb{R}$ is a small number. Ideally, $|\varepsilon| \ll 1$. The exact requirement for ϵ is given in Section 5.2.2. The value of ε can vary within a certain range or be fixed. With the perturbation technique, we obtain a family of solutions $v(t;\varepsilon)$ parameterized by ε at the cost of solving a (countable) collection of equations. As explained below in Section 5.2.1, we train finitely many networks, each approximately solving an equation in the collection.

5.2.1 Perturbation Theory

Consider the nonlinear ODE with nonlinear term $\varepsilon v^k(t)$,

$$\mathcal{L}v(t) + \varepsilon v^k(t) = f(t), \tag{17}$$

where $\mathcal{L}$ is a linear differential operator discussed in 5.1 and initial conditions are specified for the system at time $t = 0$. Notice that each $\varepsilon \in \mathbb{R}$ corresponds to a solution $v(t;\varepsilon)$. We expand the solution $v(t;\varepsilon)$ in terms of ε

$$v(t;\varepsilon) = \sum_{j=0}^{\infty} \varepsilon^j v_j(t) = v_0(t) + \varepsilon v_1(t) + \dots \tag{18}$$

Only $v_0(t)$ is subject to the original initial conditions at $t = 0$, while other components, v_1, v_2, $\dots$, have initial conditions of 0 at $t = 0$. Substituting Eq. 18 into Eq. 17,

$$\mathcal{L}\sum_{j=0}^{\infty}\varepsilon^j v_j + \varepsilon\left(\sum_{j=0}^{\infty}\varepsilon^j v_j\right)^k = f \tag{19}$$

$$\sum_{j=0}^{\infty}\varepsilon^j \mathcal{L}v_j + \sum_{j=0}^{\infty}\varepsilon^{j+1}\sum_{\substack{j_1+\cdots+j_k=j\\ j_1,\dots,j_k\geq 0}} v_{j_1}\dots v_{j_k} = f \tag{20}$$

$$\mathcal{L}v_0 + \sum_{j=1}^{\infty}\varepsilon^j\left(\mathcal{L}v_j + \sum_{\substack{j_1+\cdots+j_k=j-1\\ j_1,\dots,j_k\geq 0}} v_{j_1}\dots v_{j_k}\right) = f. \tag{21}$$

In order for Eq. 21 to hold true for all ε, the coefficients for each ε^j must match on both sides of Eq. 21. Hence,

$$\mathcal{L}v_0 = f \tag{22}$$
$$\mathcal{L}v_1 + v_0^k = 0 \tag{23}$$
$$\mathcal{L}v_2 + kv_0^{k-1}v_1 = 0 \tag{24}$$
$$\mathcal{L}v_3 + \frac{k(k-1)}{2}v_0^{k-2}v_1^2 + kv_0^{k-1}v_2 = 0 \tag{25}$$
$$\vdots \qquad \vdots$$

For $\varepsilon = 0$, Eq. 18 is reduced to $v_0(t)$, which solves the linear problem $\mathcal{L}v = f$.
The above system can be solved in a *sequential* manner, either analytically or using neural networks,

1. Eq. 22 is linear in v_0 and can be solved first.
2. With v_0 known, Eq. 23 is linear in v_1 and can be solved for v_1.
3. Similarly, with v_0 and v_1 known, Eq. 24 is linear in v_2 and can be solved for v_2.
4. The process can be repeated for Eq. 25 and beyond. Only a linear ODE is solved each time.

To solve the system with PINNs, we approximate exact solutions $\{v_j(t)\}_{j=1}^{\infty}$ with neural network solutions $\{u_j(t)\}_{j=0}^{J}$ trained sequentially on Eq. 22, Eq. 23, and beyond. In practice, we only consider components up to order J to avoid the infinity in expansion 18. Ideally, J should be large enough so that higher order residuals in expansion 18 can be neglected.

After obtaining $\{u_j(t)\}_{j=0}^{J}$, we can reconstruct the solution $u(t;\varepsilon) = \sum_{j=0}^{J}\varepsilon^j u_j(t)$ to the original nonlinear equation 17 for varying ε. See Alg. 4 for details.

5.2.2 Expansion of Bounds

The absolute error $|\eta(t;\varepsilon)| = |u(t;\varepsilon) - v(t;\varepsilon)|$is given by

$$|\eta(t;\varepsilon)| = \left|\sum_{j=0}^{J}\varepsilon^j\Big(u_j(t) - v_j(t)\Big) - \sum_{j=J+1}^{\infty}\varepsilon^j v_j(t)\right|$$

$$\leq \sum_{j=0}^{J} \Big|\eta_j(t)\Big| |\varepsilon|^j + \left|\sum_{j=J+1}^{\infty} \varepsilon^j v_j(t)\right| \tag{26}$$

where $\eta_j(t) := u_j(t) - v_j(t)$ is the *component error* between $u_j(t)$ and $v_j(t)$. Let $\mathcal{B}_j$ denote the *bound component* such that $|\eta_j(t)| \leq \mathcal{B}_j(t)$. Assuming J is large and ε is small such that higher order terms $\left|\sum_{j=J+1}^{\infty} \varepsilon^j v_j(t)\right|$ are negligible, there is

$$\Big|\eta(t;\varepsilon)\Big| \leq \mathcal{B}(t;\varepsilon) := \sum_{j=0}^{J} \mathcal{B}_j(t)\,|\varepsilon|^j \tag{27}$$

where each bound component $\mathcal{B}_j$ can be evaluated using the techinque in Section 5.1. See Alg. 4 for details.

Algorithm 4 Iterative Method for Solution and Error Bound of Nonlinear ODE 17

```
Input: Linear operator $\mathcal{L}$, nonlinear degree $k$, domain $I = [0, T]$, highest order $J$ for expansion, and a sequence $\{(t_\ell, \varepsilon_\ell)\}_{\ell=1}^{L}$ where solution $u(t;\varepsilon)$ and error bound $\mathcal{B}(t;\varepsilon)$ are to be evaluated
Output: Solution $\{u(t_\ell;\varepsilon_\ell)\}_{\ell=1}^{L}$ and error bound $\{\mathcal{B}(t_\ell;\varepsilon_\ell)\}_{\ell=1}^{L}$
Require: $t_\ell \in I$, and $|\varepsilon_\ell|$ to be small (ideally $|\varepsilon_\ell| \ll 1$)
Ensure: $\eta(t_\ell;\varepsilon_\ell) \leq \mathcal{B}(t_\ell;\varepsilon_\ell)$
  $u_0, r_0, \leftarrow$ net solution, residual of $\mathcal{L}u_0 = f$
  $\{\mathcal{B}_0(t_\ell)\}_{\ell=1}^{L} \leftarrow$ bound of $\left|\mathcal{L}^{-1} r_0\right|$ at $\{t_\ell\}_{\ell=1}^{L}$
  for $j \leftarrow 1 \dots J$ do
      Macro $\mathrm{NL}_j[\phi] \leftarrow \sum_{\substack{j_1+\dots+j_k=j-1 \\ j_1,\dots,j_k \geq 0}} \phi_{j_1} \dots \phi_{j_k}$
      $u_j, r_j \leftarrow$ net solution, residual of $\mathcal{L}u_j + \mathrm{NL}_j[u] = 0$
      $\mathcal{B}_{\mathrm{NL}} \leftarrow$ upper bound of $|\mathrm{NL}_j[u] - \mathrm{NL}_j[v]|$
      $\{\mathcal{B}_j(t_\ell)\}_{\ell=1}^{L} \leftarrow$ bound of $|\mathcal{L}^{-1} r_j|$+$|\mathcal{L}^{-1}\mathcal{B}_{\mathrm{NL}}|$
  end for
  $\{u(t_\ell;\varepsilon_\ell)\}_{\ell=1}^{L} \leftarrow \left\{\sum_{j=0}^{J} \varepsilon_\ell^j u_j(t_\ell)\right\}_{\ell=1}^{L}$
  $\{\mathcal{B}(t_\ell;\varepsilon_\ell)\}_{\ell=1}^{L} \leftarrow \left\{\sum_{j=0}^{J} \varepsilon_\ell^j \mathcal{B}_j(t_\ell)\right\}_{\ell=1}^{L}$
  return $\{u(t_\ell;\varepsilon_\ell)\}_{\ell=1}^{L}$ , $\{\mathcal{B}(t_\ell;\varepsilon_\ell)\}_{\ell=1}^{L}$
Note 1: $\mathcal{B}_0$ and $\mathcal{B}_{1:J}$ can be evaluated using Alg. 1 or 2.
Note 2: $\mathcal{B}_{\mathrm{NL}}$ can be estimated even though exact solutions $v_{0:j-1}$ are unknown. This is because $v_i \in [u_i - \mathcal{B}_i, u_i + \mathcal{B}_i]$ for all $i$, and $u_{0:j-1}$, $\mathcal{B}_{0:j-1}$ are known from previous iterations.
```

6 ERROR BOUND FOR PDE

This section considers first-order linear PDEs defined on a 2-dimensional spatial domain Ω,[2]

$$a(x,y)\frac{\partial v}{\partial x} + b(x,y)\frac{\partial v}{\partial y} + c(x,y)v = f(x,y) \tag{28}$$

[2]Similar techniques can be used for other classes of PDEs and higher dimensions where the method of characteristics applies.

with Dirichlet boundary constraints defined on $\Gamma \subset \partial\Omega$,

$$v\big|_{(x,y)\in\Gamma} = g(x,y). \tag{29}$$

We partition the domain into infinitely many characteristic curves $\mathcal{C}$, each passing through a point $(x_0, y_0) \in \Gamma$. The resulting curve is a parameterized integral curve

$$\mathcal{C} : \begin{cases} x'(s) = a(x,y) \\ y'(s) = b(x,y) \end{cases} \text{where } (\cdot)' = \frac{\mathrm{d}}{\mathrm{d}s} \text{ and } \begin{matrix} x(0) = x_0 \\ y(0) = y_0. \end{matrix}$$

For any $(x(s), y(s))$ on $\mathcal{C}$, functions (v, a, b, c, f) can be viewed as univariate functions of s. By chain rule, there is

$$a(x,y)\frac{\partial v}{\partial x} + b(x,y)\frac{\partial v}{\partial y} = x'(s)\frac{\partial v}{\partial x} + y'(s)\frac{\partial v}{\partial y} = v'(s).$$

Hence, Eq. 28 is reformulated as an ODE along curve $\mathcal{C}$,

$$v'(s) + c(s)v(s) = f(s) \quad \text{s.t. } v(0) = g(x_0, y_0), \tag{30}$$

where $v(s)$, $c(s)$, and $f(s)$ are shorthand notations for $v(x(s), y(s))$, $c(x(s), y(s))$, and $f(x(s), y(s))$, respectively.

In particular, if $c(x,y) \neq 0$ for all $(x,y) \in \Omega$, both sides of Eq. 28 can be divided by $c(x,y)$, resulting in a residual of $r(x,y)/c(x,y)$ where $r(x,y)$ is the residual of the original problem. By Eq. 8, a constant error bound on $\mathcal{C}$ is $|\eta(s)| \leq \max_s |r(s)/c(s)|$. Hence, a (loose) constant error bound B (see Alg. 5) over the entire domain Ω is

$$|\eta(x,y)| \leq B := \max_{(x,y)\in\Omega} \left|\frac{r(x,y)}{c(x,y)}\right|. \tag{31}$$

Algorithm 5 Constant Err Bound for Linear 1st-Order PDE

```
Input: Coefficient $c(x,y)$ in Eq. 28, residual information $r(x,y)$ and domain of interest $\Omega$
Output: A constant error bound $B \in \mathbb{R}^+$
Require: $c(x,y) \neq 0$ for all $(x,y) \in \Omega$
Ensure: $|\eta(x,y)| \leq B$ for all $(x,y) \in \Omega$
  $\{(x_k, y_k)\}_k \leftarrow$ sufficiently dense mesh grid over $\Omega$
  $B \leftarrow \max_k \left|\frac{r(x_k, y_k)}{c(x_k, y_k)}\right|$
  return $B$
```

Independent of the assumption $c(x,y) \neq 0$, in scenarios where the curve $\mathcal{C}$ passing through any (x,y) can be computed, the error can be computed using Alg. 6.

7 RELEVANT EXPERIMENTS

In this section, we perform experiments on equations with manufactured solutions using the *NeuroDiffEq* library [Chen et al., 2020], which provides convenient tools for training PINNs.

Algorithm 6 General Err Bound for Linear 1st-Order PDE

```
Input: Coefficients $a(x,y)$, $b(x,y)$, $c(x,y)$ in Eq. 28, residual information $r(x,y)$, domain of interest $\Omega$, Dirichlet boundary $\Gamma \subset \partial\Omega$, and a sequence of points $\{(x_\ell, y_\ell)\}_{\ell=1}^{L}$ where error is to be bounded
Output: Error bound $\{\mathcal{B}(x_\ell, y_\ell)\}_{\ell=1}^{L}$ at given points
Require: Integral curve of vector field $\left[a(x,y)\; b(x,y)\right]^T$ passing through any point $(x_\ell, y_\ell) \in \Omega$ is solvable
Ensure: $|\eta(x_\ell, y_\ell)| \le \mathcal{B}(x_\ell, y_\ell)$ for all $\ell$
  $\mathcal{C}_g \leftarrow$ general solution to $\begin{cases} x'(s) = a(x,y) \\ y'(s) = b(x,y) \end{cases}$
  for $\ell \leftarrow 1 \dots L$ do
      $x(s), y(s) \leftarrow$ instance of $\mathcal{C}_g$ passing through $(x_\ell, y_\ell)$
      $s^* \leftarrow$ solution to $x(s) = x_\ell,\ y(s) = y_\ell$
      $\mathcal{B}(x_\ell, y_\ell) \leftarrow e^{c(s^*)} \int_0^{s^*} r(x(s), y(s)) e^{-c(x(s),y(s))\, s} \mathrm{d}s$
  end for
  return $\{\mathcal{B}(x_\ell, y_\ell)\}_{\ell=1}^{L}$
```

First, we train networks to solve equations and collect their residual information r. Then, we apply Alg. 1–6 (where applicable) to derive error bounds using only residual information r and equation structure, characterized by its differential operator $\mathcal{D}$. Lastly, we show that the absolute error strictly falls within the bounds, regardless of how well the networks are trained.

Throughout this section, we always use networks with two hidden layers, each consisting of 32 hidden units. Depending on whether the problem is an ODE or PDE, a network can have a single input t or two inputs (x, y), but always have a single output. The activation function is $\tanh$. Unless otherwise noted, the training domain is $I = [0, 1]$ for ODEs and $\Omega = [0, 1]^2$ for PDEs. We use a *PyTorch* Adam optimizer with default hyperparameters to train networks for 1000 epochs.

Notice that we list these configurations only for the reproducibility of visualizations. Our error-bounding algorithm works under any other configurations.

7.1 SINGLE LINEAR ODE WITH CONSTANT COEFFICIENTS

Here, we study three equations $v'' + 3v' + 2v = f(t)$, $v'' + v = g(t)$, and $v'' - v' = h(t)$, whose characteristic roots are $\{-1, -2\}$, $\{\pm i\}$, and $\{0, 1\}$ respectively. By Section 5.1.1, the first two equations can be bounded with either Alg. 1 or Alg. 2, while the last must be bounded with Alg. 2.

They all satisfy initial conditions $v(0) = v'(0) = 1$. We pick $f(t) = 2t^2 + 8t + 7$, $g(t) = t^2 + t + 3$, and $h(t) = 1 - 2t$, so that the manufactured solution is $v(t) = t^2 + t + 1$ for all three equations. Fig. 1 shows that both $\mathcal{B}_{loose}$ (Alg. 1) and $\mathcal{B}_{tight}$ (Alg. 2) strictly bounds the absolute error.

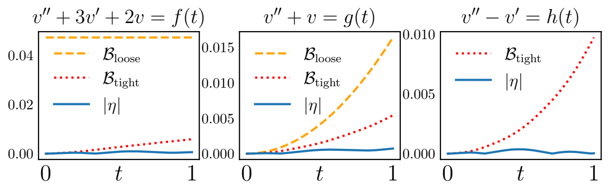


Figure 1: Loose bound (Alg. 1) and tight bound (Alg. 2) for 3 second-order linear ODE with constant coefficients. Notice that the loose bound cannot be applied to the third equation since it has characteristic roots with positive real part.

7.2 LINEAR ODE SYSTEM WITH CONSTANT COEFFICIENTS

In this subsection, we train 6 networks to solve a 6-dimensional linear system of ODEs with constant coefficients, namely, $\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v} + A\mathbf{v} = \mathbf{f}$. We pick $A = PJP^{-1}$ where $J = \begin{pmatrix} J_1 & & \\ & J_2 & \\ & & J_3 \end{pmatrix}$ with $J_1 = \begin{pmatrix} 4 & 1 & \\ & 4 & 1 \\ & & 4 \end{pmatrix}$, $J_2 = \begin{pmatrix} 3 & 1 \\ & 3 \end{pmatrix}$, $J_3 = 2$, and P is a random orthogonal matrix.

We pick the initial conditions to be $\mathbf{v}(0) = P(0\,0\,1\,0\,1\,1)^T$ and the forcing function to be $\mathbf{f}(t) = P(\cos t + 4\sin t + \ln(1+t),\ \frac{1}{1+t} + 4\ln(1+t) + (t+1),\ 4t + 5,\ 2t + 3t^2 + e^t,\ 4e^t,\ 2\cos t - \sin t)^T$, so that the manufactured exact solution is $\mathbf{v}(t) = P(\sin t,\ \ln(t+1),\ t+1,\ t^2,\ e^t,\ \cos t)^T$.

After obtaining the residual information $\mathbf{r}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}(t) + A\mathbf{u}(t) - \mathbf{f}(t)$, we apply Alg. 3 to obtain componentwise bound and norm bound of $\boldsymbol{\eta} = \mathbf{u} - \mathbf{v}$. It is shown in Fig. 2 that the bounds hold over the domain.

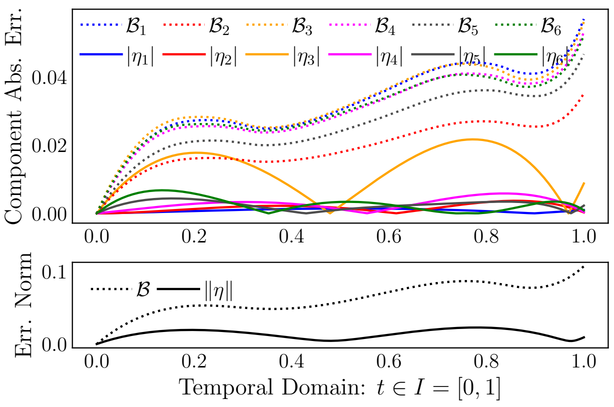


Figure 2: *Componentwise* bound (upper) and *norm* bound (lower) for linear ODE system with constant coefficients

7.3 NONLINEAR ODE – DUFFING EQUATION

In this subsection, we consider a Duffing oscillator, which is characterized by the following 2nd order nonlinear ODE:

$$\frac{\mathrm{d}^2 v}{\mathrm{d}t^2} + 3\frac{\mathrm{d}v}{\mathrm{d}t} + 2v + \varepsilon v^3 = \cos t, \tag{32}$$

under initial conditions $v(0) = 1$ and $v'(0) = 1$, where ε controls the nonlinearity of the equation. Using Alg. 4, we solve the equation on $I = [0, 2]$ for linspaced $\varepsilon \in (-0.9, 0.9)$ using neural networks and bound the errors. The input J to Alg. 4 is chosen to be 6. Namely, we expand the solution and bound components from degree 0 to 6.

The analytical solution to Eq. 32 is complicated. Hence, we use the RKF4(5) method to compute numerical solutions that are close enough to exact solutions for visualization purposes. See Fig. 3 for network solutions against RKF4(5) solutions and Fig. 4 for error bounds against absolute error.

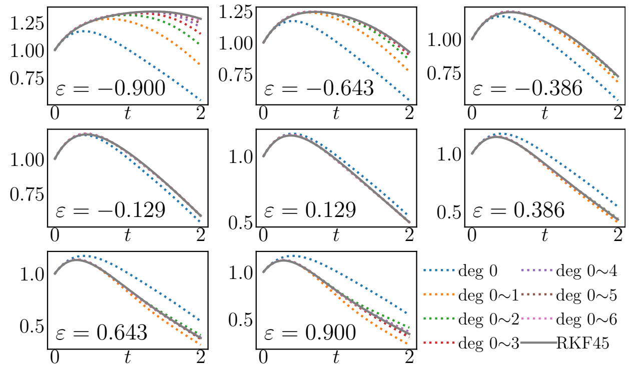


Figure 3: RKF45 and Network Solutions (max-degree 0∼6) to Duffing equation 32 for $\varepsilon \in (-0.9, 0.9)$

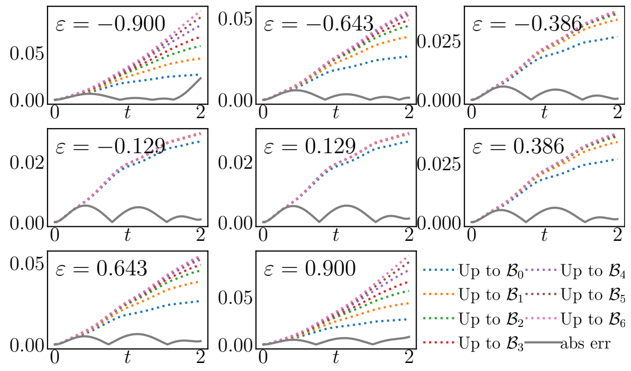
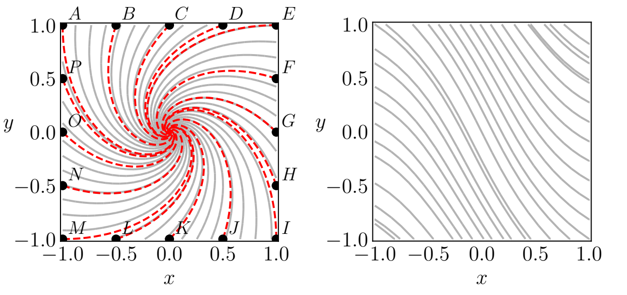


Figure 4: True Error vs. error bound (max-degree 0∼6) of neural network solution to Duffing Equation 32 for $\varepsilon \in (-0.9, 0.9)$

7.4 LINEAR PDE SYSTEM WITH NONCONSTANT COEFFICIENTS



(a) $\mathcal{C}: \begin{cases} x' = -x - y \\ y' = x - y \end{cases}$ (b) $\mathcal{C}: \begin{cases} x' = x^2 + y^2 + 1 \\ y' = x^2 - y^2 + 2 \end{cases}$

Figure 5: Characteristics curves of Eq. 33 (left) and Eq. 34 (right). The red curves, with staring points A to P, are selected for visualization of absolute error and error bound in Fig. 6.

7.4.1 PDE Error Bound Evaluation Using Alg. 6

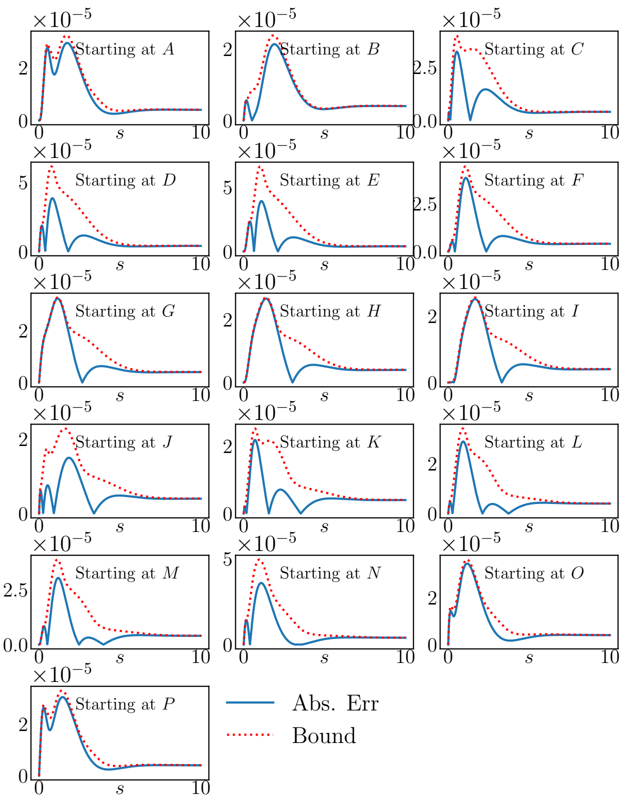


Figure 6: Absolute error and error bound on selected characteristic curves. These characteristic curve start at points A through P as shown in Fig. 5a. The blue solid curves are absolute error along the characteristic curves and red dotted curves are corresponding bounds.

We try to solve the following first-order linear PDE,

$$(-x-y)\frac{\partial v}{\partial x}+(x-y)\frac{\partial v}{\partial y}+v=3x-2y \qquad (33)$$

in spatial domain $\Omega=[-1,1]^2$. The boundary constraints are $v(x,\pm 1)=2x\pm 3$ and $v(\pm 1,y)=3y\pm 2$. The manufactured solution is given by $v(x,y)=2x+3y$. The characteristic curves are integral curves $\mathcal{C}:\begin{cases}x'(s)=-x-y\\ y'(s)=x-y\end{cases}$, or $\mathcal{C}:\begin{cases}x(s)=R_0e^{-s}\cos(s+\theta_0)\\ y(s)=R_0e^{-s}\sin(s+\theta_0)\end{cases}$, where $R_0=\sqrt{x_0^2+y_0^2}$ and $\theta_0=\operatorname{atan2}(y_0,x_0)$ are constants determined by the starting point $(x_0,y_0)\in\Gamma=\partial\Omega$. See Figure 5 for visualization.

Since the analytical expression of the characteristic curves is known, Alg. 6 can be applied to evaluate the bound on each curve. We choose 16 characteristic curves with starting points A, B, …, P, equidistantly placed on the boundary (Fig. 5a). We plot the absolute error and the computed error bound along these characteristic curves in Fig. 6. It can be seen that absolute error lies strictly within the bounds.

7.4.2 PDE Error Bound Evaluation Using Alg. 5

Consider the following PDE

$$(x^2+y^2+1)\frac{\partial v}{\partial x}+(x^2-y^2+2)\frac{\partial v}{\partial y}+(3-2x)v=f \quad (34)$$

over domain $\Omega=[-1,1]^2$, where $f(x,y)=6-4x$. The boundary constraints are $v(-1,y)=2$ and $v(x,1)=2$, and the manufactured solution is $v(x,y)=2$.

The characteristic curves $\mathcal{C}:\begin{cases}x'(s)=x^2+y^2+1\\ y'(s)=x^2-y^2+2\end{cases}$ are given by a nonlinear ODE, which is hard to solve analytically. (See Fig. 5b for visualization) Therefore, Alg. 6 cannot be applied to evaluate the error bound.

However, the coefficient $(3-2x)$ is nonzero over domain Ω. Hence, we can use Alg. 5 to compute a constant error bound $|\eta(x,y)|\le\mathcal{B}(x,y)\equiv B$ for all $(x,y)\in\Omega$. We visualize the bound and the maximum absolute error $\max_{(x,y)\in\Omega}|\eta|$ after each training epoch in Fig. 7. As expected, the bound is loose, which is about an order of magnitude larger than the max absolute error. Yet, it consistently holds true for every epoch, even during the early stages of training, when the network performs poorly.

8 CONCLUSION AND FUTURE WORK

This paper proposes various error-bounding algorithms for any PINN solution to certain classes of ODEs and PDEs. These algorithms only require the residual information $r(\cdot)$ and the equation structure $\mathcal{D}v=f$ as input. There are many

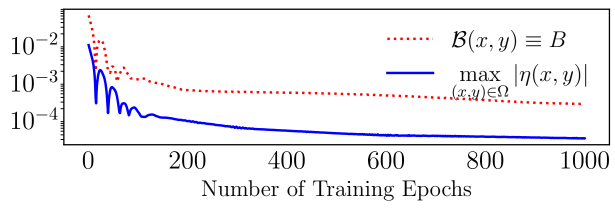


Figure 7: Constant bound B, computed using Alg. 5, and max absolute error over domain at different epochs of training.

real-world applications for which the exact solution $v(\cdot)$ is unknown or hard to compute. However, the residual information $r(\cdot)$ is usually, if not always, readily available. With our proposed algorithms, PINNs can be trained until the error is gauranteed to fall below a specified tolerance threshold. The mathematical relationship between residual and error bound also sheds light on optimizing PINN solutions for future studies.

The error-bounding algorithms proposed in this paper only apply to certain classes of ODEs and PDEs. However, the insights of this paper can be beneficial to future work that extends to more general classes of ODEs and PDEs, especially nonlinear ones. We also plan to apply these algorithms stochastic differential equations, where the error bound is a probabilistic tail bound.

References

Feiyu Chen, David Sondak, Pavlos Protopapas, Marios Mattheakis, Shuheng Liu, Devansh Agarwal, and Marco Di Giovanni. Neurodiffeq: A python package for solving differential equations with neural networks. *Journal of Open Source Software*, 5(46):1931, 2020.

Tim De Ryck and Siddhartha Mishra. Error analysis for deep neural network approximations of parametric hyperbolic conservation laws. *arXiv preprint arXiv:2207.07362*, 2022a.

Tim De Ryck and Siddhartha Mishra. Error analysis for physics-informed neural networks (pinns) approximating kolmogorov pdes. *Advances in Computational Mathematics*, 48(6):1–40, 2022b.

Tim De Ryck and Siddhartha Mishra. Generic bounds on the approximation error for physics-informed (and) operator learning. *arXiv preprint arXiv:2205.11393*, 2022c.

Shaan Desai, Marios Mattheakis, Hayden Joy, Pavlos Protopapas, and Stephen Roberts. One-shot transfer learning of physics-informed neural networks. 2021.

Cedric Flamant, Pavlos Protopapas, and David Sondak. Solving differential equations using neural network solution bundles. *arXiv preprint arXiv:2006.14372*, 2020.

Olga Graf, Pablo Flores, Pavlos Protopapas, and Karim Pichara. Uncertainty quantification in neural differential equations. *arXiv preprint arXiv:2111.04207*, 2021.

Mengwu Guo and Ehsan Haghighat. Energy-based error bound of physics-informed neural network solutions in elasticity. *Journal of Engineering Mechanics*, 148(8): 04022038, 2022.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Michael A Jenkins and Joseph F Traub. A three-stage variable-shift iteration for polynomial zeros and its relation to generalized rayleigh iteration. *Numerische Mathematik*, 14(3):252–263, 1970.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Pola Lydia Lagari, Lefteri H Tsoukalas, Salar Safarkhani, and Isaac E Lagaris. Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions. *International Journal on Artificial Intelligence Tools*, 29(05):2050009, 2020.

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

Isaac E Lagaris, Aristidis C Likas, and Dimitris G Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

Marios Mattheakis, David Sondak, Akshunna S Dogra, and Pavlos Protopapas. Hamiltonian neural networks for solving differential equations. *arXiv preprint arXiv:2001.11107*, 2020.

Kevin Stanley McFall and James Robert Mahan. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, 20 (8):1221–1233, 2009.

Kshitij Parwani and Pavlos Protopapas. Adversarial sampling for solving differential equations with neural networks. *arXiv preprint arXiv:2111.12024*, 2021.

Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

N Sukumar and Ankit Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *arXiv preprint arXiv:2104.08426*, 2021.

Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *arXiv preprint arXiv:1710.00211*, 2017.