Mini Project Report
PINN for PDEs

Contents

1 Physics Informed Neural Networks

1.1 History

Physics-Informed Neural Networks (PINNs) were introduced by Raissi, Perdikaris, and Karniadakis around 2017–2019 [1]. They embed physical laws, typically differential equations, into the loss function of neural networks, enabling models to respect known physics even with limited data. This innovation marked a major step towards scientific machine learning.

Before PINNs, neural networks were mostly data-driven and lacked physical interpretability. PINNs combined machine learning with domain knowledge, and have since been applied across fields like fluid dynamics, heat transfer, and biological systems. Key developments include adaptive sampling [2] and domain decomposition methods [3].

PINNs now represent a vital research area linking AI and physics-based modeling.

1.2 Introduction to PINNs

Physics-informed neural networks (PINNs) leverage recent developments in automatic differentiation to compute derivatives of neural networks with respect to their input coordinates and model parameters. These networks are constrained to respect symmetries, invariances, or conservation principles originating from the physical laws governing the observed data, as modeled by general time-dependent and nonlinear partial differential equations (PDEs).

The general form of parametrized nonlinear PDEs considered is:

$$u_t + \mathcal{N}[u; \lambda] = 0, x \in \Omega, t \in [0, T] \tag{1}$$

where $u(t, x)$ denotes the latent solution, $\mathcal{N}[\cdot; \lambda]$ is a nonlinear operator parametrized by λ, and $\Omega \subset \mathbb{R}^D$.

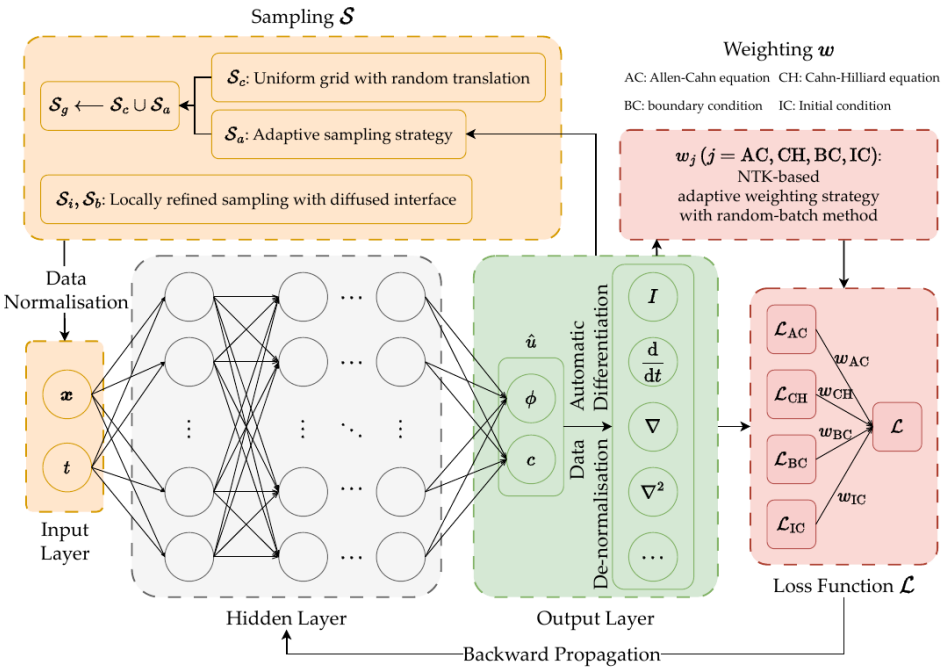


Figure 1: Physics-Informed Neural Network architecture

1.3 The Intuition behind the Algorithm

Physics-informed neural networks integrate information from both measurements and PDEs by embedding the PDEs into the loss function using automatic differentiation. The workflow involves:

1. Constructing a neural network $u(t, x; \theta)$ with θ as trainable parameters
2. Specifying measurement data $\{t_i, x_i, u_i\}$ and residual points $\{t_j, x_j\}$ for the PDE
3. Defining a loss function combining data mismatch and PDE residual terms
4. Training the network to find optimal parameters by minimizing the loss

The key innovation is using automatic differentiation to compute derivatives of the neural network output with respect to its inputs, enabling direct encoding of physical laws.

2 Data-Driven Solution of Partial Differential Equations

Let us start by concentrating on the problem of computing data-driven solutions to partial differential equations.

2.1 Continuous Time Models

For continuous time models, we define $f(t, x)$ as:

$$f := u_t + \mathcal{N}[u]$$

and approximate $u(t, x)$ with a deep neural network, resulting in a physics-informed neural network $f(t, x)$. The shared parameters are learned by minimizing:

$$MSE = MSE_u + MSE_f$$

where N_u refers to the number of boundary points, N_f refers to the number of collocation points, MSE_u corresponds to initial/boundary data and MSE_f enforces the PDE structure.

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2, \tag{2}$$

and

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2. \tag{3}$$

$$Rate = \frac{\log\left(\frac{MSE[N_{fi}]}{MSE[N_{fi+1}]}\right)}{\log\left(\frac{N_{f_{i+1}}}{N_{f_i}}\right)} \tag{4}$$

2.1.1 Burgers' Equation

The Burgers equation serves as a prototype for hyperbolic conservation laws, and its solutions can be constructed analytically by defining a function and computing the right-hand side accordingly. In this work, a **Physics-Informed Neural Network (PINN)** is used to approximate solutions, replacing the function $u(t, x)$ with a neural network

representation. By leveraging automatic differentiation, the network satisfies the Burgers equation inherently before training.

During training, the network minimizes the residual loss while maintaining conservation properties similar to numerical methods. The model is trained using the **L-BFGS** optimization algorithm on small datasets, with empirical evidence suggesting convergence given a well-posed PDE. Sensitivity studies confirm the robustness of this approach, with results showing accurate predictions for the Burgers equation.

The network, with **9 layers and 3021 parameters**, achieves a low relative L_2-norm error of 1.2×10^{-3}, outperforming previous Gaussian process-based methods. A systematic evaluation across different training and collocation points further highlights the efficiency of PINNs in solving PDEs with minimal data requirements.

The Burgers' equation in one dimension:

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, x \in [-1, 1], t \in [0, 1] \tag{5}$$

with initial condition $u(0, x) = -\sin(\pi x)$ and boundary conditions $u(t, -1) = u(t, 1) = 0$.

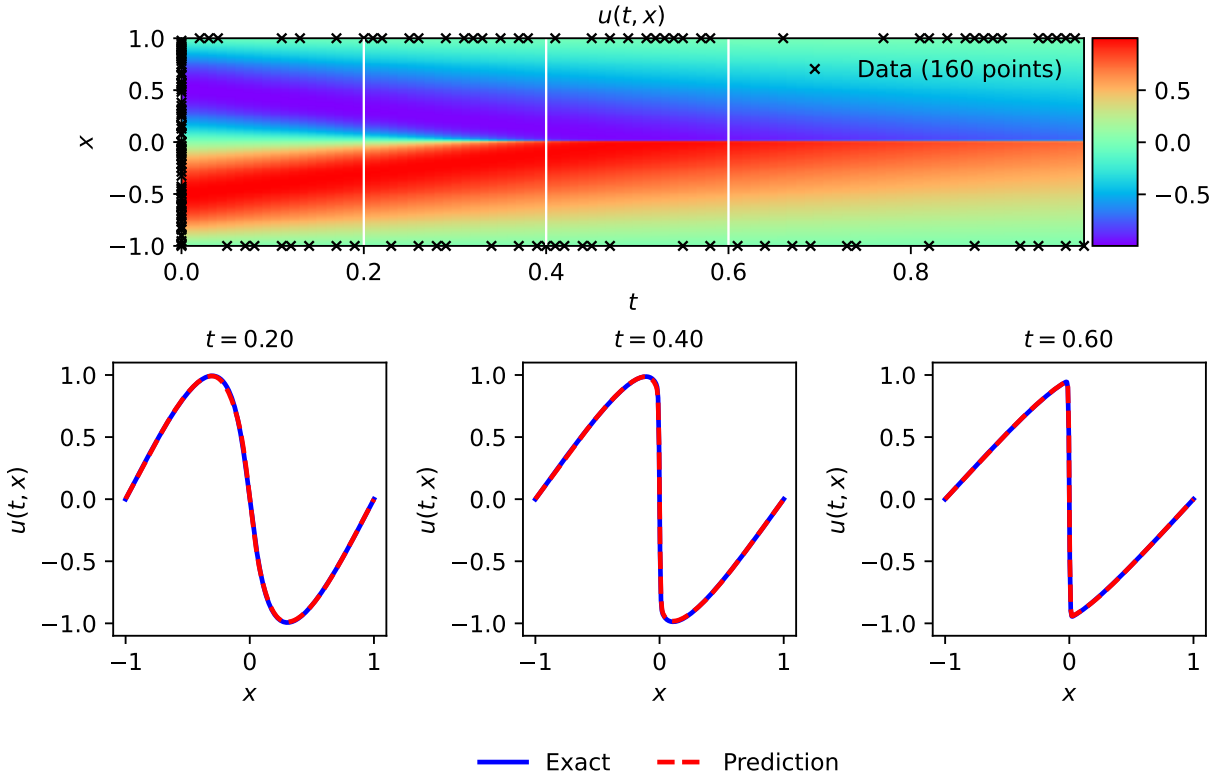


Figure 2: Soluion for equation (5)

Table A.2 shows the resulting relative L2 for different number of hidden layers, and different number of neurons per layer, while the total number of training and collocation points is kept fixed to $N_u = 100$ and N_f= 10,000, respectively. As expected, we observe that as the number of data points increases, the predictive accuracy increases.

	2000	4000	6000	8000	10000
20	0.535667241	0.049695846	0.014588796	0.013886805	0.010786311
40	0.249364823	0.030792063	0.0091363	0.006794268	0.003439679
60	0.185548812	0.022392275	0.008312913	0.00668326	0.005492738
80	0.174256042	0.019572591	0.00463555	0.003494029	0.002582258
100	0.151201725	0.004660192	0.004047269	0.002652773	0.001231156

Table 1: N_f vs N_u graph

	2000–4000	4000–6000	6000–8000	8000–10000
20	5.2225	5.8639	4.2605	0.2210
40	7.3256	5.1587	4.2234	1.3273
60	8.3370	5.2153	3.4444	0.9779
80	8.8426	5.3923	5.0068	1.2669
100	9.3693	8.5816	0.4902	1.8931

Table 2: Rate table for N_f vs N_u

As shown in the table, increasing the values of N_u and N_f leads to a significant decrease in the loss function, signifying higher accuracy in the model's performance.

2.1.2 Korteweg-de Vries (KdV) Equation

We consider a mathematical model of waves on shallow water surfaces; the Korteweg–de Vries (KdV) equation. This equation can also be viewed as Burgers equation with an added dispersive term. The KdV equation has several connections to physical problems. It describes the evolution of long one-dimensional waves in many physical settings. Such physical settings include shallow-water waves with weakly non-linear restoring forces, long in- ternal waves in a density-stratified ocean, ion acoustic waves in a plasma, and acoustic waves on a crystal lattice.

The KdV equation:

$$u_t + \lambda_1 u u_x + \lambda_2 u_{xxx} = 0$$

with initial condition $u(0, x) = \cos(\pi x)$ and periodic boundary conditions.
For the KdV equation, the nonlinear operator in equations (22) is given by

$$\mathcal{N}[u(t, x)] = \lambda_1 \, u \, u_x - \lambda_2 \, u_{xxx} \tag{6}$$

and the shared parameters of the neural networks along with the parameters $\lambda = (\lambda_1, \lambda_2)$ of the KdV equation can be learned by minimizing the sum of squared errors.

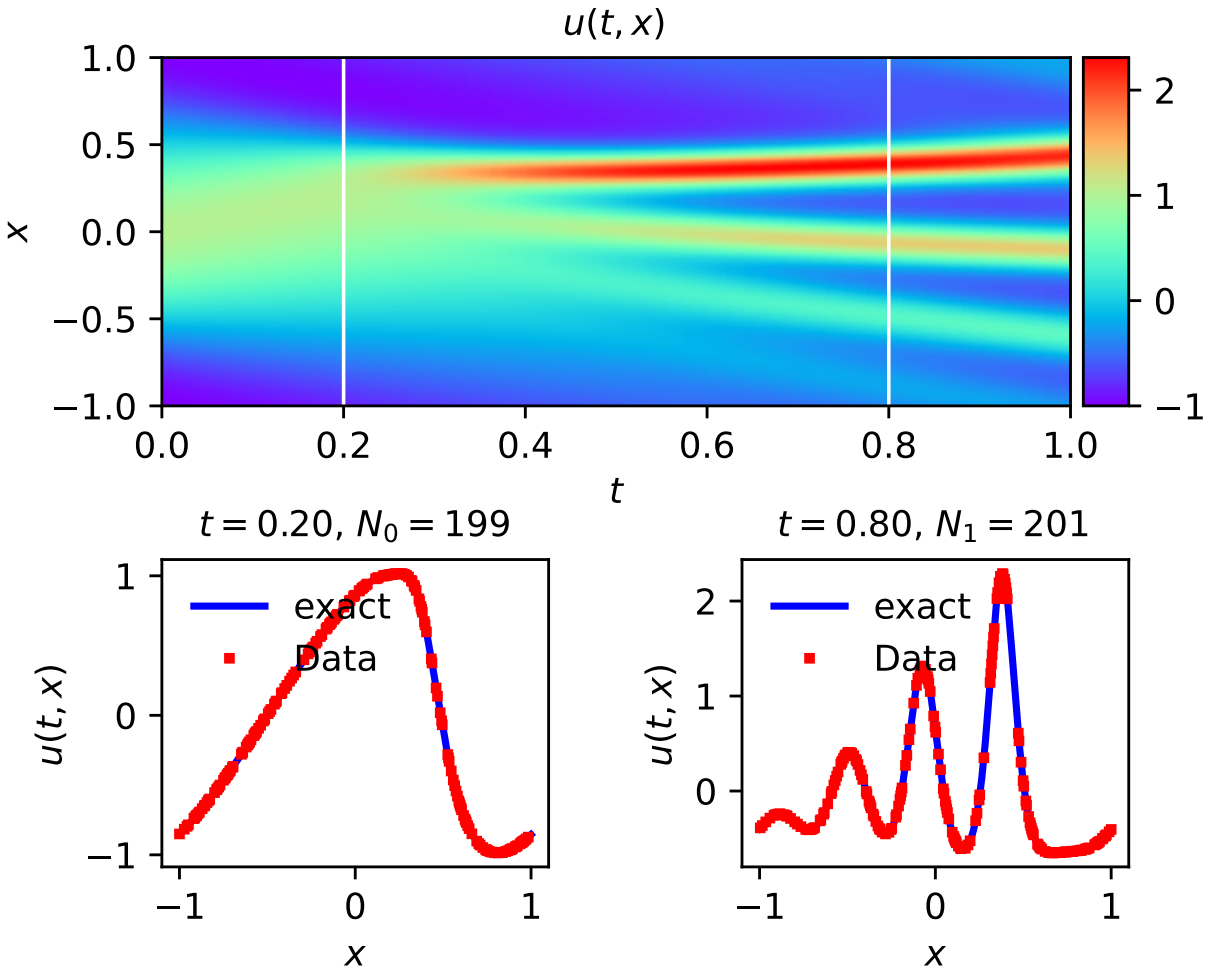


Figure 3: Kdv Equation solution

	2000	4000	6000	8000	10000
20	0.0415	0.0384	0.0323	0.0315	0.0260
40	0.0381	0.0342	0.0241	0.0227	0.0202
60	0.0310	0.0270	0.0232	0.0212	0.0206
80	0.0297	0.0240	0.0230	0.0198	0.0155
100	0.0197	0.0176	0.0150	0.01189	0.01181

Table 3: N_f vs N_u graph

N_u	2000–4000	4000–6000	6000–8000	8000–10000
20	0.1120	0.4266	0.0872	0.8599
40	0.1558	0.8632	0.2080	0.5229
60	0.1993	0.3741	0.3134	0.1287
80	0.3074	0.1050	0.5208	1.0972
100	0.1626	0.3942	0.8077	0.0303

Table 4: Rate table for N_f across N_u

As shown in the table, increasing the values of N_u and N_f leads to a significant decrease in the loss function, signifying higher accuracy in the model's performance.

2.1.3 Cahn-Hilliard equation

we consider the Cahn-Hilliard (CH) equation in a bounded domain $\Omega \in \mathbb{R}^d \quad (d \leq 3)$

$$\frac{\partial u}{\partial t} = \nabla \cdot \left(M_c \nabla \frac{\delta G}{\delta u} \right) = \nabla \cdot (M_c \nabla \mu) \tag{7}$$

$$\mu = \frac{\delta G}{\delta u} = RT \left[\ln u - \ln(1-u) \right] + L(1-2u) - a_c \nabla^2 u \tag{8}$$

$$M_c = \left[\frac{D_A}{RT} u + \frac{D_B}{RT}(1-u) \right] u(1-u) = \frac{D_A}{RT} \left[u + \frac{D_B}{D_A}(1-u) \right] u(1-u) \tag{9}$$

M_c is the mobility function in the Cahn-Hilliard equation that describes the rate of diffusion of the concentration field, typically depending on the concentration u. The chemical potential in the Cahn-Hilliard equation represents the driving force for phase separation, derived as the functional derivative of the system's free energy with respect to the concentration c.

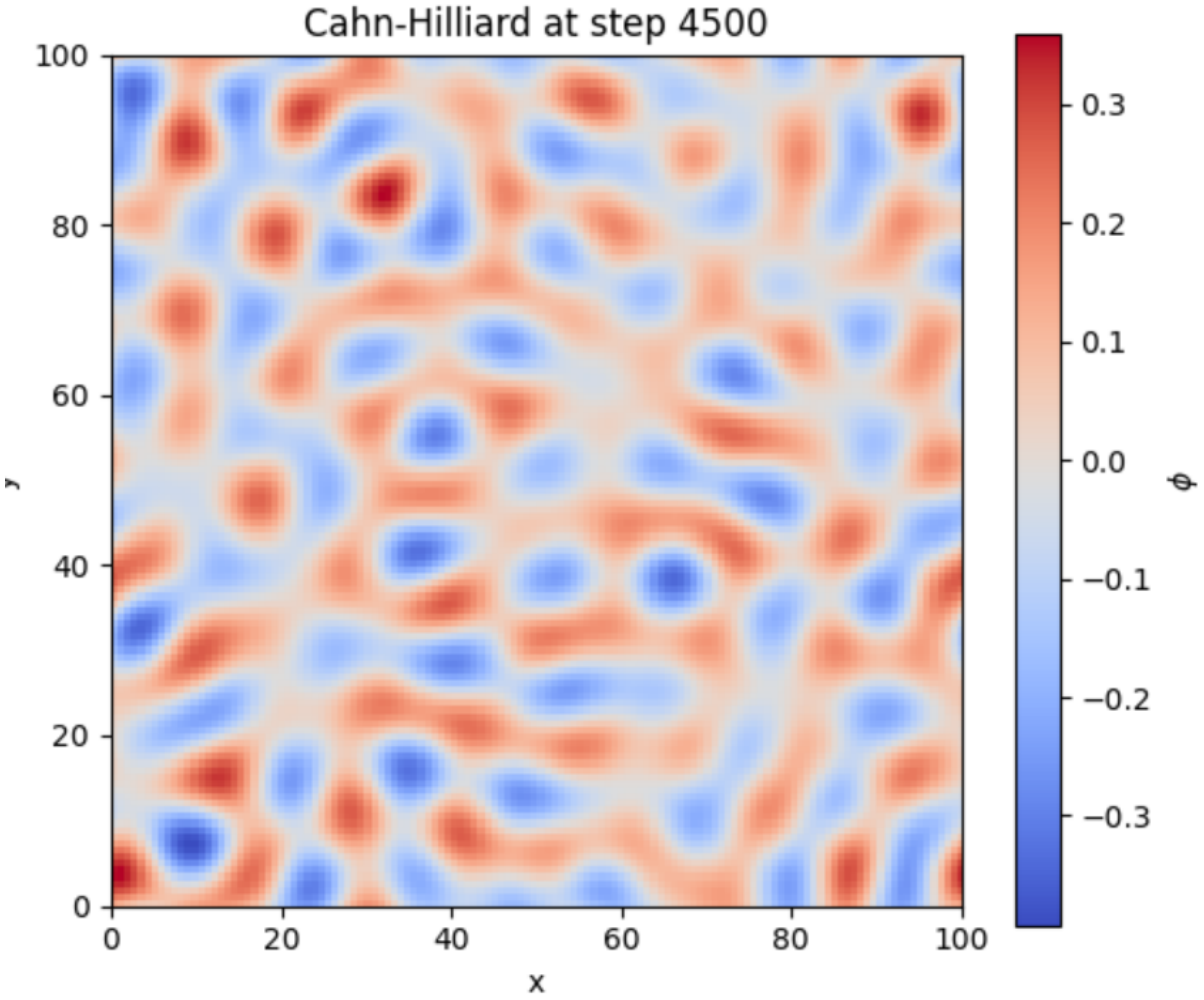
	2000	4000	6000	8000	10000
20	0.13893	0.13718	0.01532	0.01444	0.01433
40	0.13580	0.09106	0.01387	0.01375	0.01306
60	0.11929	0.08882	0.01351	0.01297	0.01276
80	0.08474	0.07869	0.01333	0.01278	0.01227
100	0.07005	0.06778	0.01286	0.01273	0.01214

Table 5: N_u vs N_f Table

	2000–4000	4000–6000	6000–8000	8000–10000
20	0.01837	5.40603	0.20720	0.03252
40	0.57668	4.64011	0.03065	0.23107
60	0.42562	4.64447	0.14070	0.07284
80	0.10693	4.37789	0.14887	0.18278
100	0.04740	4.09853	0.03770	0.20941

Table 6: Computed values across intervals of 2000 units.

As shown in the table, increasing the values of N_u and N_f leads to a significant decrease in the loss function, signifying higher accuracy in the model's performance.



2.2 Discrete Time Models

For discrete time models, we apply Runge-Kutta methods to the general PDE form:

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

$$u^{n+1} = u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}] \tag{10}$$

2.2.1 Discrete Time Model for Burgers' Equation

Following the discrete time framework proposed by Raissi et al., the Burgers' equation can be solved by leveraging implicit Runge–Kutta methods. The general discrete formulation is expressed as:

$$u^{n+c_i} = u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], \quad i = 1, \ldots, q, \tag{11}$$

$$u^{n+1} = u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}], \tag{12}$$

where $\mathcal{N}[u]$ denotes the nonlinear operator corresponding to the Burgers' equation, Δt is the time step size, and $\{a_{ij}, b_j, c_j\}$ define the Runge–Kutta scheme.

For the one-dimensional viscous Burgers' equation:

$$u_t + uu_x = \nu u_{xx}, \tag{13}$$

the nonlinear operator is $\mathcal{N}[u] = uu_x - \nu u_{xx}$.

In this framework, a multi-output neural network is employed to approximate the intermediate stages $u^{n+c_j}(x)$ and the final state $u^{n+1}(x)$. The network is trained to satisfy the discrete-time Runge–Kutta residuals at a set of collocation points, effectively enforcing the physics of the system without needing dense temporal data. This approach enables taking very large time steps without sacrificing stability or accuracy, thanks to the use of high-order implicit Runge–Kutta schemes, making it particularly efficient for stiff nonlinear PDEs like Burgers' equation.

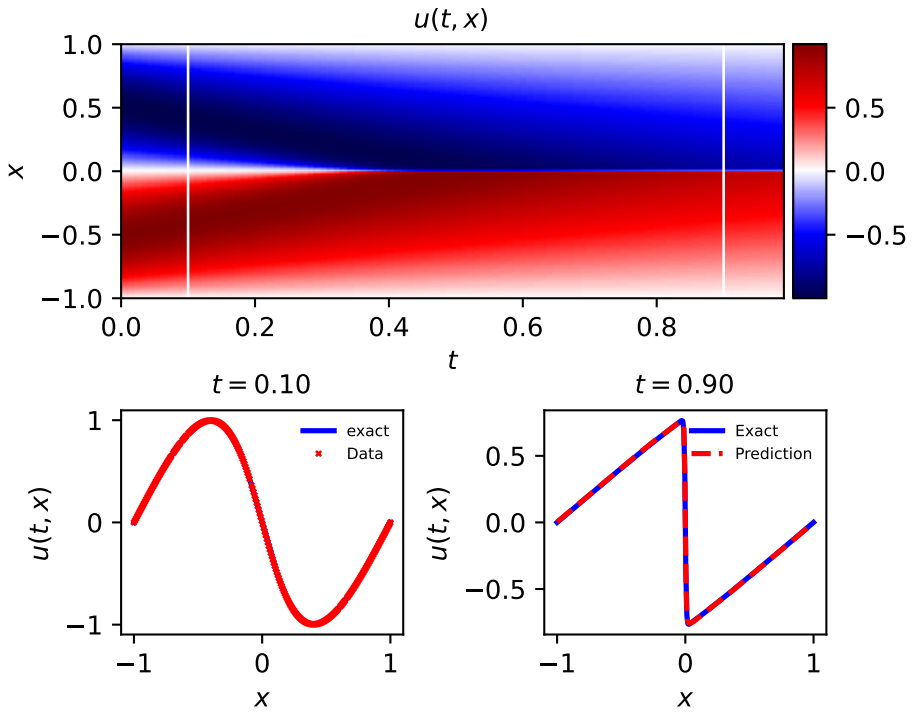
q	$\delta t = 0.2$	$\delta t = 0.4$	$\delta t = 0.6$	$\delta t = 0.8$
1	0.035028525	0.11004553	0.228297338	0.375338256
2	0.007118945	0.051910978	0.090987705	0.09543623
4	0.003480386	0.010396939	0.03266187	0.040469777
8	0.004592469	0.006252498	0.008332861	0.024373775
16	0.003610709	0.00476564	0.011588301	0.028217085
32	0.003573923	0.004282821	0.005074797	0.025276707
64	0.003372173	0.003741181	0.005913021	0.008323381
100	0.003298198	0.00281849	0.002983324	0.007659137
500	0.003292774	0.003592792	0.003480864	0.002296612

Table 7: Table of values for different q and δt

$q \setminus \delta$ (delta t)	0.2–0.4	0.4–0.6	0.6–0.8
1	1.6515	1.7998	1.7282
2	2.8663	1.3841	0.1659
4	1.5788	2.8232	0.7451
8	0.4452	0.7084	3.7309
16	0.4004	2.1915	3.0935
32	0.2611	0.4185	5.5811
64	0.1498	1.1290	1.1885
100	0.2268	0.1402	3.2774
500	0.1258	1.1037	1.4455

Table 8: Table of values for different q and δt intervals.

As shown in the Table, increasing the values of q and Δt results in a significant decrease in the loss function, indicating an improvement in accuracy and the overall stability of the Runge-Kutta method.



3 Conclusion

Physics-informed neural networks offer a powerful framework for both solving and discovering partial differential equations. The key advantages include:

- Ability to handle nonlinear problems without linearization
- Data efficiency through physics-based regularization
- Flexibility in handling various types of PDEs
- Capability to work with scattered and noisy data

In our study, we were able to determine the relationship between N_u (number of boundary points) and N_f (number of collocation points), showing that higher values of N_u and N_f resulted in significantly more accurate predictions in the continuous case. Furthermore, in the discrete case, where a third-order Runge-Kutta (RK3) method was employed, we observed that larger values of q (number of time steps) and Δt (time step size) led to improved model performance and better approximation accuracy.

The methods showcase promising results across diverse problems in computational science, opening new possibilities for data-driven modeling and scientific computing.

4 Bibliography

1. M. Raissi, P. Perdikaris, and G.E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. https://doi.org/10.1016/j.jcp.2018.10.045

2. L. Lu, X. Meng, Z. Mao, and G.E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021. https://arxiv.org/abs/2102.04626

3. A.D. Jagtap, K. Kawaguchi, and G.E. Karniadakis, "Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks," *Proceedings of the Royal Society A*, vol. 476, no. 2239, 2020. https://arxiv.org/abs/2004.04638

4. H. Shi and Y. Li, "Local discontinuous Galerkin methods with implicit-explicit multistep time-marching for solving the nonlinear Cahn-Hilliard equation," *Journal of Computational Physics*, vol. 394, pp. 719–731, 2019.

5. 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." (Duplicate entry, often cited in multiple places.)

6. GitHub Repository: https://github.com/maziarraissi/PINNs

7. GitHub Repository: https://github.com/ganatma/PINNs_Cahn_Hilliard.git