Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

SWIM Networks with Adaptive Basis Functions for Solving PDEs

Jyotishman Hazarika

Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

SWIM Networks with Adaptive Basis Functions for Solving PDEs

Author:	Jyotishman Hazarika
Supervisor:	Univ.-Prof. Dr. Felix Dietrich
Advisor:	Qing Sun, M.Sc.
Submission Date:	Nov 3rd, 2025

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Nov 3rd, 2025 Jyotishman Hazarika

Acknowledgments

Firstly, I would like to express my sincere gratitude to Prof. Dr. Felix Dietrich for providing me with the opportunity to work on this thesis and for his ongoing support. Next, I would like to express my deepest gratitude to Qing Sun for sharing her experience, being available for insightful discussions, and providing valuable feedback during the entire thesis.
My heartfelt thanks to my friends and colleagues from the CSE program for making this journey both enriching and memorable. Lastly, I would like to thank my family who have supported me throughout my life.

Abstract

The `swimpde` framework presents an efficient, mesh-free alternative to traditional solvers for Partial Differential Equations (PDEs) using the "Sampling Where It Matters" (SWIM) framework. The motivation for this thesis stems from the need to determine where we actually require the basis functions for a given simulation. This thesis extends the capabilities of the `swimpde` framework to robustly solve the 1D and 2D wave equations with adaptive basis functions. The primary contributions are threefold. First, we identify and correct the multi-block resampling strategy for second-order time-dependent PDEs, ensuring that the full physical state, including time derivatives (u and u_t), is correctly transferred between blocks. Second, we implement an adaptive resampling method for 2D problems by implementing a flexible acceptance-rejection sampling algorithm, moving beyond the limitations of the 1D inverse transform method. Third, we develop a new, robust probability density function (PDF) for resampling. This new PDF incorporates a Gaussian filter to smooth the solution gradients, which has proven essential for stable tracking of oscillatory wavefronts and preventing error amplification upon reflection, a problem that plagued naive gradient-based approaches.

These contributions are validated through a series of numerical experiments. The enhanced solver accurately matches analytical solutions for 1D and 2D wave propagation problems, which include a 1D reflecting Gaussian pulse and a 2D droplet simulation. The results demonstrate that the new PDF is efficient in terms of numerical stability for reflection problems, and the 2D resampling strategy effectively captures complex, propagating wave patterns for 2D wave problems.

Contents

1 Introduction

Deep Neural Networks have seen a surge in interest in recent years for solving Partial Differential Equations, driven by advancements in the fields of Computational Science and Engineering [17]. The practical application of neural network-based PDE solvers in industry has become more viable due to advances in hardware, such as powerful GPUs and sophisticated distributed computing frameworks, which enable the scalable training and parallel operation of large neural networks[21, 16, 23].

This has led to a major shift from traditional numerical methods to Neural Networks for approximating PDE solutions, due to their ability to handle high-dimensional problems and construct basis functions in high dimensions without the need for mesh generation. The challenge of solving a PDE can be transformed into an optimization problem by utilizing techniques such as Physics-Informed Neural Networks (PINNs) [34], which integrate physical principles directly into their learning process.

However, the training of these networks is often computationally expensive, prone to issues such as vanishing or exploding gradients, and typically involves parametrizing the solution over the entire space and time domain [39]. This necessitates an expensive hyperparameter search over network architectures and training parameters. An alternative approach that has gained traction is the use of randomized neural networks, such as Extreme Learning Machines (ELMs) [19], where the weights of the hidden layers are fixed and only the output weights are trained. This significantly reduces the training time and complexity. More recently, the idea of "Sampling Where It Matters" (SWIM) has been introduced, which is a data-driven approach to initialize the weights of the network [5]. By sampling the internal weights and using individual neurons as fixed basis functions, these methods reframe the problem, often reducing it to solving a single linear system for static PDEs or a system of ordinary differential equations for time-dependent problems.

This thesis builds upon the work of solving PDEs with sampled neural networks, specifically `swimpde` [10], with a focus on extending the resampling techniques to higher-order time-dependent problems. The primary contribution of this research will be the development and analysis of an adaptive resampling strategy for the collocation points, tailored to the dynamics of the wave equation. We will explore how to adapt the sampling in response to propagating wave fronts and evolving solution features, a critical aspect for maintaining accuracy and efficiency in long-time simulations.

In this thesis, we start with a state of the art that provides the background knowledge related to this thesis, which includes the Wave Equation section 2.1, an explanation of Adaptive Mesh Refinement (AMR) both in a classical sense, and a review of existing AMR methods with PINNs section 2.2. Then, a brief introduction to Neural Networks is given

in section 2.3. Next, we provide a quick rundown of solving time-dependent PDEs using Neural Networks in section 2.4. The main body first presents the implementation changes to the `swimpde` framework, starting with changes with forced boundary condition section 3.1.1, resampling for higher order time derivatives section 3.1.2, domain resampling for 2D space section 3.1.3, and finally changes to the probability density function section 3.1.4. Next, we move on to our experiments with empirical results section 3.2. The final chapter summarizes our method, listing some limitations and outlining future work.

2 State of the Art

2.1 Wave Equation

The origins of the wave equation can be traced back to the study of vibrating strings. Jean le Rond d'Alembert was the first to formulate and solve the equation in his work, providing a general solution that described the superposition of two waves traveling in opposite directions. Leonhard Euler further contributed to the understanding of wave phenomena shortly thereafter, exploring solutions involving trigonometric series and also generalizing them for higher dimensions.

The principles established by d'Alembert and Euler were later generalized to two and three dimensions, allowing the equation to model a much broader range of physical systems [9, 13]. The foundational nature of the wave equation is evident in some groundbreaking papers on acoustics and theoretical physics, where it serves as the starting point for analyzing more complex wave phenomena [31]. Its relevance to this day is demonstrated by its continued use as the primary model for acoustic propagation in modern textbooks and research [1, 24], and it now serves as the basis for advanced computational techniques, such as approximating solutions with neural operators [25].

The wave equation is a second-order partial differential equation that provides a fundamental description of how waves propagate through a medium over time. It mathematically models phenomena such as vibrating strings, the propagation of sound, and ripples on the surface of water. In its simplest one-dimensional form, the equation is expressed as:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{2.1}$$

Here, $u(x,t)$ represents the displacement or amplitude of the wave at a specific position x and time t, and c is a constant representing the propagation speed of the wave. This elegant formulation relates the acceleration of the displacement at a point ($\partial^2 u/\partial t^2$) to its local curvature ($\partial^2 u/\partial x^2$), capturing the core mechanism of wave propagation.

The equation is readily generalized to describe waves in multiple spatial dimensions. For waves on a two-dimensional surface, the equation for a function $u(x,y,t)$ reads as:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{2.2}$$

This can be expressed more compactly using the Laplacian operator, ∇^2, which repre-

sents the sum of the spatial second derivatives:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u \tag{2.3}$$

This operator is central to numerous physical models, from electromagnetism to fluid dynamics. In the context of the wave equation (e.g., $u_{tt} = c^2 \Delta u$), the Laplacian Δu quantifies the spatial characteristics of the wave. Intuitively, the Laplacian at a given point provides a measure of the function's local curvature, indicating the degree to which the value u at that point deviates from the average value of its immediate surroundings. For example, a positive Laplacian ($\Delta u > 0$) at a point implies the function's value there is lower than the average of its neighbors, a concept crucial for understanding diffusion and wave propagation [30].

Next, to ensure a unique and physically meaningful solution, the wave equation must be provided with initial and boundary conditions. Initial conditions specify the state of the system at a starting time (e.g., initial displacement and velocity). Boundary conditions, which define the behavior of the wave at the edges of the spatial domain ($\delta\Omega$), are also critical for creating a well-posed problem.

The main types include:

- **Dirichlet conditions**, which specify the value of the function u directly on the boundary.
- **Neumann conditions**, which define the value of the normal derivative of the function on the boundary, often characterizing the flow across it.
- **Cauchy conditions**, which specify both the function and its normal derivatives on a portion of the boundary and are particularly relevant for hyperbolic PDEs like the wave equation.

Often, these conditions are combined to create mixed boundary conditions. The appropriate choice is dictated by the type of problem we deal with.

2.2 Adaptive Mesh Refinement

When we attempt to solve Partial Differential Equations that describe natural phenomena, such as tsunamis or air flows, we often encounter complicated equations that may have no exact solution, and we must resort to numerical solutions. Some examples of methods that can be used to obtain numerical solutions include the Finite Difference Method, the Finite Element Method, and the Finite Volume Method. The core idea is to discretize the problem domain into a set of points or cells, collectively known as a mesh. The more points we have, the finer our mesh, the better our approximation will be; however, this comes at a steep computational and memory cost. Additionally, in any computational simulation,

we generally do not require uniform precision throughout the entire grid, as some areas may have high gradients and others may have zero values.

This imbalance led to the development of Adaptive mesh refinement (AMR). It says, "put the points where you need them the most." A more scientific definition would be: "A numerical technique that dynamically refines (adds points) or coarsens (removes points) from a computational grid during a simulation to enhance accuracy in regions with steep gradients while reducing resolution where the solution is smooth."

2.2.1 Classical Approaches to AMR

The classical approach to AMR, particularly for hyperbolic systems that describe wave propagation, involves a structured, patch-based refinement strategy. As introduced by Berger and LeVeque, this method overlays finer grid patches on top of a coarser base grid in regions identified as needing higher resolution [3]. These patches can be recursively refined to achieve the desired level of accuracy. For time-dependent problems, refinement is performed in both space and time, with the time step being refined by a same factor as space, to maintain stability and accuracy. The decision to refine is typically driven by an error estimation procedure, such as Richardson extrapolation, which identifies regions where the solution's truncation error is large.

A similar approach is to adapt the mesh based on the geometric properties of the solution itself. For nonlinear dispersive wave equations, such as the Korteweg-de Vries (KdV) equation, the main features are often stable, traveling waves known as solitons. Fraga and Morris developed a method that explicitly locates these solitons within the solution profile [14]. Once identified, fine, uniform sub-meshes are placed to cover the spatial extent of each soliton, including a buffer region to account for its movement. The regions between these solitons are then filled with a much coarser mesh. This approach, while less general than error-based indicators, is highly effective for problems where the solution structure is well understood and dominated by distinct, localized features. It minimizes the overhead of error estimation and places points precisely where the wave action is concentrated.

The classical geometric refinement of a mesh can be viewed through the lens of adapting the basis of functions used to approximate the solution. In a finite element or finite volume context, refining the mesh introduces new, smaller elements, which in turn add new basis functions with smaller support to the function space. This allows the discrete solution to capture finer details.

2.2.2 Modern Approaches: Adaptive Sampling in Physics-Informed Neural Networks (PINNs)

The core idea of adapting the computational grid can be viewed more abstractly as adapting the basis of functions used to approximate the solution. This perspective provides a natural bridge to modern machine learning techniques, particularly Physics-Informed

Neural Networks (PINNs). PINNs are neural networks trained to solve PDEs by minimizing a loss function based on the PDE's residual—the amount by which the network's output fails to satisfy the equation. Like classical methods, PINNs can struggle to resolve solutions with steep gradients when trained on a fixed, uniform set of points (known as collocation points). To overcome this, the philosophy of AMR has been reborn in the context of adaptive sampling.

Modern adaptive methods, such as those applied in the Boundary Element Method (BEM) for the wave equation, directly address the approximation quality within a chosen function space. For instance, Aimi et al. [2] present a space-time adaptive BEM where local error indicators are derived from a residual-based a posteriori error estimate. These indicators guide the refinement of the basis of piecewise polynomials in either space or time. This allows the method to respond not only to geometric singularities (like corners or edges in the domain) but also to transient singular behavior in the time domain, such as the initial phase of a wave front.

The moving mesh method describes a situation where the number of nodes (and thus the number of basis functions) remains constant, but their spatial locations are dynamically adjusted to follow the evolving features of the solution. Lu et al. demonstrate this for the Regularized Long-Wave (RLW) equation, where a moving mesh PDE is solved to cluster the nodes in regions of high solution activity, such as solitary waves [26]. This can be interpreted as adapting the basis by translating and stretching the support of the basis functions to optimally represent the wave at each time step.

The Residual-based Adaptive Refinement (RAR) for PINNs [33] describes that during training, the algorithm identifies regions in the domain where the PDE residual is highest. It then adaptively adds new collocation points to these high-error regions, forcing the network to focus its learning capacity on the most challenging parts of the solution. This is a mesh-free approach, where refinement occurs during the sampling of training data rather than on a geometric grid.

Recently, there has been a considerable amount of focus on improving the placement of collocation points in PINNs. The authors in [28] also do this by clustering points in high gradient zones. This approach, however, is not automatic and depends on knowing the solution's behavior in advance. A similar adaptive strategy in [32] samples collocation points from a probability distribution proportional to a piecewise constant approximation of the loss function. Other such similar adaptive strategies are also explained and compared in [38, 27, 41]

To conclude, the classical sense of adaptive meshing for wave problems focuses on the geometric subdivision of the domain (h-adaptivity). However, this is intrinsically linked to the broader concept of adapting the approximation basis itself. Whether by adding new basis functions through mesh refinement, increasing their polynomial order (p-adaptivity), or repositioning them through mesh movement (r-adaptivity), the underlying goal remains the same: to tailor the discrete function space to the specific structure of the wave, thereby achieving an accurate and efficient numerical simulation.

2.3 Neural Networks

Artificial Neural Networks (ANNs) are mathematical frameworks that emulate the learning process of biological brains. They are constructed from a collection of connected units known as neurons, which are organized to collectively analyze and learn from data patterns [15].

The Neuron

The fundamental processing element of a neural network is the neuron. It functions by receiving signals, processing them, and transmitting a new signal to other neurons. The primary components that define a neuron's operation are:

- **Input Weights (w):** Every incoming signal is multiplied by a weight. This dictates its influence on the input signal for the neuron's output.
- **Activation Function (σ):** After summing the weighted inputs and a bias, the neuron passes the result through an activation function. This function governs the firing of the neuron and introduces non-linear properties to the model, enabling it to learn complex relationships.
- **Bias (b):** A bias is an additional, learnable parameter that is added to the weighted sum of inputs. It provides the model with an adjustable offset, increasing its flexibility to fit the data.

The output of a neuron is calculated as follows:

$$z = \sum_{i=1}^{n} w_i x_i + b, \quad a = \sigma(z) \tag{2.4}$$

Where:

- x_i is an input data or input feature.
- w_i is the weight associated with the input feature.
- b is the bias.
- z is the weighted sum.
- σ is the activation function).
- a is the output of the neuron.

Network Architecture

A neural network is structured into a sequence of layers, each containing a number of neurons. The conventional architecture includes three types of layers:

- **Input Layer:** The initial layer that takes in the raw data. It does not perform any computation but simply passes the data to the first hidden layer.
- **Hidden Layers:** One or more layers situated between the input and output layers. Neurons in these layers apply transformations to the data, allowing the network to learn progressively more complex features.
- **Output Layer:** The final layer that produces the network's prediction. The structure of this layer depends on the task (e.g., a single neuron for regression, multiple neurons for classification).

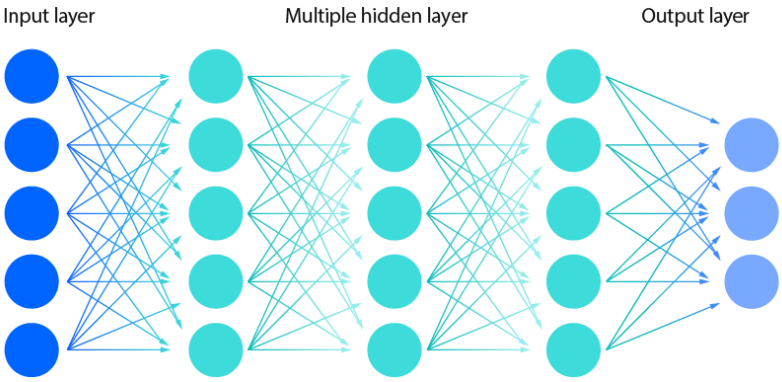


Figure 2.1: Neural Network Architecture with multiple hidden layers

Network Training

The training of a neural network is an iterative optimization process that refines the network's internal parameters, namely, the weights and biases, to minimize a loss function, which measures the discrepancy between the network's predictions and the true values. This procedure is primarily composed of two phases: forward propagation and backpropagation.

Forward Propagation

This is the inference phase, where data is passed through the network. Starting from the input layer, the output of each layer becomes the input for the next, until a final prediction is made at the output layer. The computation of each neuron follows eq. (2.4).

Backpropagation

This is the learning phase, during which the error is propagated backward through the network. It allows for the efficient calculation of the gradient of the loss function with respect to each parameter. The main steps are:

1. **Loss Calculation:** The error is quantified by comparing the network's prediction ($\hat{y}$) with the target value (y), by using a loss function, for example, Mean Squared Error.

2. **Gradient Computation:** The chain rule is applied to calculate the partial derivative of the loss function with respect to each weight and bias in the network.

3. **Parameter Update:** An optimization algorithm uses these gradients to adjust the parameters in a way that reduces the loss. The foundational algorithm for this is gradient descent, but one can opt for other algorithms like Stochastic Gradient Descent (SGD), Adam, or the Least Squares method.

2.4 Solving Partial Differential Equations with Neural Networks

2.4.1 Solving PDEs with Physics-Informed Neural Networks

Classical Physics-Informed Neural Networks (PINNs) serve as a foundational deep learning method for solving PDEs [34, 8]. They approximate the solution $u(\mathbf{x}, t)$ with a neural network, which we denote as $u^*(\mathbf{x}, t; \mu)$ following the notation in [10]. Here, μ represents the complete set of trainable network parameters (i.e., all weights and biases) and λ_i is the weighting factor for each term.

Instead of relying on a mesh, a PINN is trained by minimizing a composite loss function, $L(\mu)$, which is designed to enforce all known physical laws and data constraints simultaneously. This total loss is a weighted sum of several components, is defined as:

$$L(\mu) = \lambda_1 L_{PDE}(\mu) + \lambda_2 L_{IC}(\mu) + \lambda_3 L_{BC}(\mu) + \lambda_4 L_{Data}(\mu) \tag{2.5}$$

where λ_i are hyperparameters used to weight the influence of each loss term.

The individual loss components are defined as follows:

1. **PDE Loss (L_{PDE}):** This term enforces the governing equation itself. It is evaluated at N_{int} collocation points randomly sampled from the interior of the spatio-temporal domain. This loss is the mean-squared error of the PDE residual:

$$L_{PDE}(\mu) = \frac{1}{N_{int}} \sum_{n=1}^{N_{int}} ||u_t^*(x^{(n)}, t^{(n)}) + \mathcal{L}u^*(x^{(n)}, t^{(n)}) + \lambda \mathcal{N}(u^*)(x^{(n)}, t^{(n)}) - f(x^{(n)})||^p \tag{2.6}$$

2. **Initial Condition Loss (L_{IC}):** This term ensures the network's solution matches the known starting state of the system at $t = 0$. It is evaluated at N_{ic} points:

$$L_{IC}(\mu) = \frac{1}{N_{ic}} \sum_{n=1}^{N_{ic}} ||u_0^*(x^{(n)}) - u_0(x^{(n)})||^p \tag{2.7}$$

3. **Boundary Condition Loss (L_{BC}):** This term enforces the solution's behavior at the domain's boundaries. It is evaluated at N_b boundary points:

$$L_{BC}(\mu) = \frac{1}{N_b} \sum_{n=1}^{N_b} ||Bu^*(x^{(n)}, t^{(n)}) - g(x^{(n)})||^p \tag{2.8}$$

To compute the derivatives required for the L_{PDE} term (like u_t^* and the spatial derivatives in $\mathcal{L}u^*$), PINNs rely on automatic differentiation (AD). This allows the gradient of the total loss $L(\mu)$ to be computed with respect to all network parameters μ via the backpropagation algorithm. The network is then trained using gradient-based iterative optimization (e.g., Adam) to find the parameters μ that minimize $L(\mu)$ [21].

However, a significant drawback of this classical PINN formulation is its reliance on gradient descent optimization, which creates a challenging, high-dimensional, and non-convex optimization problem [35]. This often leads to difficulties in training and challenges in resolving sharp features, such as shocks or high-frequency dynamics.

2.4.2 Sampling Where It Matters (SWIM) for solving PDEs

Introduced by Bolager et al.[5], the Sampling Where It Matters (SWIM) method presents a data-driven approach for constructing the weights and biases of fully connected neural networks, thereby bypassing the need for traditional iterative, gradient-based optimization of hidden layers. Its fundamental principle is that the parameters for each neuron are determined directly by a pair of points sampled from the input space. Consequently, only the final linear output layer requires training via a standard least-squares method.

A key innovation of SWIM lies in its sampling strategy. Instead of a data-agnostic approach like random feature models, SWIM employs a conditional probability distribution to select pairs of input points, favoring those that exhibit a large gradient in the target function. This process is governed by the following distribution, $p^{(l)}$:

$$p^{(l)}(x_0^{(1)}, x_0^{(2)} | \{W_j, b_j\}_{j=1}^{l-1}) \propto \begin{cases} \frac{\|f(x_0^{(2)}) - f(x_0^{(1)})\|_{\mathcal{Y}}}{\|x_{l-1}^{(2)} - x_{l-1}^{(1)}\|_{\mathcal{X}}} & \text{if } x_{l-1}^{(1)} \neq x_{l-1}^{(2)} \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

Here, $f(x)$ is the target function, and $x_{l-1}^{(1)}$ and $x_{l-1}^{(2)}$ are the outputs of the $(l-1)$-th layer. This formulation mathematically ensures that point pairs corresponding to steeper gradients are more likely to be selected, strategically placing the activation functions in regions

where the function is changing most rapidly [5]. For a given pair of points, the weight ($W_{l,i}$) and bias ($b_{l,i}$) are defined as:

$$W_{l,i} = s_1 \frac{x^{(2)}_{l-1,i} - x^{(1)}_{l-1,i}}{\|x^{(2)}_{l-1,i} - x^{(1)}_{l-1,i}\|^2}, \quad b_{l,i} = \langle W_{l,i}, x^{(1)}_{l-1,i} \rangle + s_2 \tag{2.10}$$

The constants s_1 and s_2 are chosen based on the activation function to control the neuron's output.

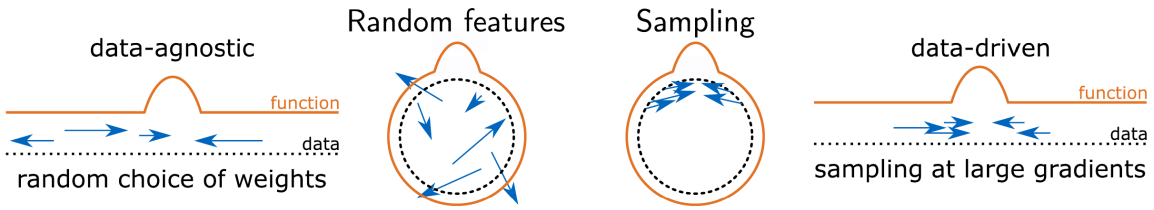


Figure 2.2: Data-agnostic model places weights randomly, compared to a Data-driven model where it samples at large gradients [5].

In the work by Datar et al., this framework was extended to solve Partial Differential Equations (PDEs), adapting the core principles of SWIM to a scientific computing context where neurons act as basis functions for a solution ansatz [10]. A significant adaptation is required because the solution to a PDE is unknown beforehand, making the gradient-based probability distribution (Eq. 2.9) unusable at the outset. Instead, an initial uniform sampling of input point pairs is used. While this initial step is data-agnostic, the key advantage of constructing basis functions from points within the domain is retained, ensuring the shifts of these functions are also within the domain.

A significant contribution of the `swimpde` framework, or now `Frozen-PINN` by Datar et al. [10], is its method for handling time-dependent PDEs. Instead of parameterizing the solution over the entire space-time domain, this approach uses a separation of variables. The neural network ansatz is structured so that the spatial basis functions are time-independent, while the output layer coefficients are time-dependent.

We consider a general time-dependent PDE defined on a spatial domain $\Omega \subset \mathbb{R}^d$ with boundary $\partial\Omega$. The goal is to find a solution $u(x,t)$ that satisfies:

$$u_t + \mathcal{L}u(x,t) + \gamma\mathcal{N}(u)(x,t) = f(x), \quad x \in \Omega, t \in [0,T] \tag{2.11}$$

subject to the boundary and initial conditions:

$$\mathcal{B}u(x,t) = g(x), \quad x \in \partial\Omega, \quad \text{and} \quad u(x,0) = u_0(x), \quad x \in \Omega \tag{2.12}$$

Here, $\mathcal{L}$ and $\mathcal{B}$ are linear operators (spatial derivatives), $\mathcal{N}$ is a nonlinear operator (in the case of nonlinear problems), and $f(x)$ is a forcing term.

The method approximates the solution $\hat{u}(x,t)$ using an ansatz with a single hidden layer comprised of M neurons and an activation function $\sigma = \tanh$ as:

$$\hat{u}(x,t) = C(t)[\Phi(x), \mathbb{1}] = \sum_{m=1}^{M} c_m(t)\phi_m(x) + c_0(t) \tag{2.13}$$

where:

- $\phi_m(x) = \sigma(w_m \cdot x^T + b_m)$ is the output of the m-th hidden neuron.
- $W \in \mathbb{R}^{M\times d}$ and $b \in \mathbb{R}^{M\times 1}$ are the space-independent weight matrix and bias vector of the hidden layer. These parameters are frozen and do not change over time, derived from eq. (2.10).
- $c(t) \in \mathbb{R}^{1\times M}$ and $c_0(t) \in \mathbb{R}$ are the time-dependent coefficients of the output layer.
- $\Phi(x) = [\phi_1(x), \ldots, \phi_M(x)]$ is the vector of hidden layer activations, and $C(t) = [c(t), c_0(t)]$.

This ansatz explicitly separates the spatial basis functions, $\Phi(x)$, from the time-dependent coefficients, $C(t)$.

Once the spatial basis $\Phi(x)$ is frozen, the ansatz (Equation 2.13) is inserted directly into the governing PDE (Equation 2.11). This substitution effectively separates the spatial and temporal components.

Since $\Phi(x)$ is fixed, the spatial derivative operators $\mathcal{L}$ and $\mathcal{B}$ act only on $\Phi(x)$, not on $C(t)$. Conversely, the time derivative u_t acts only on $C(t)$, becoming $C_t(t)[\Phi(x), \mathbb{1}]$.

This process transforms the original PDE into a system of first-order ODEs for the time-dependent coefficients $C(t)$. This reformulation is evaluated at N_c collocation points $X \in \mathbb{R}^{N_c\times d}$. The resulting ODE system is given by:

$$C_t(t) = R(X, C(t))[\Phi(X), \mathbb{1}]^+ \tag{2.14}$$

where $[\cdot]^+$ denotes the pseudo-inverse, and R is the residual term derived from the PDE:

$$R(X, C(t)) = -C(t)\mathcal{L}[\Phi(X), \mathbb{1}] - \gamma\mathcal{N}(C(t)[\Phi(X), \mathbb{1}]) + [f(X)]^T \tag{2.15}$$

The optimization problem here is simplified as the initial condition $C(0)$ is computed directly using a single least-squares solution based on the initial condition $u_0(x)$:

$$C(0) = u(X, 0)^T[\Phi(X), \mathbb{1}]^+ \tag{2.16}$$

With $C(0)$ determined, the system of ODEs (Equation 2.14) is solved forward in time using standard, efficient numerical ODE solvers (in our case, the Runge-Kutta RK45 [12]), completely bypassing the need for gradient-descent-based optimization of the temporal dynamics.

To summarize, the importance of the data-driven placement of these basis functions becomes evident when dealing with solutions that have localized features. For example, in solving nonlinear PDEs that produce sharp gradients or shocks, such as the Burgers' equation, a purely random approach like ELM struggles to represent the steep gradient. In contrast, the `swimpde` approach allows for adaptive strategies. By concentrating collocation points near an emerging shock and applying SWIM's data-driven sampling to the intermediate solution, the method can generate numerous basis functions with steep gradients and accurately place them where they are needed most, resulting in a significantly more accurate outcome [10].

3 Solving Partial Differential Equations using SWIM Networks

This chapter is divided into two main sections: implementation details and experiments. We will start by explaining all the necessary changes made to the `swimpde` repository[1] to implement adaptive basis functions, and then we will move on to the experiments, which are focused on Wave equations, that showcase the robustness of our new method.

3.1 Implementation Details

3.1.1 Extended Support for Forced Boundary Conditions

The `TimeDependentSolver` has been updated to support the direct forcing of the Zero Neumann ($\nabla u \cdot \mathbf{n}|_{\partial\Omega} = 0$) boundary condition now. The previous implementation was restricted to Zero Dirichlet condition only. This modification was vital to our problem cases, namely the Wave Equation, as we work with this boundary condition for this set of equations.

The solver's core strategy is to transform the spatiotemporal PDE into a system of ordinary differential equations (ODEs) that governs the time-dependent coefficients of an ansatz. The solution $u(x,t)$ is approximated by a linear combination of spatial basis functions $\phi_i(x)$:

$$u(x,t) \approx \sum_{i=1}^{N} c_i(t)\phi_i(x) = \mathbf{A}(x)\mathbf{c}(t)$$

The primary challenge is to enforce a spatial boundary condition within an ODE system that only explicitly evolves the temporal coefficients $c_i(t)$. A homogeneous boundary condition, represented by a linear operator B such that $B(u)|_{\partial\Omega} = 0$, imposes a linear constraint on the coefficient vector $\mathbf{c}(t)$. The evolution of $\mathbf{c}(t)$ must therefore be confined to a specific subspace where this constraint is always satisfied. The recent modifications ensure this through two critical mechanisms: correct initial projection and constrained temporal evolution.

First, we need to modify the logic in `_get_initial_state` in the time solver. We need to find the initial coefficient vector $\mathbf{c}(0)$ that is the best fit for the PDE's initial solution, $u_0(x)$, while also perfectly satisfying the boundary condition.

[1] https://gitlab.com/fd-research/swimpde

This is a constrained least-squares problem. We want to find $\mathbf{c}(0)$ that minimizes:

$$\underbrace{\|\mathbf{A}(x)\mathbf{c}(0) - u_0(x)\|^2}_{\text{Fit the initial solution}} \quad \text{subject to} \quad \underbrace{\mathbf{B}(x)\mathbf{c}(0) = 0}_{\text{Satisfy the boundary condition}}$$

The most direct way to solve this is to combine both equations into a single linear system. We stack the boundary constraint equation underneath the main solution equation:

$$\begin{bmatrix}\mathbf{A}(x)\\ \mathbf{B}(x)\end{bmatrix}\mathbf{c}(0) = \begin{bmatrix}u_0(x)\\ \mathbf{0}\end{bmatrix}$$

The code change in `_get_initial_state` implements exactly this. It builds the right-hand-side vector $\begin{bmatrix}u_0(x)\\ \mathbf{0}\end{bmatrix}$ (as `initial_with_bc`) and then solves for $\mathbf{c}(0)$ using the pre-computed inverse of the stacked matrix $\begin{bmatrix}\mathbf{A}\\ \mathbf{B}\end{bmatrix}$. This ensures that our solution starts in a valid state that already respects the neumann boundary condition.

Next, we move onto the `_state_derivative` logic. Even with a perfect start, tiny numerical errors during the ODE integration will cause the solution to drift away from the constraint, meaning $\mathbf{Bc}(t) \neq \mathbf{0}$ for $t > 0$.

We must continuously correct this drift. Since our solution is governed by its highest-order derivative (e.g., $\mathbf{c}_{tt}$ for the Wave Equation), we must inject this correction into the equation that solves for $\mathbf{c}_{tt}$.

The solver's main task is to solve the system $\mathbf{Ac}_{tt} = \mathbf{F}$, where $\mathbf{F}$ represents the PDE dynamics. To enforce the boundary, we must also satisfy the constraint's second derivative, $\mathbf{Bc}_{tt} = \mathbf{0}$.

To correct for drift, we use a stabilization technique. Instead of setting the constraint equation to $\mathbf{0}$, we set it to a correcting force proportional to the current error, $\mathbf{Bc}(t)$. This creates a combined system that is solved at every time step:

$$\underbrace{\begin{bmatrix}\mathbf{A}\\ \mathbf{B}\end{bmatrix}}_{\texttt{evaluation}} \quad \underbrace{\mathbf{c}_{tt}}_{\texttt{highest\_order}} = \underbrace{\begin{bmatrix}\mathbf{F}\\ -k\cdot(\mathbf{Bc})\end{bmatrix}}_{\texttt{concatenated update}}$$

This is exactly what the code change in `_state_derivative` implements. It extracts $\mathbf{c}$, computes the current error $\mathbf{Bc}$ (as `boundary_correction`), stacks it with the PDE dynamics F (the `update`), and then solves for $\mathbf{c}_{tt}$ using the precomputed inverse. This ensures our solution is continuously pulled back to the valid boundary constraint throughout the simulation.

3.1.2 Resampling for Second-Order Time-Dependent PDEs

When resampling is enabled, the solver employs a multi-block strategy to solve a PDE. This allows the ansatz (the set of basis functions, Φ) to be re-adapted to the solution's changing features, which is crucial for efficiency and accuracy. To implement this mathematically correctly, the state of the system at the end of one block must be accurately transferred as the initial state for the next.

The previous implementation had a critical flaw when resampling was enabled, which occurred at the handoff between computation blocks. For a time-dependent problem, the solver must track the entire state of the system, which includes the solution u as well as its time derivatives (like u_t, u_{tt}, etc.).The flaw was that when starting a new block, the solver would only use the solution u from the end of the previous block to set the new initial conditions. It failed to carry over the information about the time derivatives (u_t, u_{tt}), incorrectly assuming they were zero. This resulted in a non-physical discontinuity at the start of each new block, compromising the accuracy of the entire simulation.

The correction rectifies this by properly distinguishing between two distinct tasks at the start of a new block ($i > 0$). Firstly, ansatz fitting where the new set of basis functions for the current block, Φ_i, only needs to learn the spatial shape of the solution field, $u(x, t_{end})$. Secondly, ODE Initialization, where the new ODE system for the coefficients, $[\mathbf{c}, \dot{\mathbf{c}}, \dots]$, requires initial values for the full physical state, $[u, u_t, \dots, u_{t^{k-1}}]$, evaluated at t_{end}.

The new logic is implemented in two parts: the main `fit` loop, which manages the block-to-block logic, and the new helper function, which reconstructs the state , as shown in algorithm 1. The new helper method, `_create_full_initial_solution_fn`, has been implemented to correctly reconstruct this full state vector from the previous block's solved coefficients, ensuring a smooth and accurate transfer of data as shown in algorithm 2.

The algorithm initializes the solver and sets the initial state functions based on the properties of the equation. It then iterates through each time block. The critical logic change occurs for any block after the first block. Here, it performs the two distinct tasks: it creates a target function for fitting the new ansatz, which provides only the solution u, and then it calls the helper function to create a target function for the full ODE state. These functions are then used to fit the new ansatz and compute the initial coefficient vector, respectively, before solving the ODE for the current time block.

The helper method, `_create_full_initial_solution_fn`, acts as a factory: it takes the index of the previous block and a specific time t, and it returns a new function (the `full_initial_solution_fn` defined inside it). This new function encapsulates the previous block's ansatz (Φ_{prev}) and its solved ODE state. When this newly returned function is called with a set of spatial points, it evaluates the ODE solution at time t to get the final coefficient vector $[\mathbf{c}, \dot{\mathbf{c}}, \dots]$. It then iterates through each time order, reconstructing the corresponding physical field (e.g., $u = \Phi_{\text{prev}} \cdot \mathbf{c}$, $u_t = \Phi_{\text{prev}} \cdot \dot{\mathbf{c}}$, etc.) and returns them as a single, concatenated vector, $[u, u_t, \dots]$.

Algorithm 1 Main Solver Loop (`fit` method logic)

Require: TimeSpan $T = [T_{\text{start}}, T_{\text{end}}]$, `n_time_blocks`, Domain, Equation, ResamplingEnabled
Ensure: List of block-wise ODE solutions `block_solutions`

Initialize all necessary things.
`block_size` $\leftarrow (T_{\text{end}} - T_{\text{start}})$/`n_time_blocks`
for $i \leftarrow 0$ to `n_time_blocks` - 1 **do**
    $t_{\text{start\_block}} \leftarrow T_{\text{start}} + i \cdot \textit{block\_size}$
    $t_{\text{end\_block}} \leftarrow t_{\text{start\_block}} + \textit{block\_size}$
    if $i > 0$ **then**
        if ResamplingEnabled **then**
            Create gradient `gradient_fn` using block $i - 1$ at $t_{\text{start\_block}}$
            Resample domain using `gradient_fn`.
        end if
        Get solution u from block $i - 1$ at $t_{\text{start\_block}}$
        $f_{\text{initial}} \leftarrow$ Function call algorithm 2 using block $i - 1$ and $t_{\text{start\_block}}$
    else
        $f_{\text{initial}} \leftarrow$ Use equation's initial condition
        if ResamplingEnabled **then**
            Fit first ansatz using initial condition of the equation
        end if
    end if
    Get initial ODE state $\mathbf{c}_0$ (coefficients) using f_{initial}
    Solve ODE for block i from $t_{\text{start\_block}}$ to $t_{\text{end\_block}}$ with initial state $\mathbf{c}_0$
    Store solution for block i in `block_solutions`
end for
Return `block_solutions`

Algorithm 2 Create Full Initial Solution Function

Require: Previous block index $i-1$, time value t_{start}
Ensure: A function `FullInitialSolutionFn(points)`
1: Get solved coefficients from block $i-1$ at time t_{start}
2: Determine how coefficients are split (for $u, u_t, \ldots$)
3: Initialize an empty list for the full solution
4: **for** each time order k (e.g., $u, u_t, \ldots$) **do**
5: Extract the specific coefficients for order k
6: Evaluate the basis functions from block $i-1$ at the input `points`
7: Reconstruct the field for order k by combining basis and coefficients
8: Append the reconstructed field to the list
9: **end for**
10: Concatenate all fields in the list into one vector
11: **Return** the full state vector $[u, u_t, \ldots]$

3.1.3 Domain Resampling for 2-D Spaces

Adaptive sampling is a critical technique for enhancing the efficiency of numerical solvers, particularly in physics-informed neural networks, for problems with localized, high-frequency features such as shock fronts or complex wave patterns, as seen in section 2.2. By concentrating computational points in regions of high gradients or complex behavior, we can achieve higher accuracy with fewer resources [20]. A common way to implement this is the magnitude of the solution's spatial gradient, $\|\nabla u(x)\|$. Points are then resampled from a probability distribution proportional to this gradient norm. This method is implemented to handle the steep gradients in the non-linear Burgers' equation, where the resampling probability density is defined as $p(x) \sim |\nabla \hat{u}(x, t_r)|$ [10]. The initial implementation in swimpde provided a `ResamplingDomain` class for 1D problems, but extending this to 2D required a change in strategy.

Limitations of the Original 1D Approach

The 1D resampling strategy is based on inverse transform sampling. This method works by first computing a probability density function (PDF) based on the solution's gradients. This PDF is then integrated to find the cumulative distribution function (CDF). By sampling from a uniform distribution and passing these values through the inverse of the CDF, new points are generated that are concentrated in regions specified by the PDF [36, 11]. While efficient in 1D, this method is difficult to generalize to higher dimensions because it requires computing and inverting a multi-dimensional CDF, which is computationally expensive and complex to implement robustly.

The New 2D Sampling Strategy: Acceptance-Rejection

To overcome these limitations, we have implemented an acceptance-rejection sampling algorithm [29] for the 2D case. This method, originally developed by John von Neumann, is more general and avoids the direct computation of the inverse CDF. The core idea is to sample uniformly from a bounding box that contains the domain and then accept a sample with a probability proportional to the value of the target PDF at that point. The updated `resample_domain` method now contains distinct logic for 1D and 2D cases, with the 2D implementation following the steps outlined in algorithm 3.

Algorithm 3 2D Acceptance-Rejection Sampling

Input: Current sample points $\{\mathbf{x}_i\}$, gradient function ∇f, PDF function g.
Compute gradient magnitudes $g_i = ||\nabla f(\mathbf{x}_i)||_2$ for all points.
Compute PDF values $p_i = g(g_i)$ and find the maximum, $M = \max(\{p_i\})$.
Construct a KD-Tree from the current sample points $\{\mathbf{x}_i\}$.
Initialize an empty set of new points, $\mathcal{P}_{new}$.
while $|\mathcal{P}_{new}| <$ required number of points **do**
    Generate a candidate point $\mathbf{x}_c$ uniformly from the domain's bounding box.
    Find the nearest neighbor $\mathbf{x}_n$ to $\mathbf{x}_c$ in the KD-Tree, and get its PDF value p_n.
    Generate a random number $u \sim U(0, 1)$.
    if $u < p_n/M$ **then**
        Add $\mathbf{x}_c$ to $\mathcal{P}_{new}$.
    end if
end while
Return: The set of new points $\mathcal{P}_{new}$.

The algorithm begins by calculating a probability density value for each existing point, derived from the magnitude of the solution's gradient at that point. It then generates random candidate points from a uniform distribution covering the entire domain. To decide whether to keep a candidate point, the algorithm finds the PDF value of the closest original sample point, using an efficient KD-Tree[2] for this nearest-neighbor search [18]. The candidate is accepted with a probability proportional to this PDF value. This process is repeated until the desired number of new points has been collected, resulting in a new set of points that are more densely clustered in areas where the solution's gradient is high.

Robustness and Fallback

The acceptance-rejection method may be inefficient sometimes if the target distribution is highly peaked. In such cases, a large number of candidate points might be rejected, slowing down the resampling process. To ensure the method terminates and generates

[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html

the required number of points, a fallback mechanism is included. If the algorithm fails to generate enough points after a set number of iterations, the remaining points are sampled uniformly from the domain. This ensures that the simulation can proceed, although with a partially sub-optimal sampling distribution.

In conclusion, the `ResamplingDomain` class has been extended to support 2D spatial domains by incorporating a robust acceptance-rejection sampling strategy. This modification significantly broadens the utility of our numerical framework, allowing it to tackle a wider range of multidimensional problems.

3.1.4 Resampling Probability Density Function

The goal of adaptive resampling is to dynamically allocate computational resources (collocation points) to regions where the solution exhibits the most significant variation or complexity. The effectiveness of the resampling algorithm is highly dependent on the probability density function (PDF) we pick.

The initial implementation in `swimpde` was inspired by work on problems featuring sharp shocks, such as the Burgers' equation [10]. This method employed a PDF directly proportional to the magnitude of the solution's spatial gradient, $p(x) \propto ||\nabla u(x)||_2$. While effective for capturing sharp, monotonic shock fronts, this direct approach proved less robust for oscillatory solutions, such as those of the wave equation. Neural network approximations, particularly those with a fixed basis like SWIM, can introduce spurious high-frequency oscillations or noise into the computed gradients, even when the overall solution appears smooth [40, 22]. A PDF based directly on these potentially noisy gradients can then disproportionately concentrate sample points around non-physical peaks, neglecting other important areas of the wavefront and leading to numerical instability, as clearly observed in the reflecting wall experiment section 3.2.2.

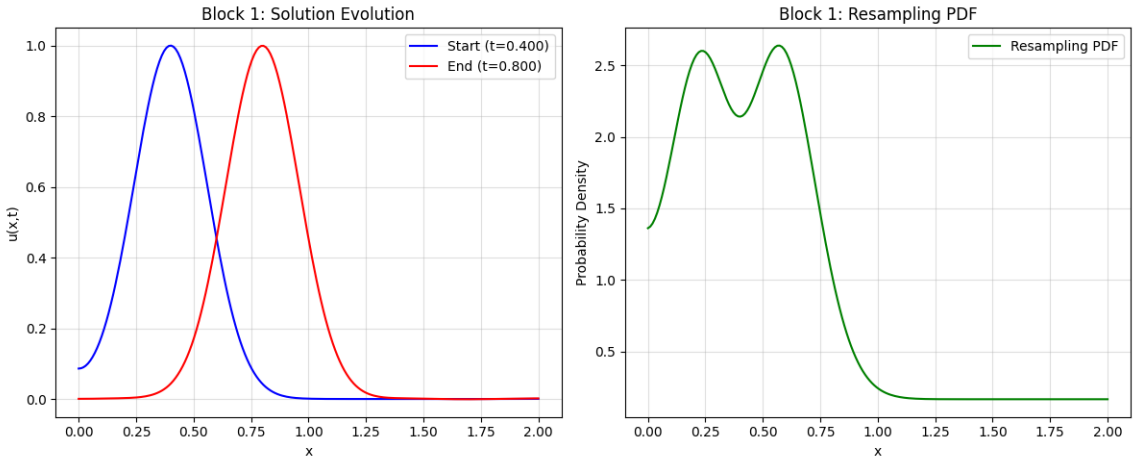


Figure 3.1: Resampling PDF logic showcase

To create a more stable and effective distribution, particularly for the wave equation, a new PDF function was implemented. This function incorporates a Gaussian filter to smooth the gradient field before generating the probability distribution, leading to a more robust and generalized resampling strategy, drawing inspiration from techniques used in adaptive mesh refinement [6]. The key steps are:

1. **Gradient Magnitude Calculation:** The function first ensures it is working with scalar values by calculating the L2 norm ($||\nabla u(x,t)||_2$) at each existing sample point x, as the gradient is a vector field in multiple dimensions.

2. **Baseline Addition:** A small baseline value ϵ, typically proportional to the maximum gradient magnitude observed ($\epsilon = \alpha \max(||\nabla u||_2)$. Where α is a small constant like 0.01 or 0.1), is added to all gradient magnitudes. This ensures that even regions with near-zero gradients retain a non-zero probability of being sampled, preventing the sampler from completely abandoning smooth regions and maintaining a degree of exploration across the domain.

3. **Gaussian Filtering:** The core improvement is the application of a Gaussian filter[3] (using `scipy.ndimage.gaussian_filter`) to the baseline-adjusted gradient magnitudes. This filter acts as a spatial low-pass filter, averaging out sharp, high-frequency numerical noise. By blurring the peaks in the raw gradient data, the filter helps the resampling focus on the broader development or region of significant average change associated with the wavefront, rather than overfitting to potentially erroneous local maxima. The standard deviation (σ) of the Gaussian kernel serves as a crucial tunable parameter controlling the degree of smoothing.

4. **Normalization:** Finally, the smoothed and baseline-adjusted values are normalized by their sum $\sum$(filtered values) to create a valid probability distribution where $\sum p(x_i) = 1$.

This enhanced approach to PDF generation provides several advantages. It makes the resampling process significantly more robust to noisy gradients, inherent in approximating oscillatory functions. It prevents overfitting to specific collocation points within the wave's pattern, instead encouraging sampling along the entire local development of the wavefront. This ultimately improves the overall numerical stability of the adaptive sampling, especially in scenarios involving reflections or complex wave interactions. While other metrics, such as the magnitude of the PDE residual [32, 41] or curvature (Laplacian) of the solution, could also guide resampling, the smoothed gradient offers a practical balance between targeting feature sharpness and computational simplicity within the `swimpde` framework.

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html

3.2 Experiments

For the following experiments, we use the `tanh` activation function, σ. For the data-dependent SWIM sampling section 2.4.2, the basis function parameters w_m and b_m are determined using s_1 and s_2 eq. (2.10). These parameters are set specifically for `tanh` such that the neuron's output is -0.5 at one selected collocation point and $+0.5$ at the other. In our implementation, `n_basis` defines the number of basis functions, and `n_time_blocks` defines the number of temporal intervals used for the periodic resampling strategy.

3.2.1 A Simple 1D Wave

To validate the existing adaptive resampling methodology in `swimpde`, we conduct numerical experiments on a 1D Wave equation. We choose a simple problem to test whether the resampling works well with the wave equation.

Problem Setup

The 1D wave equation, defined as:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{3.1}$$

where $u(x,t)$ is the wave displacement, and c is the wave propagation speed, which is set to $c = 1.0$ for these experiments.

The spatial domain for the simulation is a unit interval defined by $\Omega = [0,1]$, and the simulation is run over the time interval $t \in [0,1]$.

Initial and Boundary Conditions

The initial displacement u_0 and initial velocity u_{t0} are given by:

$$u_0(x) = \cos(\pi x)$$

$$u_{t0}(x) = 0$$

Homogeneous Neumann boundary conditions are enforced at both ends of the domain ($x = 0$ and $x = 1$), meaning the spatial derivative of the solution is zero:

$$\frac{\partial u}{\partial x} = 0 \quad \text{on } \partial\Omega$$

Exact Solution

For the given initial and boundary conditions, the problem has a known analytical solution, which serves as the ground truth for evaluating the accuracy of our numerical method. The exact solution is:

$$u(x,t) = \cos(\pi x)\cos(\pi t)$$

This solution represents a standing wave pattern that oscillates in time.

Numerical Results and Discussion

To demonstrate the efficiency and correctness of the adaptive resampling strategy, we use the Wave Equation as a benchmark. This particular problem case required two critical enhancements to the `TimeDependentSolver` before a meaningful comparison could be made.

First, solving our Wave Equation problem necessitates a forced zero neumann boundary condition ($\nabla u \cdot \mathbf{n} = 0$), which the solver did not previously support. This was resolved by implementing the two-part strategy detailed in section 3.1.1: (1) the initial coefficient vector $\mathbf{c}(0)$ is now found by solving a constrained least-squares problem to perfectly satisfy the boundary condition at $t = 0$, and (2) a continuous `boundary_correction` term is added to the ODE's state derivative. This term acts as a spring, constantly pulling the solution back to the valid subspace and counteracting any numerical drift that would otherwise violate the constraint during time integration.

Second, the Wave Equation is a second-order-in-time PDE, meaning its full physical state is defined by both the solution u and its time derivative u_t. As detailed in section 3.1.2, the previous resampling logic had a critical flaw: when handing off between time blocks, it would only transfer u, incorrectly resetting the velocity u_t to zero. This non-physical discontinuity has been resolved. The solver now uses a helper function to reconstruct the full state $[u, u_t]$ from the end of the previous block, ensuring this complete information is used as the initial condition for the next block.

With these two core issues addressed, we can now compare the solver's performance meaningfully. fig. 3.2 shows the solution obtained without using the resampling method, where sample points are distributed uniformly in the domain.

In fig. 3.3, we show the results when adaptive resampling is activated. The combined effect of correct Neumann boundary handling (section 3.1.1) and proper second-order state handoff (section 3.1.2) allows the resampling strategy to function correctly. The prediction is visibly closer to the ground truth, demonstrating that our enhanced resampling method is now effective and accurate for the wave Equation.

To ensure the correctness under the hood, we also plot the sample points along the solution and check if they are concentrated in regions where they are needed the most. In fig. 3.4 we can see that it is correctly depicted, where the plot clearly shows that the density

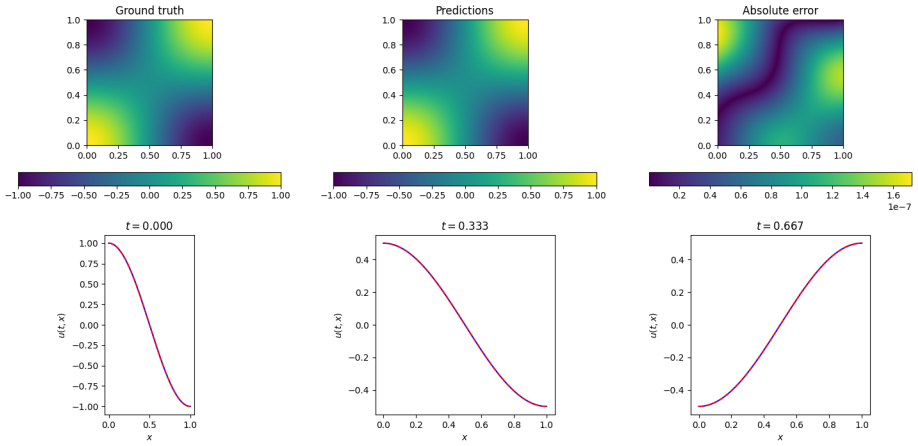


Figure 3.2: Comparison between ground truth and SWIM solution for the non-resampled case, with `n_basis` = 50

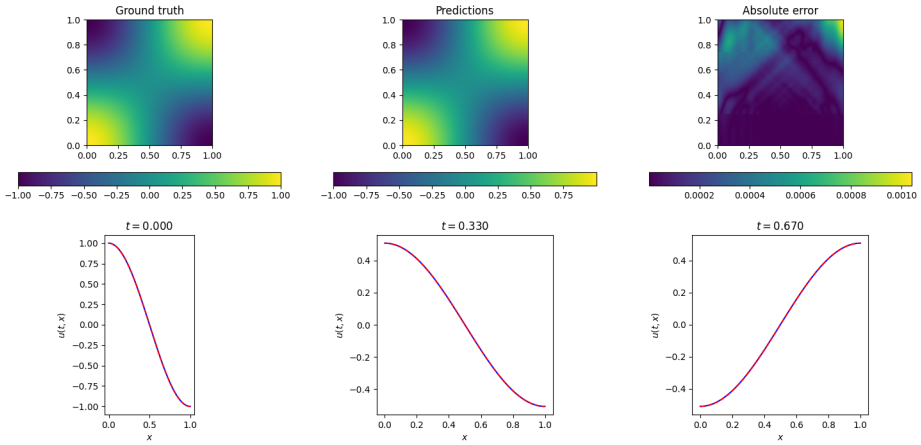


Figure 3.3: Comparison between ground truth and SWIM solution for the resampled case, with `n_basis` = 50 and `n_time_blocks` = 4

of the sample points is highest in areas with large gradients (in this case, the steepest parts of the wave).

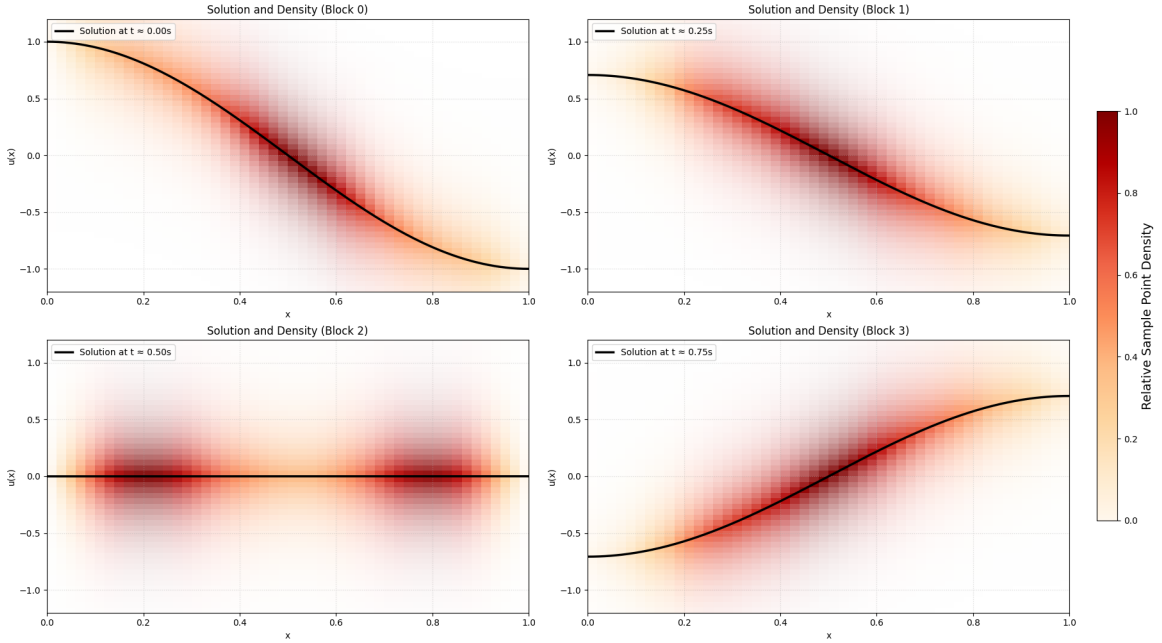


Figure 3.4: Solution and sampling density across each time block with the solution

3.2.2 Reflecting Wall

This section outlines the mathematical formulation for a 1D wave propagation problem involving a Gaussian pulse. An analytical solution exists for this configuration, providing a correctness check for validating the numerical method.

Problem Setup

The physical phenomenon is modeled by the 1D linear wave equation eq. (3.1), similar to our previous problem. The spatial domain is defined for $x \in [0, 2]$, and the simulation is conducted over the time interval $t \in [0, 4]$. For this problem, the material density is zero, and the wave speed is set to $c = 1$.

Initial and Boundary Conditions

A unique solution is determined by the following initial and boundary conditions [7].

The initial state of the system at $t = 0$ consists of a Gaussian displacement profile centered at the left boundary and zero initial velocity:

$$u(x, 0) = 2 \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{3.2}$$

$$\frac{\partial u}{\partial t}(x, 0) = 0 \tag{3.3}$$

where the parameter σ is defined as $\sigma = c/(2\pi f)$, with a frequency $f = 1$ Hz. Homogeneous Neumann boundary conditions are applied.

Exact Solution

The analytical solution for this problem is adapted from [7]. It is expressed as the superposition of four traveling Gaussian waves, which account for the initial wave and its reflections from the boundaries. The total solution is defined as:

$$u(x, t) = \tilde{u}_{+,1}(x, t) + \tilde{u}_{+,2}(x, t) + \tilde{u}_{-,1}(x, t) + \tilde{u}_{-,2}(x, t) \tag{3.4}$$

Let the wall position be $L = 2$, the individual components are defined as follows:

$$\tilde{u}_{+,1}(x, t) = \begin{cases} \exp\left(-\frac{(x-ct)^2}{2\sigma^2}\right) & \text{if } -L < x - ct \leq L \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

$$\tilde{u}_{+,2}(x, t) = \begin{cases} \exp\left(-\frac{(x-ct+2L)^2}{2\sigma^2}\right) & \text{if } -L < x - ct + 2L \leq L \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

$$\tilde{u}_{-,1}(x, t) = \begin{cases} \exp\left(-\frac{(x+ct)^2}{2\sigma^2}\right) & \text{if } -L < x + ct \leq L \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

$$\tilde{u}_{-,2}(x, t) = \begin{cases} \exp\left(-\frac{(x+ct-2L)^2}{2\sigma^2}\right) & \text{if } -L < x + ct - 2L \leq L \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

This solution provides the ground truth for validating the numerical results within the domain $x \in [0, L]$.

Numerical Results and Discussion

The correctness of the numerical results is evaluated against an exact solution, with the comparative results depicted in fig. 3.5. We observe that the solution plot has a notable divergence between the SWIM solution and the ground truth. The absolute error plot in fig. 3.5 further reconfirms that there is a significant accumulation of error as the simulation progresses in time.

To provide a more granular view of the solution's behavior, snapshots at various time instances are presented in figs. 3.6 and 3.7. We can see that during the initial phases of the simulation (from t=0.00s to t=0.50s), the SWIM solution maintains a high degree of accuracy with the exact solution. However, as the wave front approaches the domain boundary, noticeable errors begin to occur. Figure 3.7 captures the solution's behavior at later time steps (from t=2.00s to t=3.50s), where the inaccuracies become more pronounced. A key observation is the amplification of errors as the wave interacts with the wall and reflects off it. This suggests a propagation of numerical errors, which are worsened upon reflection.

This amplification of error, particularly upon reflection, highlights a common challenge in numerical methods for hyperbolic PDEs like the wave equation. Small errors introduced from the spatial discretization (i.e., the projection onto the finite element space) or the time-stepping scheme are not necessarily dissipated; instead, they are propagated with the wave's characteristic speed. The interaction with the boundary condition can act as a source of new errors or, as seen here, amplify existing ones, leading to a progressive loss of accuracy and eventual instability [42]. The observed divergence strongly suggests that the old PDF for the resampling strategy, combined with this boundary interaction, creates a feedback loop where numerical errors accumulate rather than decay.

The results presented so far highlight a limitation in the current implementation of the `swimpde` algorithm. The primary source of the observed numerical instability appears to be twofold. Firstly, the propagation of accumulated errors, a common challenge in numerical schemes. The error is not only transported with the wave but is also amplified upon reflection from the boundaries.

Secondly, and more critically, the issue is attributed to the formulation of the resampling probability density function (PDF) used in this simulation. The original PDF was designed to concentrate collocation points aggressively in regions of high gradients. While effective for sharp, localized fronts, this approach proves less robust for the oscillatory patterns of the wave equation, leading to a suboptimal distribution of collocation points and contributing to the escalating errors observed post-reflection.

To fix this, we implement a new resampling PDF, which incorporates a Gaussian filter to smooth the gradient field and ensure a more stable and generalized sampling strategy. This approach, detailed in section 3.1.4, gives us better results as we will soon see.

In figs. 3.9 and 3.10 we showcase the solver solution with the new PDF to showcase how the PDF function behaves with time. A key observation is that it behaves properly before and after colliding with the wall, too. The new results presented in figs. 3.8 and 3.11 show us that the predicted solution coincides with the ground truth, validating our resampling strategy.

During our experiments, we observed that in the first time block, the solver placed sample points uniformly across the domain. Instead of starting with a uniform distribution of sample points, the first resampling is guided by the initial condition of the PDE, now as shown in figs. 3.13a and 3.13b. The ansatz is first fit to this known initial state, and the resulting gradients are used to perform the first resampling. This provides the solver

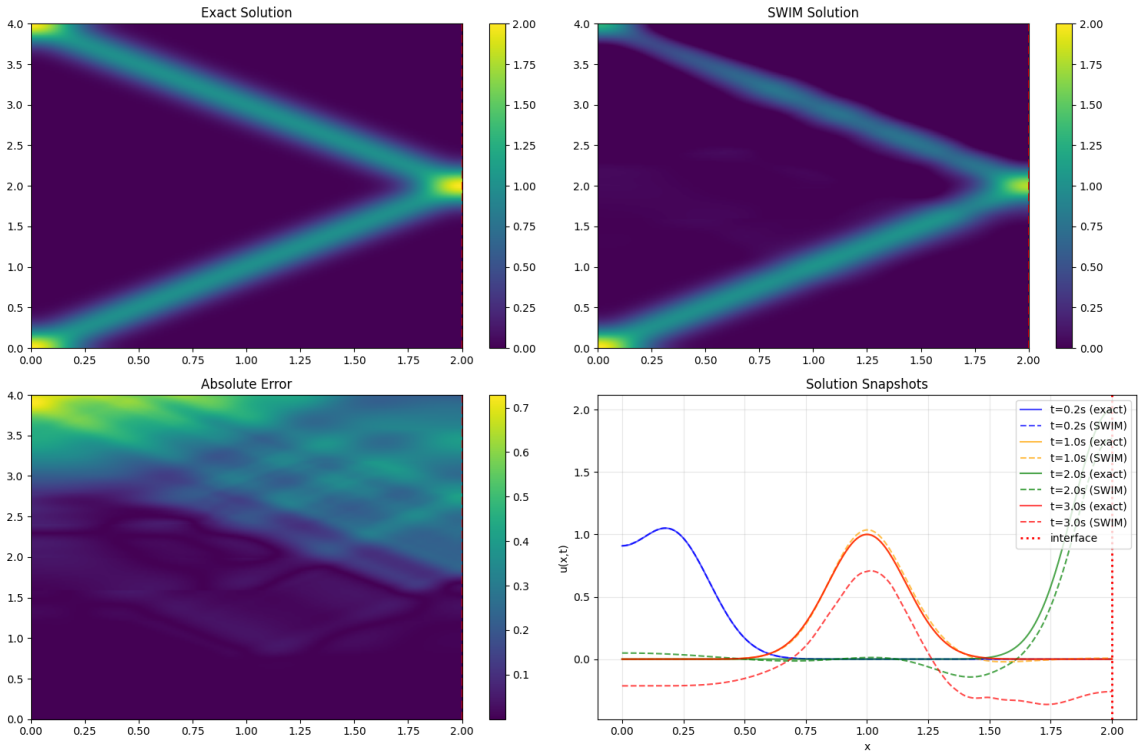


Figure 3.5: Comparison between exact solution and SWIM solution with resampling enabled (old PDF) with `n_basis` = 50 and `n_time_blocks` = 10

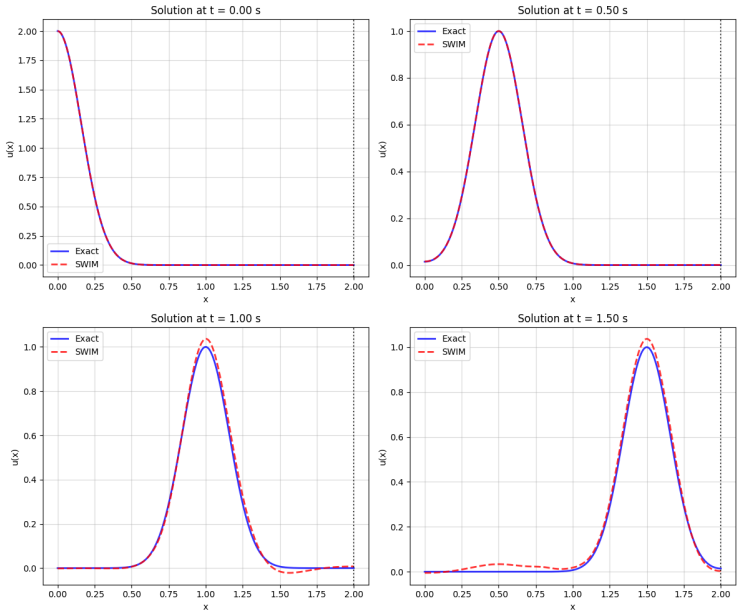


Figure 3.6: Comparison between Exact Solution and SWIM Solution with resampling enabled (old PDF)

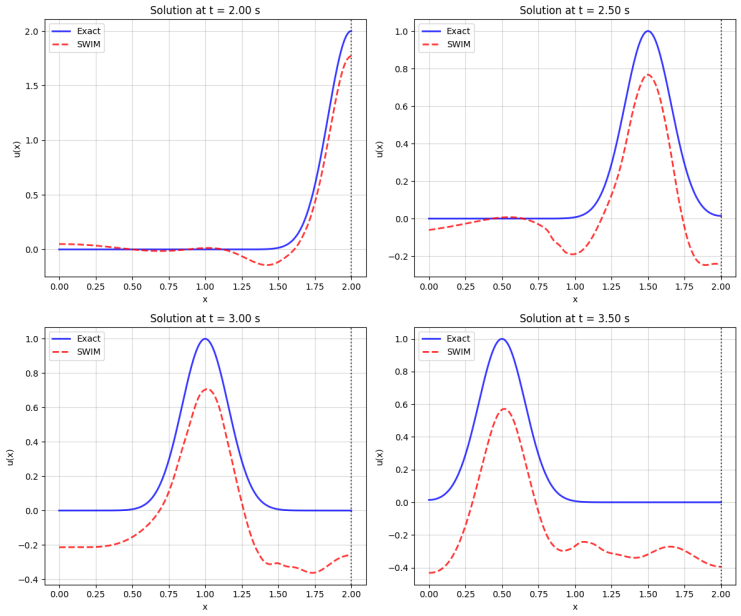


Figure 3.7: Comparison between Exact Solution and SWIM Solution with resampling enabled (old PDF)

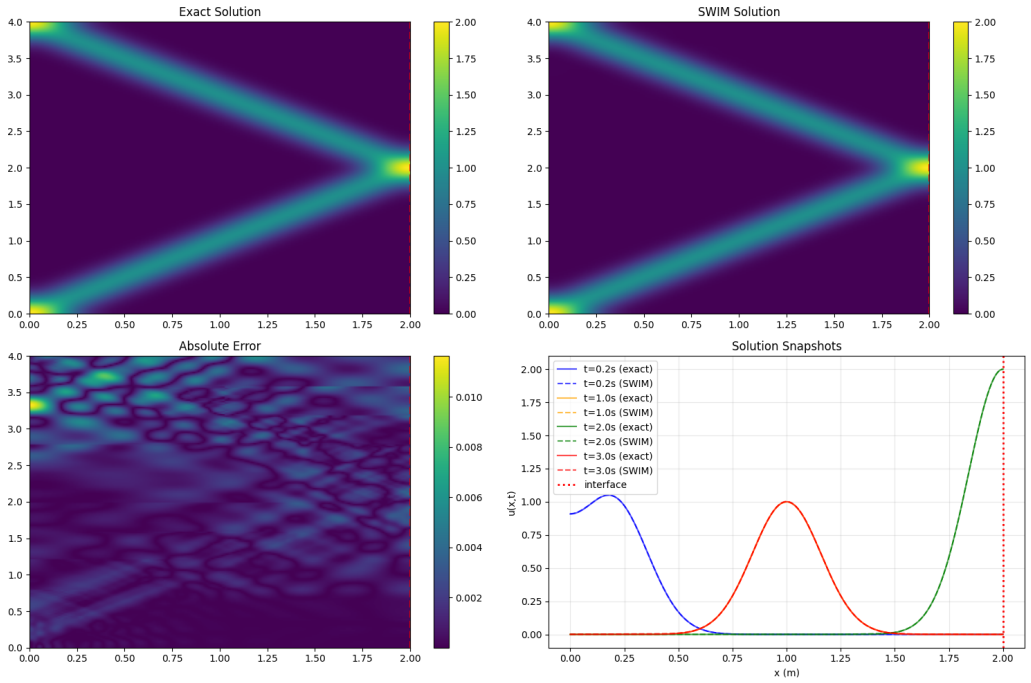


Figure 3.8: Comparison between exact solution and predicted solution with resampling enabled (new PDF), with `n_basis` = 50 and `n_time_blocks` = 10

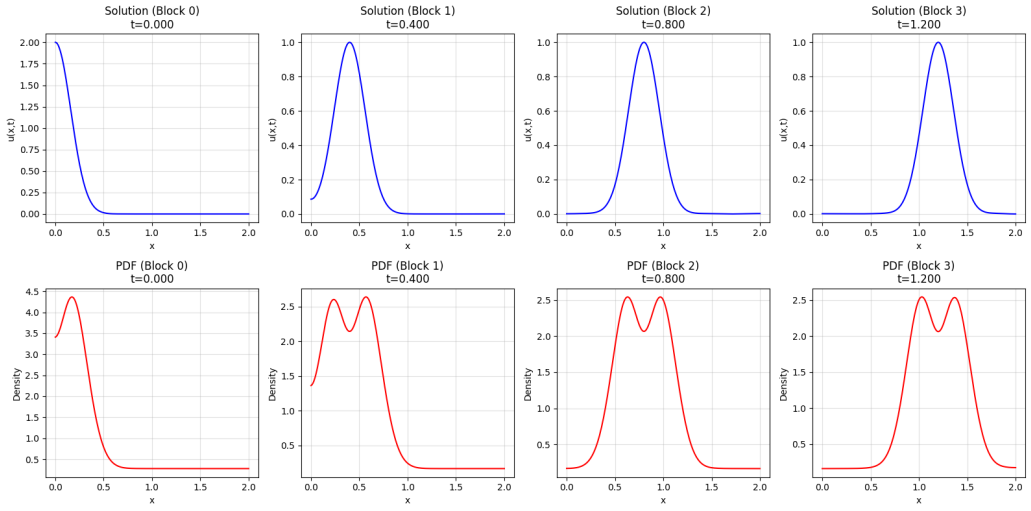


Figure 3.9: Predicted solution and PDF evolution across each time block

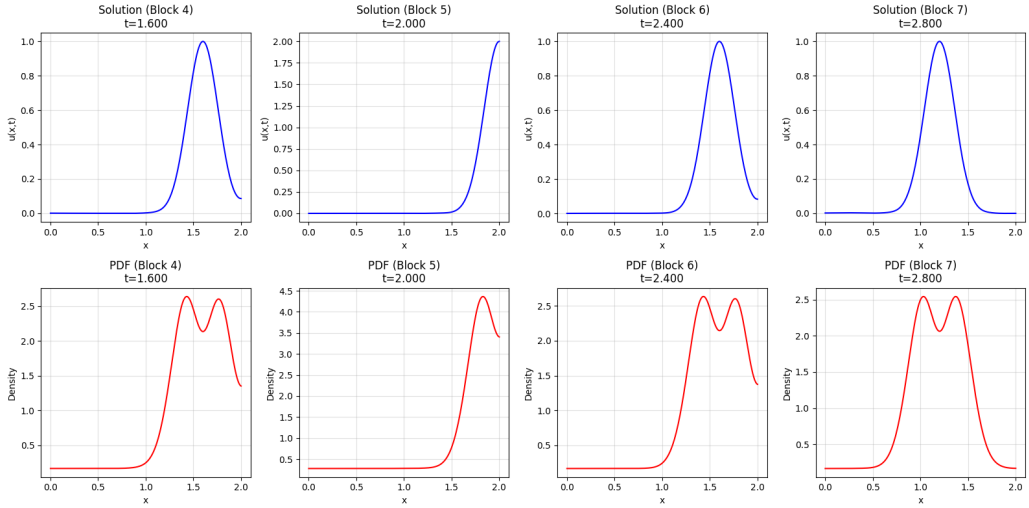


Figure 3.10: Predicted solution and PDF evolution across each time block

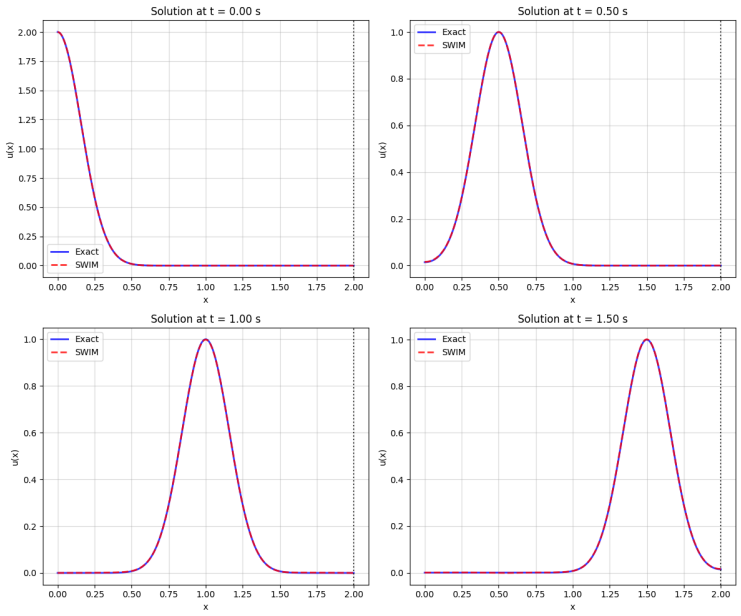


Figure 3.11: Comparison between exact solution and predicted solution with resampling enabled (new PDF) over different time stamps

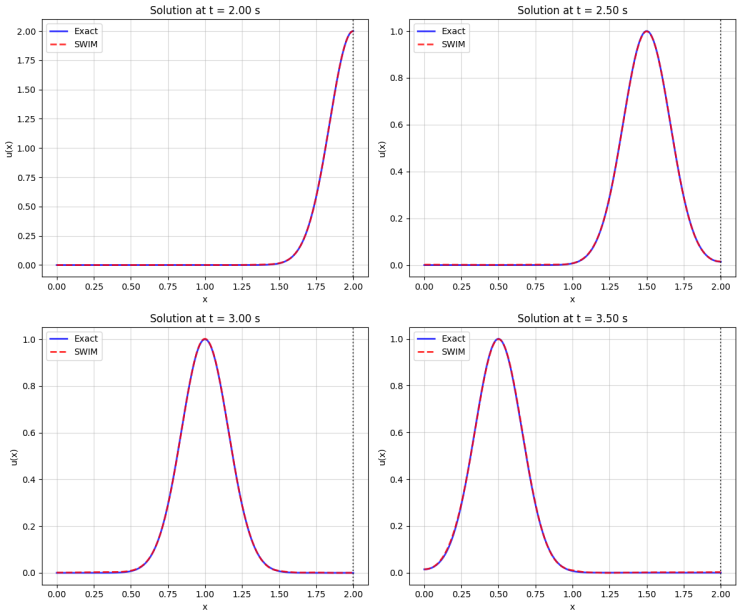


Figure 3.12: Comparison between exact Solution and predicted solution with resampling enabled (new PDF) over different time stamps

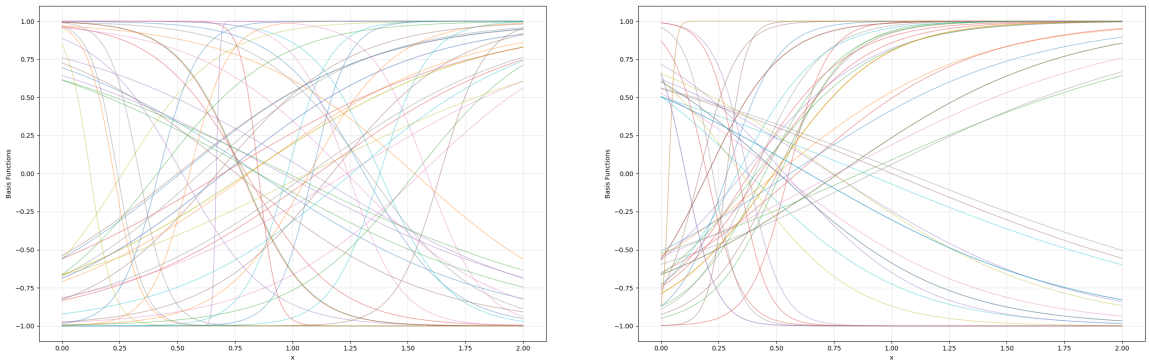
with a warm start, concentrating the initial set of sample points in regions of high initial complexity (e.g., sharp gradients or complex features in the wave's starting position) before the time-stepping even begins. This leads to a more accurate and efficient start to the simulation.

An additional verification, analogous to our previous experiment, was conducted by plotting the sample points to assess their concentration in high-gradient regions. The results, illustrated in figs. 3.14 and 3.15, confirm that the samples are appropriately localized, even subsequent to the reflection.

In fig. 3.15, an asymmetry is visible in blocks 4 and 6, with the sampling density clearly concentrated more towards the right. This occurs because the resampling is based on the numerical gradient of the predicted solution from the end of the previous block. The gradient carries some asymmetric numerical errors inherent in the model's random initialization. These small errors could be due to several factors; the wall collision is one of them, as the solver must numerically force the gradient to be zero at the boundaries. This enforcement introduces complex approximation errors into the solution, especially near the boundaries where the wave reflects. The observed asymmetry could also be due to the Gaussian filter, as the smoothing averages out high-frequency noise but preserves the underlying low-frequency, asymmetric bias in the numerical error, which becomes visible as the observed increased density on the right for blocks 4 and 6.

Upon experimentation, we found that this can be controlled by fine-tuning the baseline

and sigma factor with respect to our problem in the Gaussian filter function call. However, in our case, when the sampling density was made more symmetric, it led to the development of small numerical instabilities after the wall reflection. This observation further validated that the current PDF function, with its correct parameters, works well for our problem case.



(a) Old swimpde solver

(b) New swimpde solver

Figure 3.13: Basis functions at time block = 0

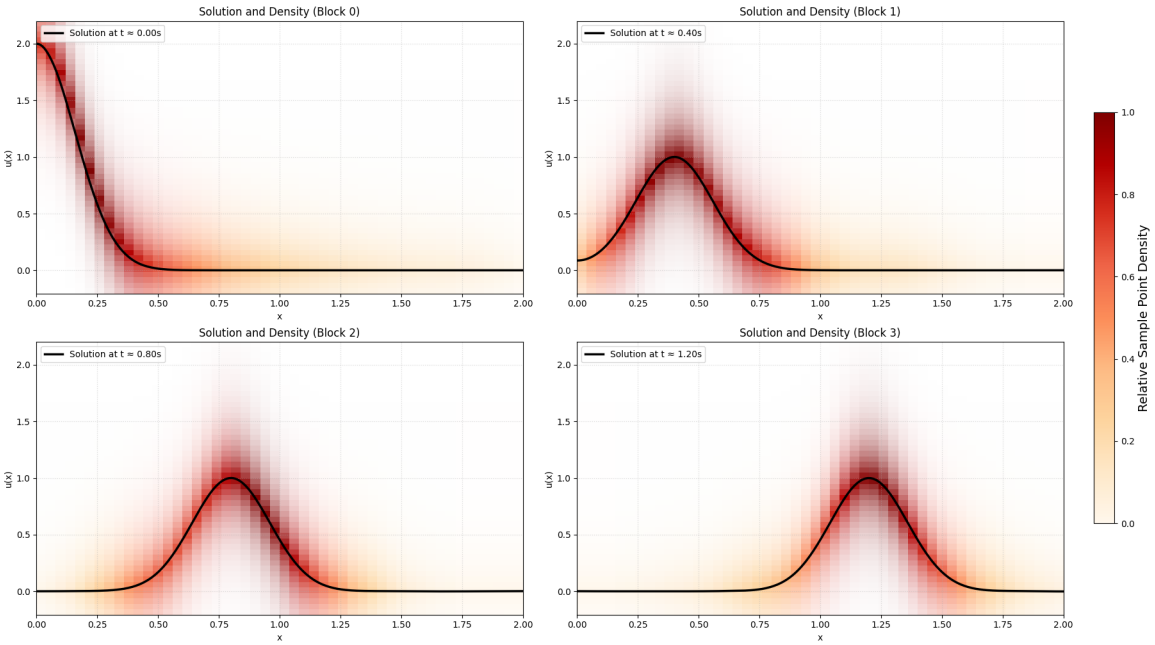


Figure 3.14: Solution and sampling density across each time block with the solution

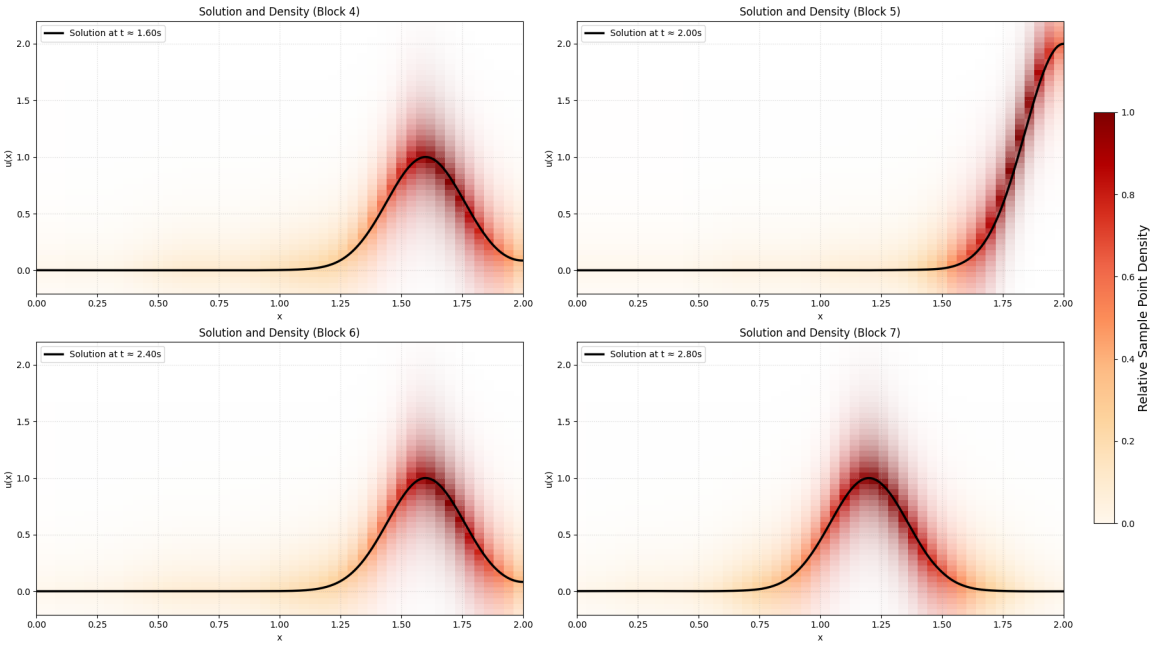


Figure 3.15: Solution and sampling density across each time block with the solution

3.2.3 2D Wave

This subsection investigates the numerical solution of the two-dimensional wave equation, a fundamental hyperbolic partial differential equation that models phenomena such as vibrating membranes, sound waves, and the propagation of light. The specific problem is defined over a unit square domain with homogeneous Neumann boundary conditions.

Problem Setup

The two-dimensional linear wave equation, with displacement $u(x, y, t)$ of a point (x, y) at time t is given by:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{3.9}$$

where c represents the wave propagation speed. For this specific problem, the wave speed is set to $c = 1$.

The problem is solved over a unit square spatial domain Ω and a finite time interval.

$$\begin{aligned} \text{Spatial Domain:} \quad & \Omega = \{(x, y) \mid 0 \leq x \leq 1,\ 0 \leq y \leq 1\} \\ \text{Temporal Domain:} \quad & t \in [0, 4] \end{aligned}$$

Initial and Boundary Conditions

The initial displacement $u(x, y, 0)$ and initial velocity $\frac{\partial u}{\partial t}(x, y, 0)$ are defined as:

$$u(x, y, 0) = \cos(\pi x) \cos(\pi y) \tag{3.10}$$

$$\frac{\partial u}{\partial t}(x, y, 0) = 0 \tag{3.11}$$

Homogeneous Neumann boundary conditions is imposed on all the walls of the spatial domain.

Exact Solution

For the specific setup described above, the exact solution is given by:

$$u(x, y, t) = \cos(\pi x) \cos(\pi y) \cos(\omega t) \tag{3.12}$$

where ω is the angular frequency. This analytical form serves as a benchmark against which the accuracy of the numerical solution is evaluated.

Numerical Results and Discussion

The wave equation detailed above was solved using our solver, and the results were then compared against the exact solution to validate the accuracy of the simulation. Figure 3.16 presents snapshots of the predicted solution alongside the exact solution at different points in time. A comparison reveals that the predicted results are in close agreement with the analytical solution across the full temporal domain. This confirms that the model accurately captures the essential wave dynamics..

Achieving this level of accuracy required a significant update to the domain resampling function in order to handle 2D space. The original 1D inverse transform sampling method, which relies on inverting a cumulative distribution function (CDF), does not generalize well, as multi-dimensional CDF inversion is computationally complex. We therefore implemented a 2D-specific acceptance-rejection sampling strategy. This approach generates uniform candidate points across the domain and probabilistically accepts them based on the gradient magnitude of the nearest existing sample point (found using a KD-Tree). This method allowed us to effectively concentrate points in regions with high spatial gradients, which was critical for accurately resolving the complex wave patterns and second-order derivatives in the 2D wave equation. A detailed description of this 2D implementation is provided in section 3.1.3.

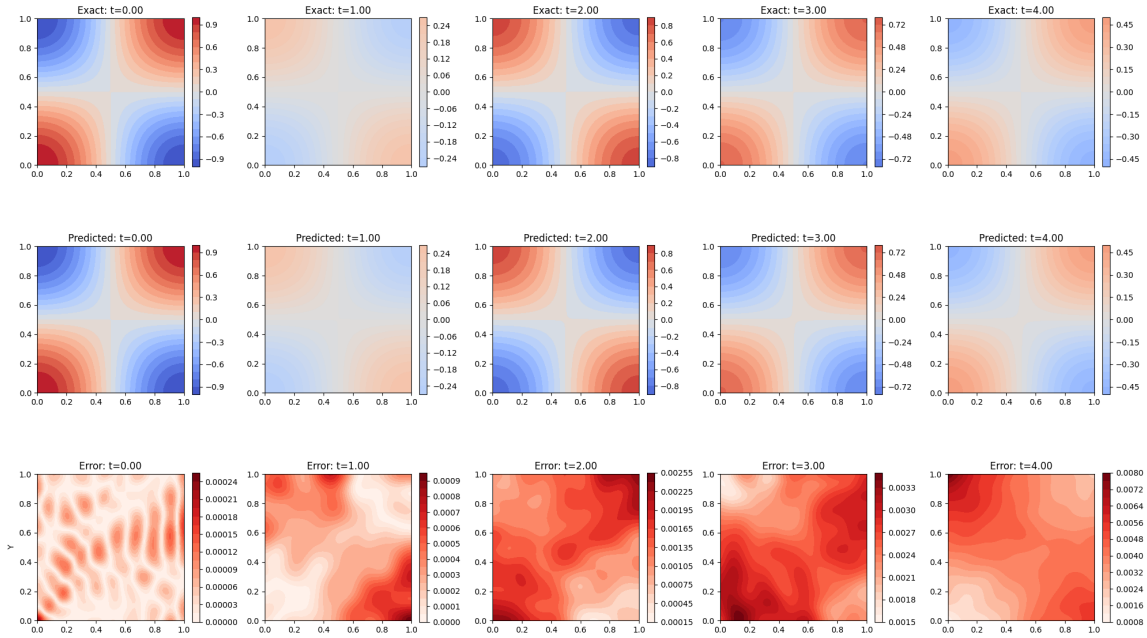


Figure 3.16: Comparison of the predicted solution and the exact solution at different time steps with `n_basis` = 100 and `n_time_blocks` = 8. The close agreement between the plots visually confirms the high accuracy of our solver.

3.2.4 Droplet

This subsection details the mathematical formulation of the two-dimensional wave propagation problem with a Gaussian pulse.

Problem Setup

The phenomenon is modeled by the two-dimensional linear wave equation, which is identical to eq. (3.9). The simulation is conducted over a square spatial domain, denoted by Ω, and for a finite time interval.

- **Spatial Domain:** The domain Ω is defined as the Cartesian product of two closed intervals:
$$\Omega = [0, 5] \times [0, 5] = \{(x, y) \in \mathbb{R}^2 \mid 0 \leq x \leq 5,\ 0 \leq y \leq 5\}$$
- **Temporal Domain:** The simulation evolves over the time interval $t \in [0, 4]$.

Initial and Boundary Conditions

To obtain a unique solution, the state of the system at the initial time ($t = 0$) must be specified. This requires defining both the initial displacement and the initial velocity of the wave. The initial configuration is a Gaussian pulse centered at $(1, 1)$ [4]:

$$u(x, y, 0) = 0.2 \exp\left(-\frac{(x-1)^2 + (y-1)^2}{0.1}\right)$$

The system is assumed to start from rest, meaning the initial velocity across the entire domain is zero:

$$\frac{\partial u}{\partial t}(x, y, 0) = 0$$

The interaction of the wave with the boundaries of the domain $\partial\Omega$ is specified by homogeneous Neumann boundary conditions. These conditions imply that the wave reflects off the boundaries without any loss of energy, and the normal derivative of the displacement function is zero at the boundary.

$$\left.\frac{\partial u}{\partial n}\right|_{\partial\Omega} = \left.\nabla u \cdot \mathbf{n}\right|_{\partial\Omega} = 0$$

Numerical Results and Discussion

We demonstrate the effectiveness of our methodology, which is significantly enhanced by the warm start discussed in section 3.2.2 and the PDF for resampling presented in section 3.1.4.

The primary result of our simulation is the successful prediction of the 2D wave propagation with a Gaussian pulse over time, as depicted in fig. 3.17. The solver accurately

captures the initial circular wave and its subsequent evolution, including its reflection off the domain boundaries.

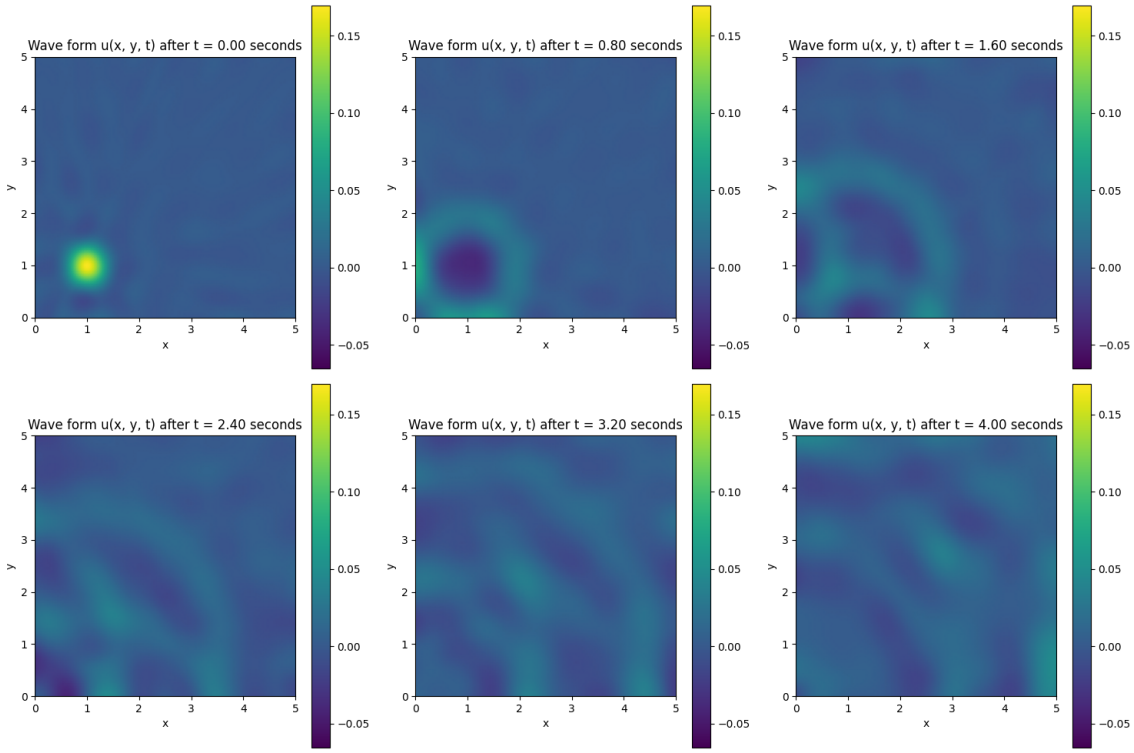
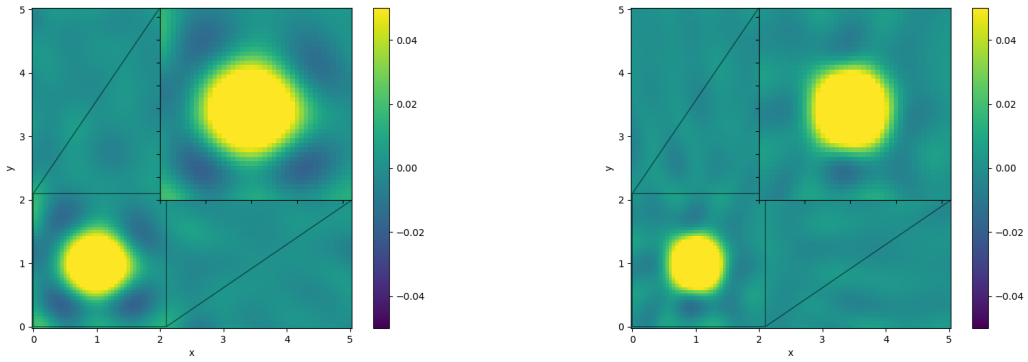


Figure 3.17: Predicted solution across multiple timestamps with `n_basis` = 200 and `n_time_blocks` = 10

The warm start mechanism is crucial for avoiding the spurious oscillations seen in fig. 3.18, a problem analogous to the Gibbs phenomenon [37]. The initial condition is a sharply localized Gaussian pulse, which is zero almost everywhere. Without the warm start approach, a uniform sampling is used, and attempts are made to approximate this localized feature with a set of globally distributed basis functions; however, as shown, a mismatch occurs. This mismatch is a known challenge that can cause the basis functions to ring or interfere with each other, resulting in high-frequency noise around the pulse, as shown in the plot.

The warm start solves this by leveraging the data-dependent nature of SWIM. By using the gradient of the initial condition to concentrate sampling points only in the small region where the pulse exists, it builds basis functions that are also localized to that specific area. This is the same principle used to resolve sharp shocks in the Burgers' equation, where basis functions are placed only "where it matters" [10]. As a result, we obtain a clean initial state that accurately represents both the pulse and a better surrounding flat, zero-

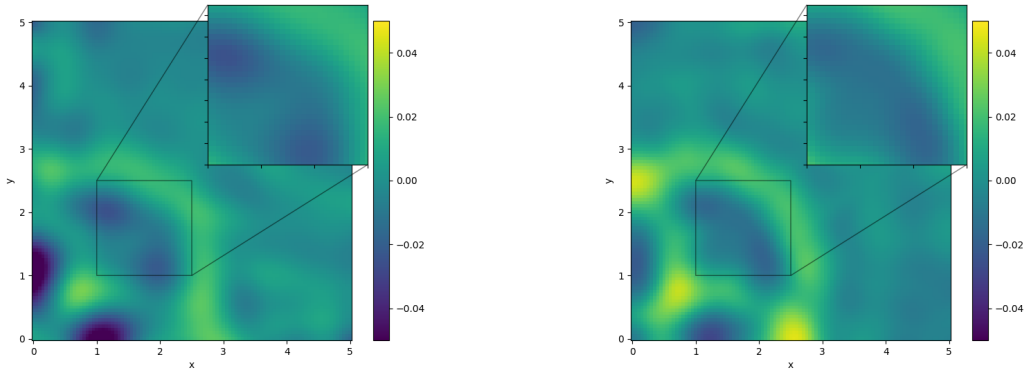
field environment, providing a much more stable and accurate foundation for the time evolution. This initial setup is important, as it also helps mitigate error propagation to some extent, as shown in fig. 3.19, where the wave front is much cleaner with the warm start.



(a) Without warm start

(b) With warm start

Figure 3.18: Predicted solution at t=0s



(a) Without warm start

(b) With warm start

Figure 3.19: Predicted solution at t=1.60s

The robustness of the adaptive sampling was enhanced by developing a new PDF that incorporates a Gaussian filter to smooth the gradient field as previously discussed section 3.1.4. This approach prevents overfitting to noisy gradients and ensures computational resources are efficiently allocated to the most critical regions of the propagating wavefronts.

The fig. 3.20 illustrates the sample points along with the solution; here, we can clearly see that our sample points are correctly placed from the outset. The points in block 0 that are not in the high gradient zone are a direct consequence of the fallback mechanism discussed in resampling logic section 3.1.3. If the primary method falls short, the code calculates the remaining number of required particles and generates them using simple uniform sampling across the domain. Therefore, the final sampling point set is a composite: the large, dense cluster is the result of successful acceptance-rejection sampling, while the sparse points across the domain are added uniformly through fallback to meet the total sample point count.

In fig. 3.21 we can see how the PDF develops compared to the solution. We notice that it always factors in a bit more area to place the sample points in, as we have previously discussed.

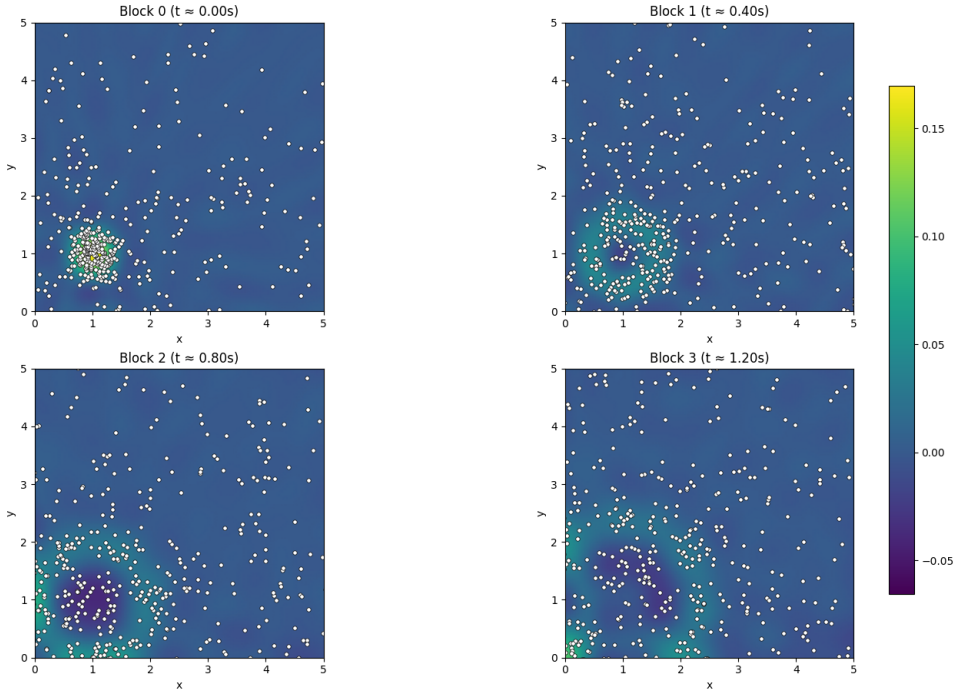


Figure 3.20: Predicted solution and sample points across different time blocks. White spots denote the sample points of our solver.

In summary, the combination of the warm start and the enhanced resampling PDF allows our solver to accurately and efficiently handle the complexities of the 2D wave equation. The warm start ensures a high-quality initial state, while the robust adaptive resampling maintains accuracy throughout the simulation by focusing computational effort precisely where it is needed.

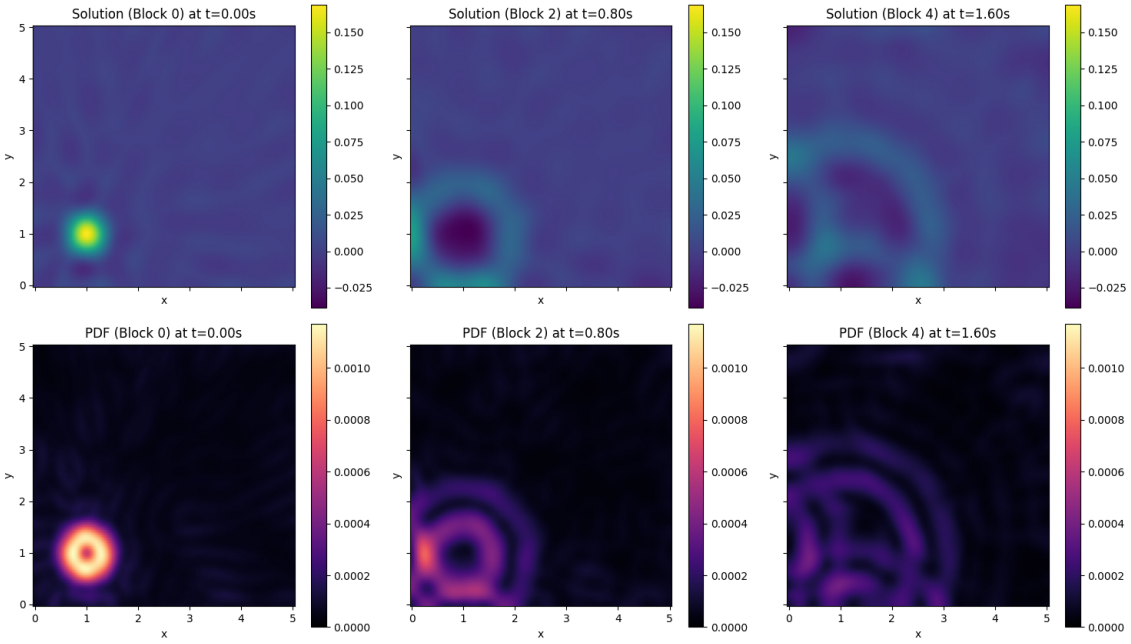


Figure 3.21: Predicted solution and PDF evolution over time.

4 Conclusion

4.1 Summary

In this thesis, we focused on enhancing the `swimpde` framework to create a more robust adaptive basis function for second-order time-dependent PDEs, with a specific focus on the wave equation in 1D and 2D. We addressed key limitations in regards to resampling in the standard solver by implementing several core extensions. We provide a thorough discussion of the implementations in section 3.1. These enhancements were validated against analytical solutions, where 1D experiments confirmed the new PDF's success in preventing error amplification, and a 2D droplet experiment demonstrated that the warm start and 2D adaptive sampling accurately capture complex wavefront propagation section 3.2.

4.2 Discussion

The results of this thesis provide significant insights into the practical application of the `swimpde` framework for hyperbolic PDEs. The most critical finding is the instability of naive, gradient-based resampling for oscillatory solutions. As demonstrated in the 1D reflecting wall experiment (section 3.2.2), a Probability Density Function based directly on gradient magnitudes can overfit to high-frequency numerical noise in the gradients. This leads to a feedback loop where errors are amplified, particularly upon interaction with boundaries. The introduction of a Gaussian filter in the PDF section 3.1.4 proved to be an effective solution, smoothing the sampling distribution and allowing the solver to focus on the wave's propagation. This highlights a broader principle: for wave-like problems, a smoothed or blurred importance sampling metric is more stable than a sharp, heuristic one.

The correction of the state transfer mechanism section 3.1.2 confirms that a naive application of resampling, which only considers the solution field u, is physically and mathematically incorrect for second-order problems. The explicit transfer of time derivatives (u_t) is non-negotiable for maintaining continuity and accuracy.

In two dimensions, the acceptance-rejection sampling section 3.1.3 proved to be a flexible and effective strategy for 2D problems. The 2D droplet simulation section 3.2.4 successfully combined all the thesis contributions, demonstrating that the warm start, 2D sampling, and robust PDF create a solver capable of handling complex, evolving wave patterns.

In conclusion, this work transforms the `swimpde` framework from a promising but limited tool into a practical and robust method for simulating wave propagation. By addressing the specific challenges of second-order state, 2D sampling, and oscillatory solutions, we have developed a solver that is both efficient and numerically stable.

4.3 Future Work

While this thesis successfully extended the `swimpde` framework, it also opens several directions for future research. The acceptance-rejection method was effective in 2D, but its efficiency may degrade in 3D due to the curse of dimensionality. Future work could explore more advanced high-dimensional sampling techniques, such as Markov Chain Monte Carlo (MCMC) methods, to enable efficient 3D adaptive sampling. Also, the 2D acceptance-rejection sampling and the nearest-neighbor search (via KD-Tree) can become bottlenecks. These components could be heavily optimized, for instance, by porting the sampling algorithms to run on a GPU.

The solver was validated on the linear wave equation. A logical next step is to apply it to more complex hyperbolic systems, such as nonlinear wave equations (e.g., shallow water equations), or problems in acoustics with varying media. The mesh-free nature of the framework is one of its primary advantages. Future research could leverage this by applying the resampling methods of the solver to problems involving complex, non-rectangular geometries, such as wave scattering around obstacles, which are challenging for traditional mesh-based methods.

Bibliography

[1] A.D. Pierce. *Acoustics – An Introduction to its physical principles and applications*. Acoustical Society of America, 1991.

[2] A. Aimi, G. Di Credico, H. Gimperlein, and C. Guardasoni. Adaptive time-domain boundary element methods for the wave equation with neumann boundary conditions. *Computers Mathematics with Applications*, 198:196–213, 2025.

[3] Marsha J. Berger and Randall J. LeVeque. Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems, 1998.

[4] Sacha Binder. Wave equation simulations 1d/2d (équation de d'alembert), 2021.

[5] Erik Lien Bolager, Iryna Burak, Chinmay Datar, Qing Sun, and Felix Dietrich. Sampling weights of deep neural networks, 2023.

[6] Chris J. Budd, Weizhang Huang, and Robert D. Russell. Adaptivity with moving grids. *Acta Numerica*, 18:111–241, 2009.

[7] Tim Bürchner, Philipp Kopp, Stefan Kollmannsberger, and Ernst Rank. Immersed boundary parametrizations for full waveform inversion. *Computer Methods in Applied Mechanics and Engineering*, 406:115893, 2023.

[8] Salvatore Cuomo, Vincenzo Schiano di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what's next, 2022.

[9] Jean le Rond d'Alembert. Recherches sur la courbe que forme une corde tenduë mise en vibration. *Histoire de l'Académie Royale des Sciences et Belles-Lettres de Berlin*, 3:214–219, 1747.

[10] Chinmay Datar, Taniya Kapoor, Abhishek Chandra, Qing Sun, Erik Lien Bolager, Iryna Burak, Anna Veselovska, Massimo Fornasier, and Felix Dietrich. Fast training of accurate physics-informed neural networks without gradient descent, 2025.

[11] Luc Devroye. *Non-Uniform Random Variate Generation(originally published with*. Springer, 1986.

[12] J. R. Dormand and P. J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6:19–26, 1980.

[13] Leonhard Euler. Supplément aux recherches sur la propagation du son. *Mémoires de l'Académie Royale des Sciences et Belles-Lettres de Berlin*, 15:185–209, 1766.

[14] Eric S Fraga and John Ll Morris. An adaptive mesh refinement method for nonlinear dispersive wave equations, 1992.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, August 2018.

[17] Jiequn Han, Arnulf Jentzen, and Weinan E. A brief review of the deep bsde method for solving high-dimensional partial differential equations, 2025.

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[19] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489–501, 2006. Neural Networks.

[20] Weizhang Huang and Robert D. Russell. *Adaptive Moving Mesh Methods*. Springer, 2010.

[21] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning, June 2021.

[22] Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks, 2021.

[23] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[24] Michael P. Lamoureux. The mathematics of pdes and the wave equation, 2006.

[25] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021.

[26] Changna Lu, Weizhang Huang, and Jianxian Qiu. An adaptive moving mesh finite element solution of the regularized long wave equation. *Journal of Scientific Computing*, 74(1):122–144, April 2017.

[27] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[28] Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[29] Luca Martino, David Luengo, and Joaquín Míguez. *Accept–Reject Methods*. Springer International Publishing, Cham, 2018.

[30] D. Morin. *Waves*.

[31] Philip M. Morse and Herman Feshbach. *Methods of theoretical physics*. 1955.

[32] Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. Efficient training of physics-informed neural networks via importance sampling. *Computer-Aided Civil and Infrastructure Engineering*, 36(8):962–977, 2021.

[33] Shu-Mei Qin, Min Li, Tao Xu, and Shao-Qun Dong. A-wpinn algorithm for the data-driven vector-soliton solutions and parameter discovery of general coupled nonlinear equations. *Physica D: Nonlinear Phenomena*, 443:133562, January 2023.

[34] 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*, 378:686–707, 2019.

[35] Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, and Madeleine Udell. Challenges in training pinns: A loss landscape perspective, 2024.

[36] C.P. Robert and G. Casella. *Monte Carlo statistical methods*. Springer, 2004.

[37] M. ten Eikelder, S. Stoter, Y. Bazilevs, and D. Schillinger. Constraints for eliminating the gibbs phenomenon in finite element approximation spaces, 2023.

[38] Jan Trynda, Paweł Maczuga, Albert Oliver-Serra, Luis Emilio García-Castillo, Robert Schaefer, and Maciej Woźniak. An h-adaptive collocation method for physics-informed neural networks. *Journal of Computational Science*, 91:102684, 2025.

[39] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020.

[40] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[41] Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403:115671, January 2023.

[42] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals, Sixth Edition*. Butterworth-Heinemann, 6 edition, May 2005.