Implementing Physics Informed Neural Networks with PyTorch

Introduction

Physics-Informed Neural Networks (PINNs) represent a novel scientific machine learning framework designed to solve problems governed by Partial Differential Equations (PDEs). By leveraging neural networks, PINNs approximate PDE solutions through the optimization of a loss function that incorporates initial and boundary conditions as well as the PDE residuals at specified points in the domain, known as collocation points. Unlike traditional methods that rely on discretizing the domain or fitting to labeled data, PINNs embed the governing physics directly into the neural network architecture. This integration transforms the problem of solving PDEs into one of optimizing a loss function, enabling a mesh-free, data-efficient approach.

The innovation of PINNs lies in their ability to encode the underlying physical laws within the network itself. This unsupervised training paradigm eliminates the need for precomputed datasets from simulations or experiments, instead utilizing the equations governing the system to guide the learning process. The framework was introduced in 2017 by Raissi et al. and later consolidated in their seminal 2019 paper, which demonstrated PINNs' applicability to both forward problems—estimating PDE solutions—and inverse problems, such as inferring unknown parameters or boundary conditions from observational data. Examples of PDEs solved using PINNs include the Schrödinger, Burgers, and Allen-Cahn equations.

The concept of embedding prior knowledge into neural networks is not entirely new. Early efforts, such as the constrained neural networks proposed by Psichogios and Ungar (1992) and the work by Lagaris et al. (1998, 2000), laid the foundation for this field. These methods approximated PDE solutions using neural networks but were often limited to regular geometries. PINNs extend these ideas to more complex domains and higher-dimensional problems, made feasible by advances in deep learning, such as improved training techniques, hardware acceleration, and the advent of automatic differentiation.

PINNs have inspired a variety of extensions and related methodologies, including the Deep Ritz Method (DRM), Deep Galerkin Methods (DGM), and variants like hp-VPINNs and conservative PINNs (CPINNs). While these frameworks share similarities, PINNs distinguish themselves by embedding both PDE residuals and initial/boundary conditions into the loss function, enabling seamless handling of diverse physical problems. PINNs have also been adapted to solve stochastic differential equations (SDEs) and integrodifferential equations (IDEs), further broadening their applicability.

The versatility and robustness of PINNs offer several advantages over conventional methods. They provide differentiable solutions with analytical gradients, facilitate on-demand solution computation, and unify the treatment of forward and inverse problems within the same optimization framework. Additionally, PINNs excel in tackling high-dimensional problems, irregular geometries, and domains that are computationally challenging for traditional numerical methods.

This project explores the application of PINNs to solve PDEs, focusing on reproducing key results from the landmark paper by Raissi et al. (2019) using PyTorch. By doing so, the project aims to highlight the strengths and limitations of PINNs as a modern computational tool for scientific and engineering challenges.

Data-driven solutions of PDEs:

The problem statement that we consider is that given the exact PDE including the initial and boundary conditions, can the neural network infer the solution to the PDE?

We consider partial differential equations of the following general form

$$\frac{\partial u}{\partial t} + \mathcal{N}[u] = 0$$

Where $u(t, x)$ is the solution to be approximated by the neural network and $\mathcal{N}[\cdot]$ is the non-linear operator specific to the equation.

Continuous Time Models:

As an example of continuous time models, we study the non-linear one-dimensional Schrodinger equation:

$$i\frac{\partial h}{\partial t} + 0.5\frac{\partial^2 h}{\partial x^2} + |h|^2 h = 0 \quad x \in [-5,5], \quad t \in [0, \pi/2]$$

With the initial condition $h(0, x) = 2\,sech\,x$ and periodic boundary conditions:

$$h(t, -5) = h(t, 5), \quad \mathrm{h_x}(\mathrm{t}, -5) = \mathrm{h_x}(\mathrm{t}, 5)$$

Further on, we can define

$$f(t, x) \equiv i\frac{\partial h}{\partial t} + 0.5\frac{\partial^2 h}{\partial x^2} + |h|^2 h$$

The shared parameters of the neural networks $h(t, x)$ and $f(t, x)$ can be learned by minimizing the mean squared error loss:

$$MSE = MSE_0 + MSE_b + MSE_f$$

Where MSE_0 corresponds to the loss on the initial data, MSE_b enforces the periodic boundary conditions and MSE_f penalizes the Schrodinger equation not being satisfied on the collocation points.

To evaluate the accuracy of the proposed method, the authors simulated a high-resolution dataset for a PDE using spectral methods. The solution was computed up to $t = \pi/2$ using the Chebfun package with a spectral Fourier discretization (256 modes) and a fourth-order Runge-Kutta integrator with a time-step of $\Delta t = \pi/2 \cdot 10^{-6}$. For the data-driven approach, the training set included $N_0 = 50$ randomly sampled data points from the high-resolution dataset at $t = 0$ and $N_b = 50$ randomly selected collocation points to enforce periodic boundary conditions. Additionally, $N_f = 20{,}000$ collocation points were randomly sampled to enforce the PDE within the solution domain. Sampling locations were generated using a Latin Hypercube Sampling strategy to ensure a space-filling distribution.

The results obtained by implementing PINN using PyTorch can be summarized by the following plots:

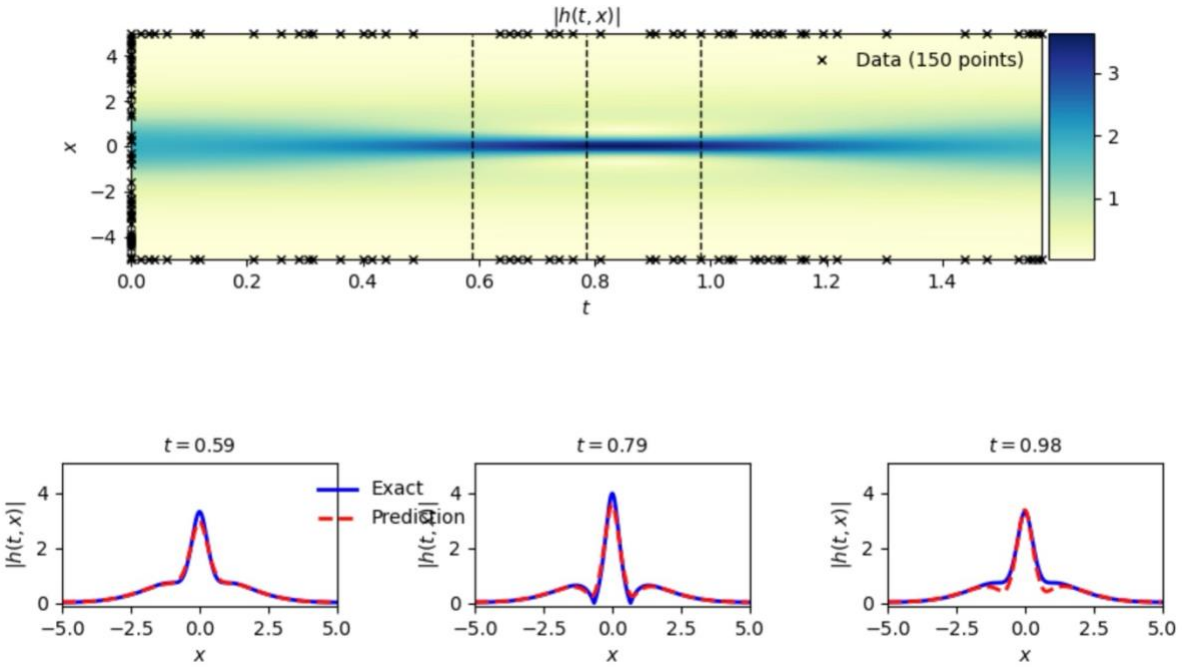

Figure 1: Top: Predicted solution |h(t,x)| along with the initial and boundary training data. Bottom: Comparison of the exact and predicted solutions corresponding to the three time snapshots depicted by dashed lines in the top panel.

Discrete Time Models:

The Runge-Kutta method is used to break down every time step into q-stages. Starting with the solution u^n at some time t^n, the Runge-Kutta scheme is used as follows

$$u^{n+c_i} = u^n - \Delta t\, a_{ij} \mathcal{N}[u^{n+c_j}] \quad i = 1, \dots, q$$

$$u^{n+1} = u^n - \Delta t b_j \mathcal{N}[u^{n+c_j}]$$

Where u^{n+1} is the solution at some later time step t^{n+1}.

Our equation of choice is the Allen-Cahn equation:

$$\frac{\partial u}{\partial t} - 0.0001 \frac{\partial^2 u}{\partial x^2} + 5u^3 - 5u = 0 \quad x \in [-1,1], \quad t \in [0,1]$$

With the initial boundary condition $u(0,x) = x^2\, cos(\pi x)$

And the periodic boundary conditions:

$$u(t,-1) = u(t,1) \quad u_x(t,-1) = u_x(t,1)$$

The non-linear operator in the Runge-Kutta scheme takes the following form

$$\mathcal{N}[u^{n+c_j}] \;=\; -0.0001 \frac{\partial^2 u^{n+c_j}}{\partial x^2} + 5(u^{n+c_j})^3 - 5u^{n+c_j}$$

Now we minimize the sum of squared errors

$$SSE = SSE_n + SSE_b$$

Where SSE_n minimizes the difference between u^n approximated by the neural network and that generated by the Runge-Kutta scheme (remember that the Runge-Kutta scheme simply evolves the solution dictated by the Allen-Cahn equation). SSE_b imposes the boundary condition on the solution.

The training and test datasets for this study were generated by simulating the Allen-Cahn equation using spectral methods. Starting with the initial conditions and periodic boundary conditions, the equation was integrated up to $t = 0.1$ using the Chebfun package. The simulation employed a spectral Fourier discretization with 512 modes and a fourth-order explicit Runge-Kutta integrator with a time-step of $\Delta t = 10^{-5}$.

The training dataset consisted of $N_n = 200$ data points randomly sampled from the exact solution at $t = 0.1$. The objective was to predict the solution at $t = 0.9$ using a single

time-step of size $\Delta t = 0.8$. A discrete-time physics-informed neural network (PINN) was employed, featuring four hidden layers with 200 neurons per layer. The network's output layer predicted 101 quantities corresponding to 100 Runge-Kutta stages $u^{n+c_i}(x), i = 1, \ldots, q$ and the solution at the final time $u^{n+1}(x)$.

The results that we obtain using PyTorch are as follows:

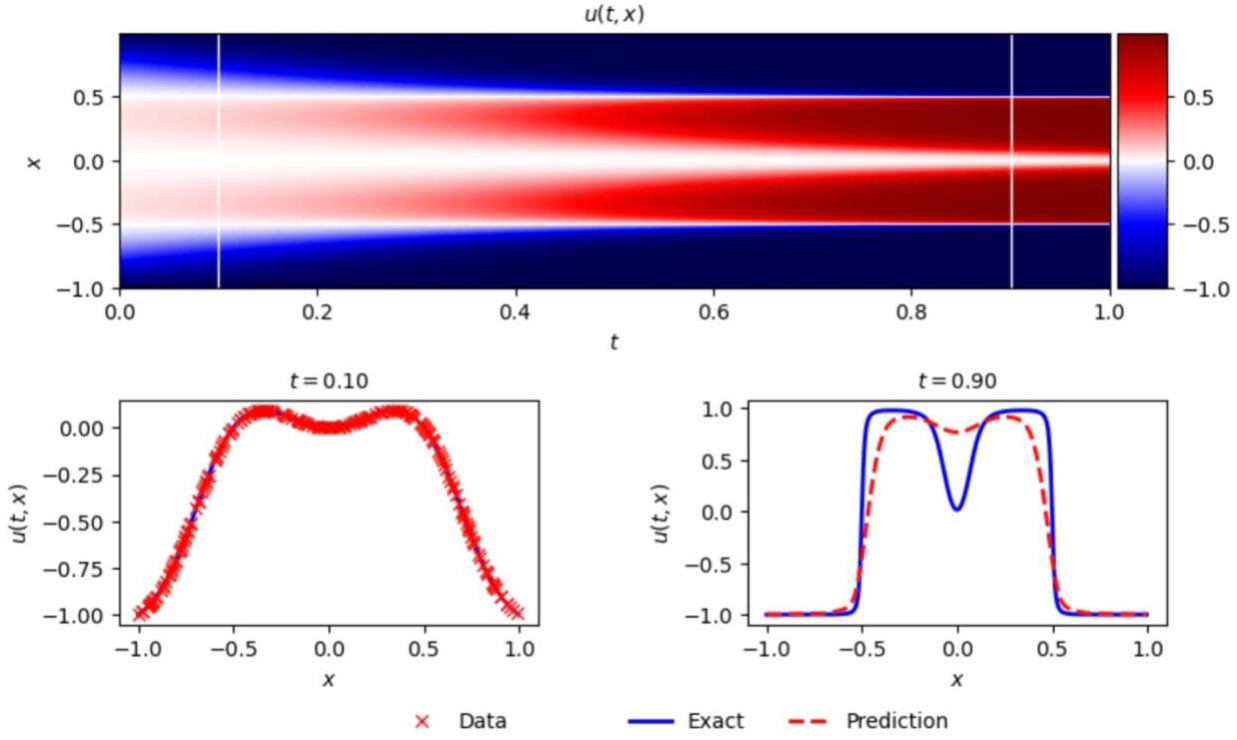

Figure 2: Allen-Cahn equation: Top: Solution u(t,x) along with the location of the initial training snapshot at t = 0.1 and the final prediction snapshot at t = 0.9. Bottom: Initial training data and final prediction at the snapshots depicted by the white vertical lines in the top panel.

Data-driven Discovery of PDEs:

Now the problem statement is that given the general form of the PDE, upto some parameters λ, what are the optimum values for λ that best describes the observed data?

Continuous Time Models

We study the famous Navier-Stokes equation in two dimensions:

$$\frac{\partial u}{\partial t} + \lambda_1 \left(u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} \right) = -\frac{\partial p}{\partial x} + \lambda_2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

$$\frac{\partial \mathrm{v}}{\partial \mathrm{t}} + \lambda_1 \left(u \frac{\partial \mathrm{v}}{\partial \mathrm{x}} + v \frac{\partial \mathrm{v}}{\partial y} \right) = -\frac{\partial \mathrm{p}}{\partial y} + \lambda_2 \left(\frac{\partial^2 \mathrm{v}}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{v}}{\partial y^2} \right)$$

where $u(t,x,y)$ denotes the x-component and $v(t,x,y)$ denotes the y-component of the velocity field and $p(t,x,y)$ is the pressure field. Here (λ_1, λ_2) are the parameters to be determined by the neural network.

Once again, we define $f(t,x,y)$ and $g(t,x,y)$ as

$$f \equiv \frac{\partial u}{\partial t} + \lambda_1 \left(u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} \right) + \frac{\partial p}{\partial x} - \lambda_2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

$$g \equiv \frac{\partial v}{\partial t} + \lambda_1 \left(u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} \right) + \frac{\partial p}{\partial y} - \lambda_2 \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right)$$

And minimize the mean squared error loss

$$MSE = MSE_u + MSE_v + MSE_f + MSE_g$$

Where MSE_u minimizes the squared error bwteen the predicted value of u and the exact value (similarly for MSE_v). MSE_f imposes that the Navier-Stokes equation for the x-component velocity field is satisfied (similarly for the y-component MSE_g).

This study considers the problem of incompressible flow past a circular cylinder, which exhibits dynamic behaviors and transitions depending on the Reynolds number $Re = u_\infty D/\nu$. With non-dimensional parameters free stream velocity $u_\infty = 1$, cylinder diameter $D = 1$ and kinematic viscosity $\nu = 0.01$, the system demonstrates a periodic steady-state characterized by the asymmetric vortex shedding pattern known as the Kármán vortex street.

High-resolution data for this problem was generated using the spectral/hp-element solver NekTar. The domain was discretized with 412 triangular elements, each using a tenth-order Jacobi polynomial expansion. Boundary conditions included a uniform velocity profile at the inlet, zero pressure at the outlet 25 diameters downstream, and periodicity along the top and bottom boundaries of the $[-15,25] \times [-8,8]$ domain. The system was integrated using a third order stiffly stable scheme until reaching a periodic steady state.

For model training, a small subset of the steady-state data was sampled from a rectangular region downstream of the cylinder. The training dataset consisted of $N = 5000$ samples, representing only 1% of the available data, chosen to demonstrate the method's ability to learn from sparse and scattered data. The goal was to identify unknown parameters λ_1 and λ_2, and reconstruct the pressure field $p(t,x,y)$ which is defined up to a constant. The predictions of velocity components $u(t,x,y)$ and $v(t,x,y)$ after training were qualitatively accurate.

The employed neural network consisted of 9 layers with 20 neurons per layer, effectively leveraging the limited and scattered data to reconstruct key flow characteristics and validate the model's predictions.

The results can be summarized by the following plots:

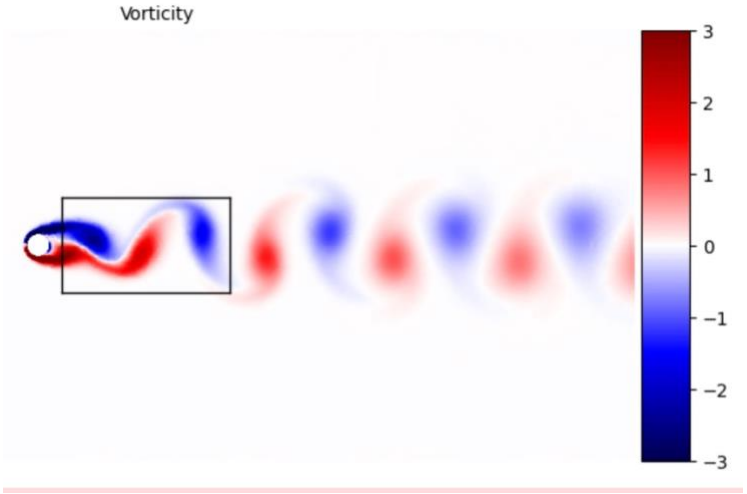

Figure 3: Navier-Stokes Equation: Incompressible flow and dynamic vortex shedding past a circular cylinder at Re = 100. The spatio-temporal data correspond to the depicted rectangular region in the cylinder wake.

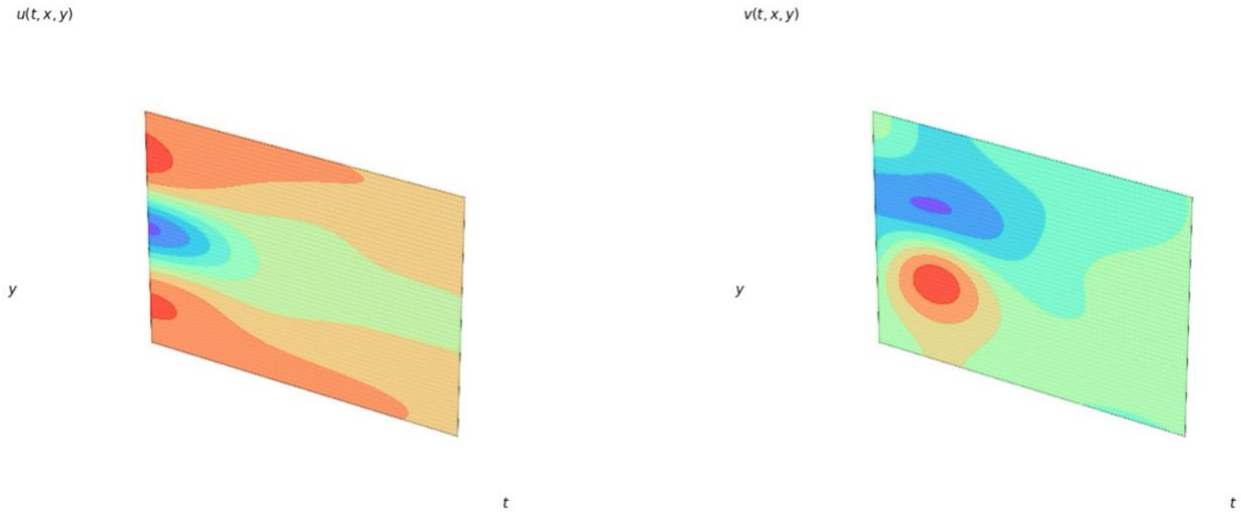

Figure 4: Locations of training datapoints for the longitudinal and transverse velocity components u(t,x,y) and v(t,x,y) respectively.

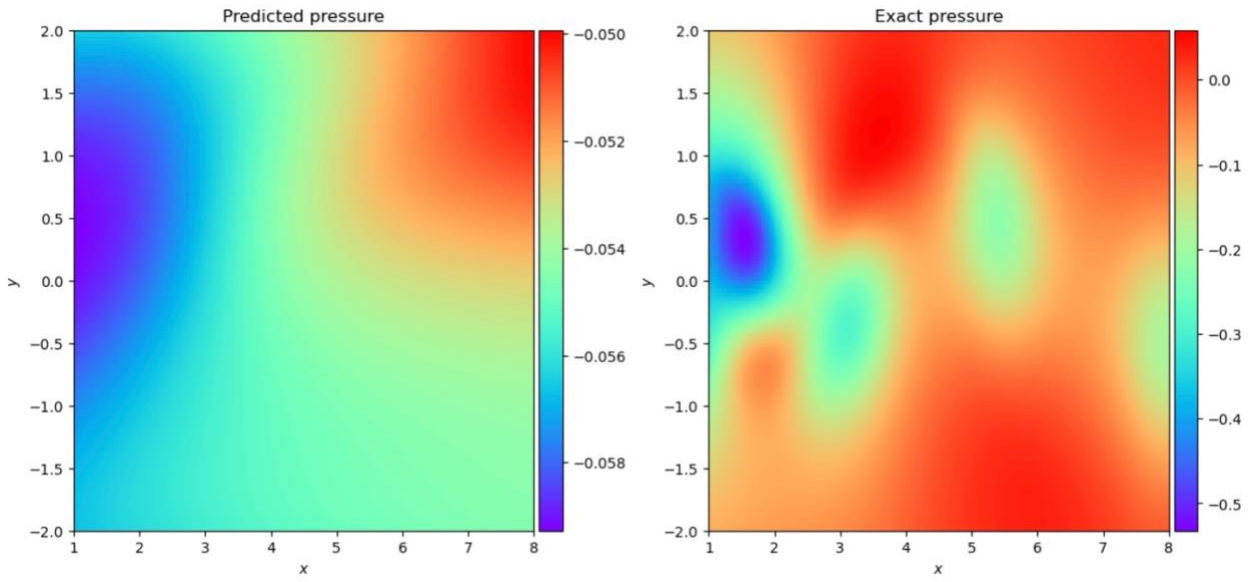

Figure 5: Navier-Stokes equation. Predicted vs exact instantaneous pressure field p(t,x,y) at a particular time.

Discrete-Time Models:

We study the Kortewedge-Vries (KdV) equation which is generally used to mathematically model shallow water waves. It can be written as

$$\frac{\partial u}{\partial t} + \lambda_1 u \frac{\partial u}{\partial x} + \lambda_2 \frac{\partial^3 u}{\partial x^3} = 0$$

With (λ_1, λ_2) being the unknown parameters to be learned by the neural network. Once more, we will employ the Runge-Kutta integration scheme to break up each time step into q stages. The corresponding non-linear operator is

$$\mathcal{N}[u^{n+c_j}] = \lambda_1 u^{n+c_j} \frac{\partial u^{n+c_j}}{\partial x} + \lambda_2 \frac{\partial^3 u^{n+c_j}}{\partial x^3}$$

And the KdV equation can be learned by minimizing the sum of squared errors.

To generate training and test datasets, the KdV equation was simulated using spectral methods. Starting with the initial condition and periodic boundary conditions, the equation was integrated up to $t = 0.1$using the Chebfun package. The simulation employed a spectral Fourier discretization with 512 modes and a fourth-order explicit Runge-Kutta integrator with a time-step of $\Delta t = 10^{-6}$.

From this dataset, two solution snapshots were extracted at $t_n = 0.2$ and $t_{n+1} = 0.8$. These were randomly sub-sampled to create a training dataset with $N_n = 199$ and $N_{n+1} = 201$ points. A discrete-time physics-informed neural network (PINN) was trained on this data by minimizing the sum of squared errors using the L-BFGS optimizer.

The neural network architecture consisted of 4 hidden layers with 50 neurons per layer. The output layer predicted the solution at q Runge-Kutta stages $u^{n+c_j}(x), j = 1, \dots, q$, where q was chosen empirically to achieve temporal error accumulation at the level of machine precision.

Results: The predicted values are $(\lambda_1, \lambda_2) = (0.994, 0.0024)$ which when compared with the exact values $(1, 0.0025)$ yields an error of 0.57% for λ_1 and 3.99% for λ_2 which falls within an acceptable margin.

Conclusion:

Physics-Informed Neural Networks (PINNs) have emerged as a powerful framework for solving partial differential equations (PDEs) by leveraging the representational capabilities of deep neural networks (DNNs) while embedding the governing physical laws into their architecture. This project explored the foundational aspects of PINNs, including their architecture, implementation, and advancements, along with variations and extensions such as Bayesian PINNs, convolutional PINNs, and generative adversarial network-based PINNs (PI-GANs).

The universal approximation theorem highlights the remarkable representational power of neural networks, which can approximate any continuous function with sufficient depth and neurons. However, the success of PINNs depends on careful design choices, including network depth, number of neurons, activation functions, and optimization strategies. Challenges such as training efficiency, overparameterization, and the scalability of PINNs to high-dimensional problems remain areas of active research.

This project demonstrated the versatility of PINNs in handling diverse tasks, such as solving forward and inverse problems, and highlighted their ability to integrate seamlessly with other machine learning paradigms. By leveraging various architectures—ranging from fully connected feedforward networks to convolutional and recurrent networks—PINNs can address complex problems involving irregular geometries, time-dependence, and stochastic dynamics.

In conclusion, PINNs represent a transformative approach to solving PDEs by combining data-driven methods with fundamental physics. The insights gained from this project underscore the potential of PINNs to address challenges in scientific computing, engineering, and beyond, while also pointing to opportunities for further research into optimizing neural network architectures and extending their applicability to broader classes of problems.

References:

1. Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). 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.
2. Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017c). Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv preprint* arXiv:1711.10561.
3. Psichogios, D. C., & Ungar, L. H. (1992). A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38(10), 1499–1511.
4. Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5), 987–1000.
5. Lagaris, I. E., Likas, A., & Fotiadis, D. I. (2000). Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5), 1041–1049.
6. E, W., & Yu, B. (2018). The Deep Ritz Method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1), 1–12.
7. Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375, 1339–1364.
8. Kharazmi, E., Zhang, Z., & Karniadakis, G. E. (2021). hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374, 113547.
9. Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6), 422–440.
10. Yang, L., Zhang, L., & Karniadakis, G. E. (2021). B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425, 109913.
11. Zhu, Y., Zabaras, N., Koutsourelakis, P.-S., & Perdikaris, P. (2019). Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394, 56–81.

12. Gao, H., Sun, L., & Wang, J.-X. (2021). PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving PDEs on irregular domains. *Journal of Computational Physics*, 428, 110079.

13. Fang, Z. (2021). Neural network meets finite volume method: A physics-informed convolutional neural network framework for PDE solving. arXiv:2104.06258.

14. Jagtap, A. D., Kharazmi, E., & Karniadakis, G. E. (2020). Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365, 113028.

15. Mishra, S., & Molinaro, R. (2021). Estimates on the generalization error of physics-informed neural networks for approximating PDEs. *Mathematics of Computation*, 90(332), 2771–2799.