Indian Academy of Sciences, Bengaluru
Indian National Science Academy, New Delhi
The National Academy of Sciences India, Prayagraj

SUMMER RESEARCH FELLOWSHIPS — 2025

Format for the final Report *,^

Name of the candidate	:	AYUSH DUTTA
Application Registration no.	:	ENGS307
Date of joining	:	13/05/2025
Date of completion	:	07/07/2025
Total no. of days worked	:	56
Name of the guide	:	Dr. NAVANEETHA KRISHNAN RAVICHANDRAN
Guide's institution	:	IISc, BENGALURU
Project title	:	Adaptive Neural Galerkin Technique for High-Dimensional Evolutionary Systems

Address with pin code to which the certificate could be sent:

House no: 2/17, Road No: 2/4, Baghajatin Colony, P.O-Pradhan Nagar, Siliguri, Pin-734003

E-mail ID: dutta7759@gmail.com

Phone No: 8972851567

TA Form attached with final report : YES ✓ NO ______

If, NO, Please specify reason

Ayush Dutta
Signature of the candidate
Date: 07/07/2025

[illegible]
Signature of the guide
Date: 07/07/2025

IMPORTANT NOTES:

*** This format should be the first page of the report and should be stapled with the main report. The final report could be anywhere between 20 and 25 pages including tables, figures etc.**

^ The final report must reach the Academy office within 10 days of completion. If delayed fellowship amount will not be disbursed.

(For office use only; do not fill/tear)

Candidate's name:		Fellowship amount:
Student:	Teacher:	Deduction:
Guide's name:		TA fare:
KVPY Fellow:	INSPIRE Fellow:	Amount to be paid:
PFMS Unique Code:		A/c holder's name:
Others		

Indian Academy of Sciences, Bengaluru
Indian National Science Academy, New Delhi
The National Academy of Sciences, India, Allahabad

SUMMER RESEARCH FELLOWSHIP PROGRAMME 2025

FINAL REPORT

Adaptive Neural Galerkin Technique for High-Dimensional Evolutionary Systems

Under the supervision of -

Dr. Navaneetha Krishnan Ravichandran

Assistant Professor

Department of Mechanical Engineering

IISc Bangalore

Submitted By -

Ayush Dutta

Summer Research Fellow

Application No - ENGS307

IIEST Shibpur

Acknowledgement

I would like to express my heartfelt gratitude to **Dr. Navaneetha Krishnan Ravichandran**, Assistant Professor, Department of Mechanical Engineering, Indian Institute of Science (IISc), Bangalore, for giving me the opportunity to carry out my summer internship at the **Nanoscale Energy Transport Lab (NETLab)**.

His expert guidance, valuable feedback, and continuous support have played a crucial role in enriching my understanding of nanoscale energy transport. The intellectually stimulating environment of NETLab helped me develop strong research skills and deepened my curiosity in the field.

I would also like to acknowledge the role of the **Indian Academy of Sciences**, **Indian National Science Academy**, and **The National Academy of Sciences, India** for organizing the **Summer Research Fellowship Programme (SRFP) 2025**, through which I received this wonderful opportunity.

Lastly, I gratefully acknowledge the online resources, research publications, and academic references that supported my project and helped shape my understanding. These resources contributed significantly to the successful completion of my work.

Thank you to everyone who contributed directly or indirectly to this learning experience.

Index

Abstract .... Page 2
1. Introduction .... Page 2
2. Problem Setup .... Pages 3–4
3. The Dirac-Frenkel Variational Principle .... Page 5
4. Neural Galerkin Scheme with Adaptive Learning .... Pages 5–10
5. Neural Architectures .... Pages 11–13
6. Numerical Experiments .... Pages 14–23
7. Extended Formulation – Degree Two Equation .... Pages 23–31
8. Conclusion .... Page 31
9. References .... Page 32

Adaptive Neural Galerkin Techniques for High-Dimensional Evolutionary Systems

Final Report by Ayush Dutta

May 2025 – July 2025

Abstract

High-dimensional problems, such as those in used to describe the dynamics of a motion, can be precisely solved by deep neural networks. However, they require high-quality training data, which is frequently difficult to obtain, in order to perform this effectively. In order to solve high-dimensional partial differential equations (PDEs), this work presents a technique called Neural Galerkin, which creates its own training data through active learning. It accomplishes this by gradually learning over time and determining when and where to gather new data using a physics-based concept known as the Dirac-Frenkel principle. Neural Galerkin learns as it goes, guided by how the solution changes, in contrast to other machine learning techniques that train all at once and don't take training data collection into account.This improves its ability to solve complicated problems where features change in particular places, such as particle systems or wave motion.

1 Introduction

Solving partial differential equations (PDEs) is a major task in science and engineering. These equations basically help in describing the dynamics of many physical or mechanical phenomenon which change over time and space like heat, waves, interaction between particles, etc. Tradition methods like the finite difference method and several other classical tools such as the classical Galerkin method are used generally used for solving PDEs and have worked well with many problems specially in low dimensions but once the number of variables increase – they begin to fail. When we try to simulate systems that involve many variables (or dimensions), traditional methods for solving PDEs (partial differential equations) become too slow or even impossible to use — this problem is known as the **curse of dimensionality**, where the computational requirements as well as the memory requirements grow exponentially.For instance, equations like Schrödinger's (used in quantum mechanics) or kinetic equations like the Fokker-Planck and Boltzmann equations become extremely high-dimensional if they describe the behavior of several interacting particles. If there are more than just a few particles, the number of dimensions increases rapidly, and even advanced numerical methods like **multigrid**, **fast multipole**, or **adaptive mesh refinement** techniques are not enough to handle them.

To deal with such high-dimensional problems, we need a new kind of approach that doesn't rely on traditional grid-based discretization. Instead of using a grid (which becomes impractical in high dimensions), we need methods that can **adaptively represent the solution** in a smarter way.

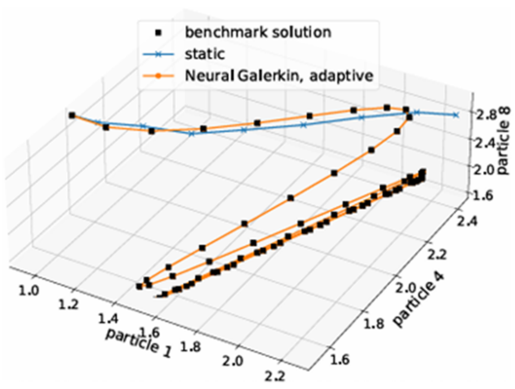


Figure 1: *Particle in a harmonic Trap (An example of High Dimensional PDE)*

In machine learning, this kind of adaptivity is often achieved using **deep neural networks (DNNs)**, which are good at automatically learning the important patterns or "features" in complex, high-dimensional data. This makes them suitable for representing solutions to high-dimensional PDEs in a more efficient way.

That's where the Neural Galerkin approach comes into picture The Neural Galerkin method is a contemporary technique that integrates deep learning— specifically, neural networks—with the Galerkin principle. Simply put, it substitutes a trainable neural network that can recognize the shape of the solution as it changes over time for the fixed basis functions (such as sin, cos, or polynomials) used in traditional Galerkin methods. The Neural Galerkin method uses a neural network with weights and biases (parameters) that vary over time, rather than fixed basis functions. Under the guidance of a physical principle known as the Dirac-Frenkel variational principle, these parameters are not optimized once but rather evolve gradually over time. In summary, our approach takes advantage of adaptivity in both approximating the solution and selecting sampling points.

2 Problem Setup

This section describes the type of math problems we aim to solve using deep learning. We concentrate on partial differential equations (PDEs) that change over time and space, such as those involving heat, fluid, or probability. Another name for these is evolution equations. We are interested in solving equations which have these characteristics:

- Localized areas of the solution may change over time.
- There are many spatial dimensions.

Now taking about Evolution equations. Let's say we have some space X, and we're interested to know how a quantity u(t, x) changes:

- t is time (from 0 to ∞),
- x is a position in the space X,
- $u(t, x)$ is the value of what we're studying (like temperature at time t and position x).

The evolution of u is described by this equation:

$$\frac{\partial u}{\partial t}(t, x) = f(t, x, u), \quad u(0, x) = u_0(x)$$

This means:

- The rate of change of u over time is controlled by a function f,
- We are also given an initial condition $u_0(x)$, which tells us what u looks like at time $t = 0$.

Depending on how we define the function f, this general equation can represent many different physical models. For example, if f includes terms for movement (advection), spreading (diffusion), or reaction (like growth or decay), it becomes an **advection-diffusion-reaction equation**.

We assume proper boundary conditions are used, which means the solution:

1. Exists,
2. Is unique,
3. Changes smoothly when we change the starting point $u_0(x)$.

We aim to solve time-dependent PDEs using deep neural networks.

Instead of traditional methods (like finite differences), we approximate the solution $u(t, x)$ with a neural network $\tilde{u}(t, x; \theta)$, where θ are the network's parameters. The inputs are time t and space x; the output approximates the solution.

To train the network, we define the residual:

$$r(t, x; \theta) = \frac{\partial \tilde{u}}{\partial t}(t, x; \theta) - f(t, x, \tilde{u})$$

This measures how well the network satisfies the PDE. If $r = 0$ everywhere, the network solves the PDE exactly.

We train the network by minimizing the mean squared residual:

$$\min_{\theta} \mathbb{E}_{t,x} \left[r(t, x; \theta)^2 \right]$$

Since computing the exact expectation is infeasible, we use Monte Carlo sampling: draw n random points (t_i, x_i) and minimize

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} r(t_i, x_i; \theta)^2$$

However, in high dimensions, random sampling often misses regions where the solution has important variations (localized features). As a result, the network may fail to learn the correct solution—even if it's capable of representing it.

This is a key challenge of the **curse of dimensionality**, where the number of required samples grows exponentially with the number of dimensions.

3 The Dirac-Frenkel variational principle

When finding the precise solution is too difficult, the **Dirac–Frenkel variational principle** can be used to approximate how a quantum system changes over time. It employs a more straightforward set of potential solutions that rely on a few parameters rather than attempting to solve the entire complex equation. The principle decides how these parameters should alter at each instant to ensure that the approximate solution as closely resembles the actual quantum behavior as feasible. This is accomplished by projecting the precise change onto the space of all potential changes permitted by the selected simpler solutions. This approach preserves crucial physical characteristics, such as energy conservation, while simplifying the solution of challenging quantum problems. The Dirac–Frenkel variational principle says: **At every moment, pick the direction of change of your approximate solution that makes the error (difference from the true solution) as small as possible, in a specific "projected" way.**

If your exact solution $u(t)$ satisfies

$$\frac{d}{dt}u(t) = F(u(t)),$$

but you approximate it by $\tilde{u}(t;\theta(t))$, where $\theta(t)$ are time-dependent parameters, then the Dirac–Frenkel principle requires

$$\left\langle \delta\tilde{u}, \frac{d}{dt}\tilde{u} - F(\tilde{u}) \right\rangle = 0$$

for all variations $\delta\tilde{u}$ in the approximation space. Here, $\langle\cdot,\cdot\rangle$ is an inner product measuring the difference.

This means the difference between the time derivative of the approximation and the true dynamics $F(\tilde{u})$ is orthogonal (perpendicular) to all possible variations in the approximation space, which leads to the best possible approximation in that space.

The Dirac-Frenkel variational principle is a foundational concept in Neural Galerkin schemes for solving high-dimensional evolution equations.

4 Neural Galekin Scheme with Adaptive Learning

We consider a time-dependent partial differential equation (PDE) for a function $u(x,t)$ defined on a spatial domain $\mathcal{X}$ and time domain $[0,\infty)$, given by:

$$\partial_t u(x,t) = f(x,t,u), \quad u(x,0) = u_0(x)$$

Here, f is the source term and $u_0(x)$ is the initial condition.

To solve this PDE, we use a parametric representation of the solution, where the solution $u(x,t)$ is approximated by a nonlinear function $\hat{u}(x,\theta(t))$. This is expressed as:

$$u(x,t) = \hat{u}(x,\theta(t))$$

This approach corresponds to the ansatz:

$$u(t,x) = U(\theta(t),x)$$

Here, $\theta(t)$ is a time-dependent parameter vector, and U is a function (e.g., a neural network) that maps parameters and space to the approximate solution. Time enters the representation only through the parameter $\theta(t)$, which allows the solution to carry information from previous time steps. The function U may be nonlinear in $\theta(t)$, unlike traditional linear model reduction techniques. Substituting this ansatz into the PDE and applying the chain rule gives the local-in-time residual:

$$r_t(x, \theta, \dot{\theta}) = \nabla_\theta \hat{u}(x, \theta) \cdot \dot{\theta} - f(x, t, \hat{u}(x, \theta))$$

This residual measures the mismatch between the time derivative implied by the parameter dynamics and the PDE's right-hand side.

The goal is to find the parameter evolution $\theta(t)$ that minimizes this residual. Since no training data is used, the parameter is learned directly from the PDE by minimizing the squared residual:

$$\dot{\theta}(t) \in \arg\min_{\eta \in \Theta} \int_{\mathcal{X}} |r_t(\theta, \eta, x)|^2 \, d\nu(x)$$

Here, ν is a positive measure on $\mathcal{X}$, and the optimization is done pointwise in time. This approach follows the Dirac–Frenkel variational principle, resulting in an initial value problem for $\theta(t)$ rather than a global-in-time optimization.

The initial parameter θ_0 is obtained by minimizing the mismatch between the initial condition and the parametric representation:

$$\theta_0 \in \arg\min_{\theta \in \Theta} \int_{\mathcal{X}} |u_0(x) - U(\theta, x)|^2 \, d\nu(x)$$

This formulation allows us to approximate the PDE solution by evolving a finite-dimensional parameter $\theta(t)$ over time while minimizing the residual error at each time step. By representing the solution $u(x, t)$ with a neural network $U(\theta(t), x)$, where $\theta(t)$ denotes time-dependent network parameters, the Neural Galerkin method provides a principled framework for solving time-dependent partial differential equations (PDEs). The central objective is to evolve the parameters $\theta(t)$ such that the neural network output closely approximates the true solution while minimizing the residual of the PDE at each time instance.

This formulation is grounded in the **Dirac–Frenkel variational principle**, which dictates that the temporal evolution of the approximate solution must lie within the tangent space of the trial manifold induced by the neural network. Concretely, the residual of the PDE is projected orthogonally to the space spanned by the parameter gradients $\nabla_\theta U$. This condition results in a system of **Euler–Lagrange equations** governing the parameter dynamics.

The resulting parameter evolution can be expressed as a system of ordinary differential equations:

$$M(\theta) \, \dot{\theta}(t) = F(t, \theta)$$

Mass matrix:

$$M(\theta) = \int_{\mathcal{X}} \nabla_\theta U(\theta, x) \otimes \nabla_\theta U(\theta, x) \, d\nu(x)$$

This term captures the geometry of the parameter space as influenced by the network architecture. The operator $\otimes$ represents the outer product.

Force vector:

$$F(t,\theta) = \int_{\mathcal{X}} \nabla_\theta U(\theta,x)\, f(t,x,U(\theta,x))\, d\nu(x)$$

This projects the PDE's right-hand side $f(t,x,u)$ onto the span of parameter gradients.$d\nu(x)$ is a measure over the spatial domain $\mathcal{X}$, which may be static or dynamically adjusted to improve approximation quality over time.

This ODE system evolves the parameters $\theta(t)$ such that the neural network remains as close as possible to the true PDE solution in a residual-minimizing sense. The Neural Galerkin method generalizes dynamic low-rank approximation strategies by incorporating nonlinear trial spaces defined by neural networks. It also connects with broader variational principles used in model reduction and scientific machine learning. By directly integrating the PDE structure into the training dynamics, the method enables data-free, physics-informed learning, making it suitable for high-dimensional, complex dynamical systems where traditional discretization methods are computationally prohibitive.

The variational equation leads to a system of ordinary differential equations (ODEs) for the neural network parameters $\theta(t)$. To solve this system numerically, time is discretized into steps t_k, where $t_{k+1} = t_k + \delta t_k$, and δt_k can vary and be chosen adaptively.

At each time step, the parameters $\theta_k \approx \theta(t_k)$ are updated using standard ODE integration methods — either **explicit** (like Euler or Runge–Kutta) or **implicit** methods. This turns training the network into solving the ODE system step by step over time.

Explicit Integrators

1. Forward Euler Method

The simplest explicit scheme is the Forward Euler method. At each time step t_k, the parameters are updated using:

$$M(\theta_k)\theta_{k+1} = M(\theta_k)\theta_k + \delta t_k F(t_k,\theta_k)$$

This update is linear in form and directly computes the next parameter state θ_{k+1} using known quantities from the current time step. In practice, the mass matrix $M(\theta_k)$ may be ill-conditioned or singular. To ensure numerical stability and avoid inversion issues, it is common to regularize the system as:

$$(M(\theta_k) + \lambda I)\theta_{k+1} = (M(\theta_k) + \lambda I)\theta_k + \delta t_k F(t_k,\theta_k)$$

2. Runge–Kutta–Fehlberg Method (RK45)

The RK45 method is an adaptive explicit integrator that adjusts the time step δt based on local error estimates. It is particularly effective for handling problems with rapid transients or varying time scales.

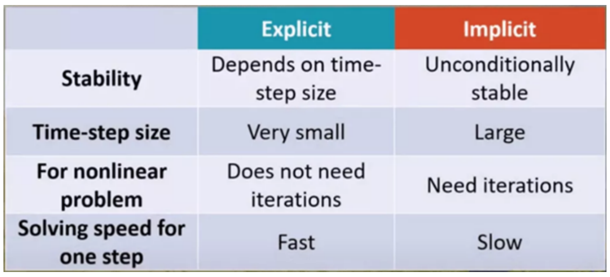
In the Neural Galerkin context, RK45 proceeds by evaluating the force vector $F(t,\theta)$ at several intermediate stages within the time step, combining them to produce a high-order accurate update for θ. Although more computationally expensive per step than Euler's method, its adaptive nature often results in better efficiency and stability over long integration horizons.

Implicit Integrators

1. Backward Euler Method For stiff problems, implicit methods provide enhanced stability. The Backward Euler scheme updates the parameters by solving the equation:

$$M(\theta_{k+1})\theta_{k+1} = M(\theta_{k+1})\theta_k + \delta t_k F(t_{k+1}, \theta_{k+1})$$

This formulation requires solving a nonlinear system at each step, since both the mass matrix and force vector depend on the unknown θ_{k+1}. Solution typically involves iterative methods such as Newton–Raphson, which may be computationally demanding but offer superior robustness in stiff regimes.



	Explicit	Implicit
Stability	Depends on time-step size	Unconditionally stable
Time-step size	Very small	Large
For nonlinear problem	Does not need iterations	Need iterations
Solving speed for one step	Fast	Slow

Figure 2: Comparison between explicit and implicit integration methods.

Alternatively, one could discretize the problem in time and then derive the Euler-Lagrange equations for this time-discrete optimization problem, rather than first deriving the variational formulation in continuous time and then discretizing the Euler-Lagrange equations. This strategy is comparable to the distinction between the "optimize-then-discretize" and "discretize-then-optimize" approaches, which are frequently employed in inverse problems and PDE-constrained optimization.

The operators $M(\theta)$ and $F(t, \theta)$, which appear in the neural-network-based approximation of solutions to partial differential equations (PDEs), are now the focus of efficient estimation using an **active learning strategy** for Neural Galerkin schemes. The core concept is to enhance the accuracy of Monte Carlo estimations by adapting the sampling probability measure in response to the neural network parameters θ as they evolve over time. For general parametrizations of the solution $U(\theta, x)$, the integrals defining $M(\theta)$ and $F(t, \theta)$ do not have closed-form expressions and must be estimated numerically.

In low-dimensional problems, these integrals can be approximated by quadrature on a grid. However, in high dimensions, quadrature becomes infeasible due to the curse of dimensionality.

Instead, when the measure ν over the domain X is a probability measure, one can use Monte Carlo estimation: draw n samples $\{x_i\}_{i=1}^n$ independently from ν, and replace the integral by empirical averages over these samples. Although this estimator is efficient in approximating integrals over high-dimensional spaces, it performs poorly when the PDE solution $U(\theta)$ develops localized features, i.e., regions with sharp changes or concentrated mass.

To address this, the objective function

$$J_t(\theta, \eta) = \frac{1}{2}\int_X |r_t(\theta, \eta, x)|^2 \, d\nu(x)$$

is reformulated by allowing the measure ν to depend on θ, defining a parameter-dependent measure ν_θ:

$$J_\theta(\theta, \eta) = \frac{1}{2}\int_X |r_t(\theta, \eta, x)|^2 \, d\nu_\theta(x).$$

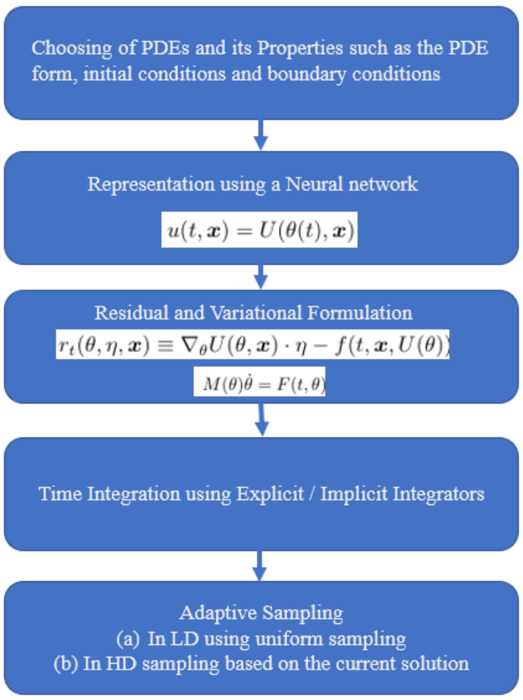

Choosing of PDEs and its Properties such as the PDE form, initial conditions and boundary conditions
Representation using a Neural network
$u(t, \boldsymbol{x}) = U(\theta(t), \boldsymbol{x})$
Residual and Variational Formulation
$r_t(\theta, \eta, \boldsymbol{x}) \equiv \nabla_\theta U(\theta, \boldsymbol{x}) \cdot \eta - f(t, \boldsymbol{x}, U(\theta))$
$M(\theta)\dot{\theta} = F(t, \theta)$
Time Integration using Explicit / Implicit Integrators
Adaptive Sampling
(a) In LD using uniform sampling
(b) In HD sampling based on the current solution

Figure 3: Steps to implement the Neural Galerkin approach using adaptive sampling and time integration to solve PDEs

Here, the residual $r_t(\theta, \eta, x)$ is weighted according to the distribution ν_θ, which evolves as the network parameters $\theta(t)$ change in time. This dynamic adjustment allows the sampling measure to focus on the important regions of the solution space, reducing variance in estimates.

Following this, the operators $M(\theta)$ and $F(t, \theta)$ become

$$M(\theta) = \int_X \nabla_\theta U(\theta, x) \otimes \nabla_\theta U(\theta, x) \, d\nu_\theta(x),$$

$$F(t, \theta) = \int_X \nabla_\theta U(\theta, x) \, f(t, x, U(\theta)) \, d\nu_\theta(x),$$

where f encodes the PDE dynamics, and $\nabla_\theta U(\theta, x)$ is the gradient of the neural network output with respect to parameters θ. To estimate $M(\theta(t))$ and $F(t, \theta(t))$, samples $\{x_i(t)\}_{i=1}^n$ are drawn from the time-dependent measure $\nu_{\theta(t)}$.

The Monte Carlo estimators are then given by

$$\tilde{M}(\theta(t)) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta U(\theta(t), x_i(t)) \otimes \nabla_\theta U(\theta(t), x_i(t)),$$

$$\tilde{F}(t, \theta(t)) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta U(\theta(t), x_i(t)) \, f(t, x_i(t), U(\theta(t))).$$

For these estimators to be effective, their variances should be low, which means that the sampling measure $\nu_{\theta(t)}$ must be chosen based on the current approximation $U(\theta(t))$.

For example, in problems where the PDE solution is a probability density function (such as the Fokker-Planck equation), it is reasonable to set the sampling measure

$$\nu_{\theta(t)} = U(\theta(t)).$$

This choice concentrates samples where the density is large, improving estimator accuracy.

In cases where the solution features moving localized fronts, the sampling measure could be proportional to the gradient magnitude

$$|\nabla_x U(\theta(t), x)|,$$

concentrating samples near those fronts. When direct sampling from ν_θ is difficult, importance sampling can be used.

Starting from a nominal measure ν, define a positive weight function

$$\omega : X \times \Theta \to (0, \infty),$$

with normalization constant

$$Z_\theta = \int_X \omega(x, \theta)\, d\nu(x) < \infty.$$

The adaptive measure is

$$d\nu_\theta(x) = \frac{\omega(x, \theta)}{Z_\theta}\, d\nu(x).$$

Samples $\{x_i\}$ are drawn from ν_θ and the estimators incorporate the weights:

$$\tilde{M}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta U(\theta, x_i) \otimes \nabla_\theta U(\theta, x_i)\, \omega(x_i, \theta),$$

$$\tilde{F}(t, \theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta U(\theta, x_i)\, f(t, x_i, U(\theta))\, \omega(x_i, \theta).$$

This technique enables adaptive sampling by adjusting the weights ω based on θ and maintains unbiased estimators. The method supports deep neural networks (DNNs) of arbitrary complexity for $U(\theta, x)$, allowing alignment between the network architecture and the adaptive sampling measure ν_θ.

A specific example is a shallow network with one hidden layer and Gaussian units:

$$U(\theta, x) = \sum_{i=1}^{m} c_i\, \phi(x, w_i, b_i),$$

where the parameter vector is

$$\theta = (c_i, w_i, b_i)_{i=1}^{m},$$

and the nonlinear Gaussian activation function is

$$\phi_G(x, w, b) = \exp\big(-w^2 |x - b|^2\big).$$

Here, c_i are coefficients, and w_i, b_i are features (weights and centers) of the Gaussian units.

This architecture is well-suited for approximating probability density functions like solutions to the Fokker-Planck equation. When

$$\nu_\theta = U(\theta),$$

sampling from the network $U(\theta)$ is efficient since it corresponds to sampling from a Gaussian mixture.

5 Neural Architectures

The number of layers, the kinds of neurons or units used, and the connections between them are all examples of the structure and design of neural networks, which are referred to as **neural architectures**. By regulating the *depth* (number of layers), *width* (number of neurons per layer), and *activation function type*, the architecture establishes the network's ability to learn and represent complex functions. Different architectures are better suited for different tasks. For instance, shallow networks with fewer layers are easier to train and faster, while deep networks with many layers can capture extremely complex patterns. Selecting the right neural architecture is essential for striking a balance between computational efficiency, model expressiveness, and optimization simplicity.

(a) Shallow Neural Architecture

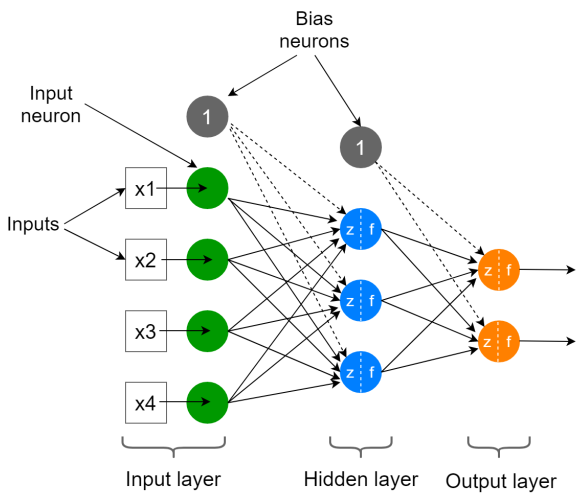


Figure 4: One Hidden Layer (Shallow) Neural Network Architecture

Neural network architectures are fundamental building blocks in approximating complex functions, especially when solving partial differential equations (PDEs) or modeling probability densities. The choice of architecture influences the network's ability to capture intricate patterns and adapt to specific problem structures. One common and simple architecture is the **shallow neural network**, which consists of a single hidden layer with a finite number of nodes.

The shallow architecture considered here comprises m nodes in the hidden layer, with the network output expressed as a weighted sum of nonlinear activation units:

$$U(\boldsymbol{\theta}, x) = \sum_{i=1}^{m} c_i \, \phi(x, w_i, b_i)$$

where $\boldsymbol{\theta} = (c_i, w_i, b_i)_{i=1}^{m}$ collects the network parameters. Specifically, $c_i, w_i \in \mathbb{R}$ are scalar parameters, and $b_i \in \mathbb{R}^d$ are vector-valued parameters representing the centers of the nonlinear units.

The activation function $\phi : \mathcal{X} \times \mathbb{R}^{d+1} \to \mathbb{R}$ applies a nonlinear transformation to the input and is chosen to suit the problem domain. Two common nonlinear units are used in this architecture:

Gaussian kernel:

$$\phi_G(x, w, b) = \exp\left(-w^2\|x - b\|^2\right)$$

which acts componentwise through the exponential function. This kernel is effective for capturing localized features and smooth variations.

Periodic Gaussian kernel:

$$\phi_{LG}(x, w, b) = \exp\left(-w^2 \left|\sin\left(\frac{\pi(x - b)}{L}\right)\right|^2\right)$$

used when the domain is periodic, i.e., $\mathcal{X} = L\mathbb{T}^d$ for some period $L > 0$. Here, the sine function operates componentwise on the vector $x - b$, enforcing periodic boundary conditions in the network output.

In this setting, the parameters c_i are referred to as the coefficients of the network, while w_i and b_i are its features.

This shallow architecture is flexible and computationally efficient, making it well-suited for tasks such as approximating solutions to PDEs or probability densities. Furthermore, its design can be aligned with the adaptive sampling measure $\nu_{\boldsymbol{\theta}}(t)$, improving training efficiency by focusing sampling efforts on important regions of the domain.

(b) Feedforward Neural Network

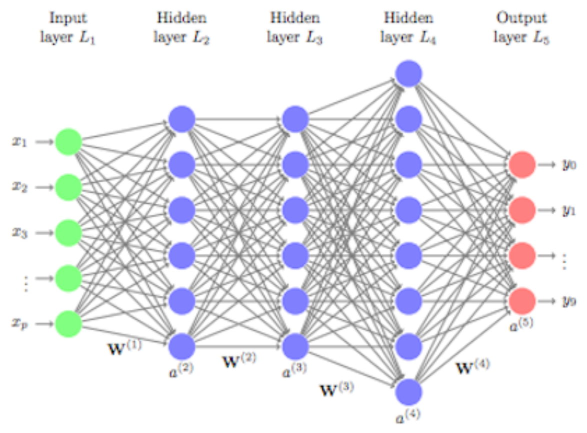


Figure 5: Diagram of a Feedforward Neural Network with Several Hidden Layers

We use a feedforward neural network with $\ell \in \mathbb{N}$ hidden layers and m nodes in each layer to approximate the solution. The function represented by the network is $U(\theta, x)$, where $\theta = (c, W_1, \ldots, W_\ell, b_1, \ldots, b_\ell)$ are the trainable parameters. Here, $c \in \mathbb{R}^m$ is a coefficient vector, $W_1 \in \mathbb{R}^{m\times d}$ connects the input to the first layer, $W_i \in \mathbb{R}^{m\times m}$ for $i > 1$ are the weight matrices for hidden layers, and $b_i \in \mathbb{R}^d$ are the bias vectors. The network applies a series of nonlinear transformations layer by layer. The output is computed as

$$U(\theta, x) = c(t)^T \tanh\left(W_\ell \tanh\left(W_{\ell-1}\left(\cdots \phi_L^{\tanh}(x, W_1, b_1)\cdots\right) + b_{\ell-1}\right) + b_\ell\right),$$

where the tanh activation is applied componentwise. The nonlinearity used in the first layer is defined as

$$\phi_L^{\tanh}(x, W, b) = \tanh\left(W \sin\left(\frac{2\pi(x-b)}{L}\right)\right),$$

which includes a sine transformation followed by the hyperbolic tangent. This special form helps the network better approximate periodic or oscillatory features in the data. Overall, the architecture is designed to be flexible and expressive, making it suitable for solving high-dimensional problems such as partial differential equations.

6 Numerical Experiments

6.1 Korteweg–de Vries Equation

The Korteweg–de Vries (KdV) equation is a third-order nonlinear partial differential equation given by:

$$\partial_t u + \partial_x^3 u + 6u\,\partial_x u = 0,$$

where $u = u(x,t)$ is a function of space and time. Here, $\partial_t u$ denotes the time derivative, $\partial_x^3 u$ the third-order spatial derivative, and $\partial_x u$ the first-order spatial derivative.

Reference Solution for the KdV Equation: Two-Soliton Interaction

To validate and benchmark the numerical simulation of the Korteweg–de Vries (KdV) equation, we use an exact analytical solution that models the interaction of two solitons. A *soliton* is a self-reinforcing solitary wave that maintains its shape while traveling at a constant speed. Solitons arise in certain nonlinear partial differential equations, such as the KdV equation, as stable wave solutions. Unlike ordinary wave packets that tend to disperse over time, solitons can interact with other solitons and emerge from the interaction unchanged in shape and speed, exhibiting particle-like behavior. This unique property makes solitons particularly significant in nonlinear wave theory and fluid dynamics.

We consider the one-dimensional KdV equation of the form:

$$\partial_t u + \partial_x^3 u + 6u\partial_x u = 0,$$

defined on the spatial domain $x \in [-20, 40) \subset \mathbb{R}$, with periodic boundary conditions.

The exact solution used for benchmarking is given by:

$$u(x,t) = 2\log_e f_{xx},$$

where the function f is defined as:

$$f = 1 + e^{\eta_1} + e^{\eta_2} + Ae^{\eta_1+\eta_2},$$

with the phase functions:

$$\eta_i = k_i x - k_i^3 t + \eta_i^{(0)}, \quad i = 1, 2,$$

and the interaction coefficient:

$$A = \left(\frac{k_1 - k_2}{k_1 + k_2}\right)^2.$$

Parameter Settings

To study the soliton dynamics, two different sets of initial parameters are considered:

Case 1:

$$k_1 = 1, \quad k_2 = \sqrt{2}, \quad \eta_1^{(0)} = 0, \quad \eta_2^{(0)} = 2\sqrt{2}$$

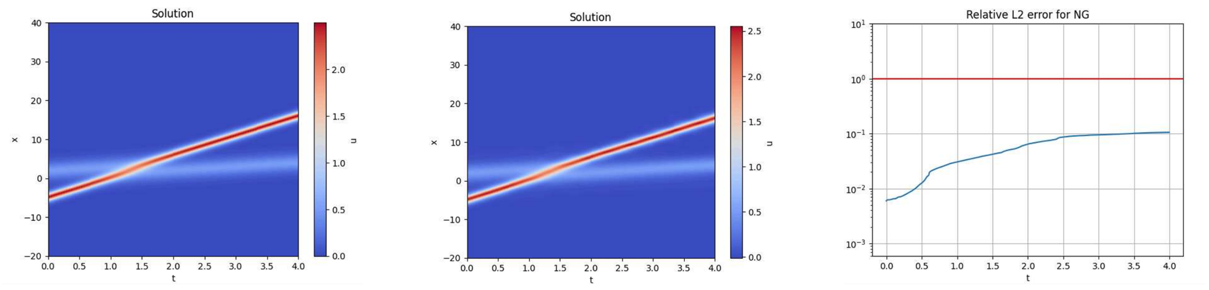
Case 2:

$$k_1 = 1, \quad k_2 = \sqrt{5}, \quad \eta_1^{(0)} = 0, \quad \eta_2^{(0)} = 10.73$$

These parameters result in two initially separated solitons moving toward each other with different velocities. As time evolves, the solitons interact nonlinearly, collide, and then re-emerge retaining their original shapes and velocities—a key characteristic of solitonic behavior.

Parametric Solution Using Adaptive Neural Galerkin Approach

- Use a **shallow neural network** , with **Gaussian units** , setting the number of nodes to $m = 10$.
- Since the spatial domain is one-dimensional, choose the integration measure as $d\nu_\theta(x) = dx$.
- Sample $n = 1000$ points **uniformly from the interval** $[0, 1]$ to estimate the matrices M and F.
- Obtain the initial parameter vector θ_0 via a **least-squares fit** to the initial condition.
- Use a **batch size of** 10^5 and run for 10^5 **iterations** during training.
- Draw samples **uniformly from the spatial domain** during training.
- Use a **learning rate of** 10^{-1}.
- Train **five replicates with different random initializations** and select the model with the **lowest test error** for robustness.
- Perform time integration of neural network parameters using the **Runge–Kutta 45 (RK45) method**, which adaptively selects the size of the time step based on the dynamics of the solution.



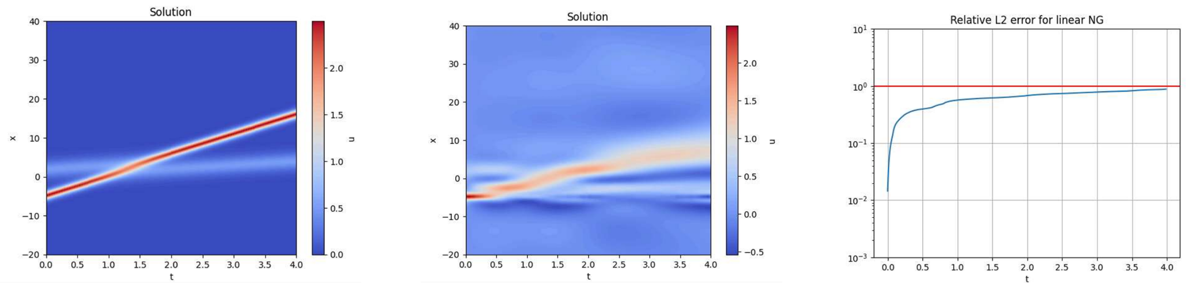
(a) Reference solution (b) Adaptive NG (c) Error over Time

Figure 6: Comparision of Reference Solution with Adaptive Neural Galekin Approach for Kdv Equation

We also examine a more basic version of the network in which the features remain constant throughout training to demonstrate the significance of adjusting nonlinear features. The network is the same in this more basic version, but there are 30 nodes rather than 10. The feature parameters w_i and b_i, which regulate the form and location of the network's Gaussian units, are fixed in this simplified version. While training, they don't update.

Either these fixed features are dispersed uniformly throughout space X, or they are selected to satisfy the initial condition and remain unchanged.

This method uses Gaussian functions distributed evenly across the domain Ω and is known as linear Galerkin with equidistant basis functions. Then, we plot the error over time to analyze the performance.



(a) Reference Solution (b) Linear NG (c) Error over time

Figure 7: Comparison of Reference Solution with Linear Neural Galerkin Approach for Kdv Equation

6.2 Allen-Cahn Equation

We consider the Allen-Cahn (AC) equation, a classical reaction-diffusion model, on the one-dimensional periodic domain $X = [0, 2\pi)$:

$$\frac{\partial u}{\partial t} = \epsilon \frac{\partial^2 u}{\partial x^2} - a(t,x)(u - u^3).$$

The parameter $\epsilon = 5 \times 10^{-2}$ controls the diffusion strength, which smooths spatial variations in u. The reaction term coefficient is time- and space-dependent, given by

$$a(t,x) = 1.05 + t\sin(x).$$

This coefficient shapes the system's dynamics: for early times $t < 1.05$, the reaction term uniformly drives the solution u towards the two stable states ± 1. When t exceeds 1.05, the spatial variation in $a(t,x)$ causes u to be driven toward ± 1 in regions where

$$\sin(x) < \frac{1.05}{t},$$

and towards zero where

$$\sin(x) > \frac{1.05}{t}.$$

Over long times ($t \to \infty$), these regions converge to the intervals $(0, \pi)$ and $(\pi, 2\pi)$, respectively, leading to spatial pattern formation.

The initial condition $u(0,x)$ is specified as the difference between two smooth localized profiles defined by

$$u(0,x) = \varphi_L^G\left(x, \sqrt{10}, \frac{1}{2}\right) - \varphi_L^G\left(x, \sqrt{10}, 4.4\right),$$

where φ_L^G is a Gaussian-type mollifier with parameter $L = \frac{1}{2}$. This provides a smooth initial profile that evolves under the combined effects of diffusion and nonlinear reaction.

Reference Solution

The benchmark solution is computed using a **finite-difference discretization** on 2048 equidistant grid points in the spatial domain. This means the spatial region of interest is divided into 2048 equally spaced points, allowing the solution to be represented with high resolution across the domain.

For the **time discretization**, a **semi-implicit Euler method** is used. In this scheme, the **linear operators** in the governing equations are treated implicitly, while the **nonlinear operators** are treated explicitly. This means that, at each time step, the linear terms are evaluated using the unknown future value, which improves stability, while the nonlinear terms are evaluated using the known current value, which simplifies the computation.

The **time-step size** is set to 10^{-5}, ensuring that the numerical solution can accurately capture rapid changes in the system, especially those associated with sharp gradients or interfaces.

The resulting solution is characterized by **relatively flat pieces separated by sharp walls**. These sharp walls correspond to regions where the solution changes abruptly, and their evolution is influenced by the changing sign structure of the coefficient $a(t,x)$ as described. The numerical method is designed to resolve these features accurately, with the high spatial resolution capturing the steep gradients and the semi-implicit time integration providing stability and efficiency for both the linear and non-linear dynamics.

Solution Using Adaptive Neural Galerkin Approach

- The adaptive Neural Galerkin approach employs a neural network with three hidden layers known as feed forward neural network, using $m = 2$ tanh units, resulting in 16 degrees of freedom, and adopts a fixed uniform measure $d\nu_\theta(x) = dx$.

- The initial condition for the Neural Galerkin scheme is fitted analogously to the procedure used for the KdV equation experiment, ensuring consistent initialization.

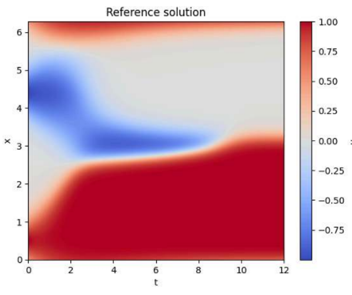
- Time integration is performed using the backward Euler discretization, an implicit method for advancing the solution in time.

- The gradient required for optimization is computed via automatic differentiation, implemented in JAX.

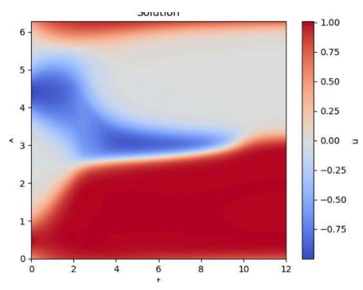
- The time-step size is set to $\delta t = 10^{-2}$.

- At each time step, the resulting nonlinear system is solved using the ADAM optimizer with 10,000 iterations to update the neural network parameters.

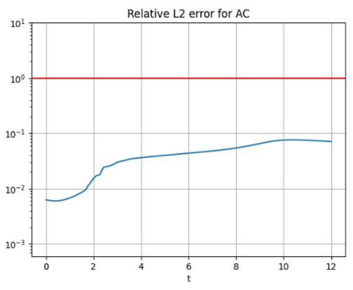
- The optimization problem at each time step is approximately solved using stochastic gradient descent (SGD) with $n = 1000$ samples to estimate the mass matrix M and the forcing vector F.
- The Neural Galerkin solution produced by this approach closely matches the benchmark solution, demonstrating the method's effectiveness.
- The adaptive Neural Galerkin method differs from traditional collocation-based neural network solvers by leveraging the Dirac-Frenkel variational principle. It minimizes the residual sequentially over time and adaptively collects new training data, guided by the evolving dynamics of the PDE solution. This adaptivity in both function approximation and sampling is especially beneficial when the solution exhibits local features that change over time
- The adaptive sampling strategy means that, at each time step, samples are drawn from regions where the solution exhibits significant features, as informed by the solution at the previous time step. This self-informed, dynamic sampling is critical for efficiently resolving solutions in high-dimensional or locally evolving problems.



(a) Reference solution



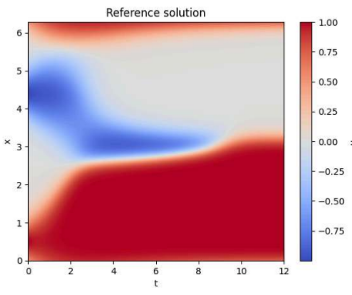
(b) Adaptive NG



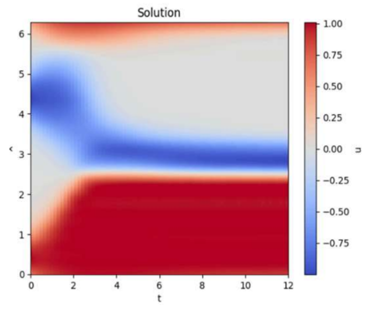
(c) Error over Time

Figure 8: Comparison of Reference Solution with Adaptive Neural Galerkin Approach for AC Equation

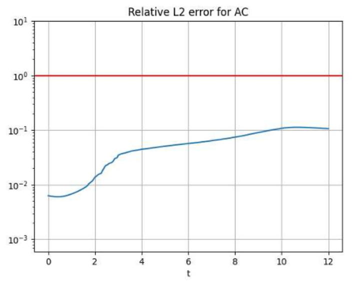
In contrast, the approximation obtained with the linear Galerkin method, which uses 16 equidistantly located Gaussian basis functions , results in a poorer approximation.



(a) Reference solution



(b) Linear Equidistant NG



(c) Error over Time

Figure 9: Comparison of Reference Solution with Linear Equidistant Neural Galerkin Approach for AC Equation

6.3 Advection Equation

We consider a high-dimensional *advection equation*, which models the passive transport of a scalar field $u(t, x)$ under a time- and space-dependent velocity field $a(t, x)$. The problem is posed on the unbounded domain $X = \mathbb{R}^d$, with governing equation:

$$\frac{\partial u}{\partial t} + a(t, x) \cdot \nabla_x u = 0, \quad u(0, x) = u_0(x).$$

Here, $x \in \mathbb{R}^d$ is the spatial variable, $t \geq 0$ denotes time, and the velocity field $a(t, x)$ is assumed to be smooth and bounded. The initial condition $u_0(x)$ satisfies $\lim_{|x|\to\infty} u_0(x) = 0$, ensuring proper decay at spatial infinity. This linear PDE describes the transport of the initial profile along trajectories defined by the velocity field.

The exact solution can, in principle, be obtained via the *method of characteristics*, which tracks how values of u are transported along paths defined by:

$$\frac{dX}{dt}(t, x) = a(t, X(t, x)), \quad X(0, x) = x.$$

The solution at any future time is given by evaluating the initial condition at the backward characteristic location:

$$u(t, x) = u_0(X(-t, x)).$$

This representation highlights that the solution simply follows the flow of the velocity field, preserving the structure of u_0 along characteristics. However, when the velocity field is complex or high-dimensional, directly computing or analyzing these trajectories becomes challenging.

Benchmark Explanation

As a benchmark, we consider situations where the velocity field $a(t, x)$ allows for an explicit or semi-explicit solution of the characteristic system. This enables direct evaluation of the solution using with high accuracy. Such benchmark solutions serve as reference points for testing numerical methods designed for high-dimensional advection problems.

In cases where the velocity field is complicated or cannot be solved analytically, numerical methods must approximate the characteristics or solve the PDE directly, often leading to challenges in capturing fine solution features or ensuring stability.

Hence, the benchmark problem provides a controlled setting with known solutions, facilitating quantitative comparison of numerical methods by measuring errors relative to this explicit solution.

Solution Using Neural Galerkin Method

Next we show how to numerically solve the advection equation directly using a Neural Galerkin scheme.

6.3.1 Advection with a time-dependent coefficient

Numerical solution using Neural Galerkin method

In this experiment, we solve a high-dimensional advection equation where the advection coefficient depends only on time. The time-dependent transport coefficient is defined as

$$a_t(t) = a_s \odot \left(\sin(a_v \pi t) + \frac{5}{4} \right),$$

where

$$a_s = [1, 2, \dots, d]^T, \quad a_v = 2 + 2\frac{[0, 1, \dots, d-1]^T}{d},$$

and $\odot$ denotes element-wise multiplication. The dimension is set as $d = 5$.

The initial condition u_0 is a mixture of two non-isotropic Gaussian packets with means and covariance matrices as follows:

The first Gaussian has mean

$$\mu_1 = \frac{11}{10}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^5,$$

and covariance matrix

$$\Sigma_1 = \frac{1}{200}\begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix}$$

for Dimension (d=5)

The second Gaussian has mean

$$\mu_2 = \frac{3}{4}\begin{bmatrix} 1.5 - \frac{(-1)^1 \cdot 1}{6} \\ 1.5 - \frac{(-1)^2 \cdot 2}{6} \\ 1.5 - \frac{(-1)^3 \cdot 3}{6} \\ 1.5 - \frac{(-1)^4 \cdot 4}{6} \\ 1.5 - \frac{(-1)^5 \cdot 5}{6} \end{bmatrix} = \frac{3}{4}\begin{bmatrix} 1.5 + \frac{1}{6} \\ 1.5 - \frac{2}{6} \\ 1.5 + \frac{3}{6} \\ 1.5 - \frac{4}{6} \\ 1.5 + \frac{5}{6} \end{bmatrix} = \frac{3}{4}\begin{bmatrix} \frac{10}{6} \\ \frac{7}{6} \\ \frac{12}{6} \\ \frac{5}{6} \\ \frac{14}{6} \end{bmatrix} = \begin{bmatrix} 1.25 \cdot \frac{10}{6} \\ 1.25 \cdot \frac{7}{6} \\ 1.25 \cdot \frac{12}{6} \\ 1.25 \cdot \frac{5}{6} \\ 1.25 \cdot \frac{14}{6} \end{bmatrix} \in \mathbb{R}^5,$$

The covariance matrix is given by

$$\Sigma_2 = \frac{1}{200}\begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Since the advection coefficient depends only on time, the exact solution to the partial differential equation is obtained by shifting the initial condition by the time-integrated velocity. The solution is given by

$$u(t, x) = u_0\left(x - \int_0^t a_s(s)\, ds\right),$$

where $u_0(x)$ is the initial condition and $a_s(t)$ is the time-dependent advection coefficient. This explicit expression serves as the benchmark for evaluating the accuracy of the numerical solution.

To analyze the solution, marginals for each dimension $i = 1, \dots, d$ are computed by integrating over all other dimensions:

$$(t, x_i) \mapsto \int_{\Omega_1} \cdots \int_{\Omega_{i-1}} \int_{\Omega_{i+1}} \cdots \int_{\Omega_d} u(t, x_1, \dots, x_d)\, dx_1 \cdots dx_{i-1} dx_{i+1} \cdots dx_d.$$

These integrals are approximated numerically using Monte Carlo integration with 8192 samples drawn from the analytical solution.

The Neural Galerkin method approximates the solution using a shallow neural network with Gaussian units:

$$U(\theta, x) = \sum_{i=1}^{50} c_i(t) \exp\left(-w_i^2(t)\|x - b_i(t)\|^2\right),$$

where each node has parameters: weight $c_i(t)$, width $w_i(t)$, and center $b_i(t)$. Since the Gaussian units are isotropic, many nodes are needed to represent the anisotropic initial condition and solution accurately.

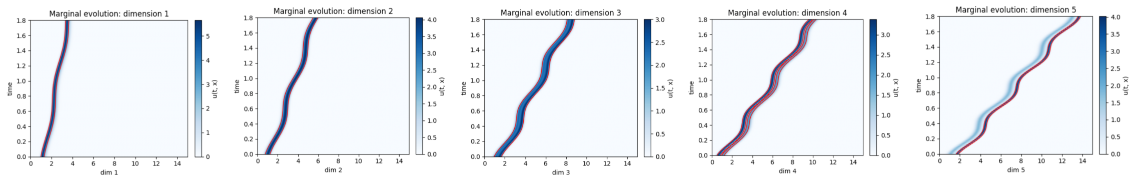
Time integration is performed with an adaptive RK45 method. At each integration step, the matrices M and forcing vectors F necessary for the Galerkin projection are estimated via samples $\{x_i\}_{i=1}^n$ drawn from an adaptive measure ν_θ defined as

$$d\nu_\theta(t)(x) = C^{-1} \sum_{i=1}^{50} \exp\left(-\frac{1}{\kappa^2} w_i^2(t)\|x - b_i(t)\|^2\right) dx,$$

where C is a normalization constant and $\kappa = 1$ in this experiment.

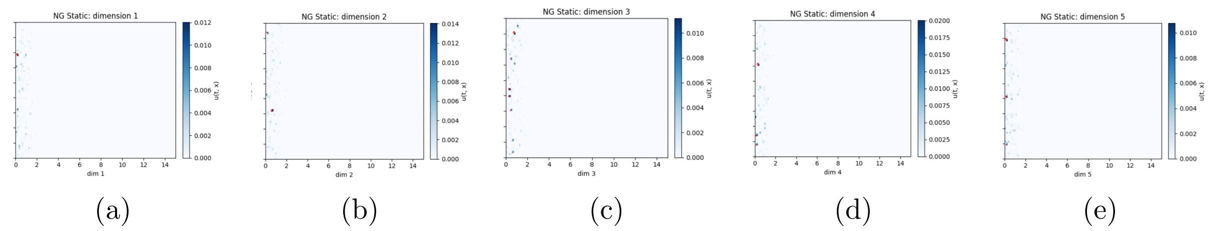
This adaptive sampling focuses on regions where the solution is significant, enabling accurate estimation of M and F with only $n = 1000$ points per timestep. In contrast, uniform sampling over the full spatial domain $[0, 15]^5$ requires orders of magnitude more points (e.g., $n = 10^5$) and still results in high errors due to insufficient coverage of localized features.

The results demonstrate the importance of the double adaptivity in Neural Galerkin methods: adapting both the function representation and the sampling measure to the evolving solution. This is a key advantage compared to standard deep learning approaches for high-dimensional PDEs.



(a) (b) (c) (d) (e)

Figure 10: Reference Solution for Advection Equation with Time-Only Varying Advection Speed



(a) (b) (c) (d) (e)

Figure 11: Advection Equation with Time-Only Varying Advection Speed using Static Neural Galerkin

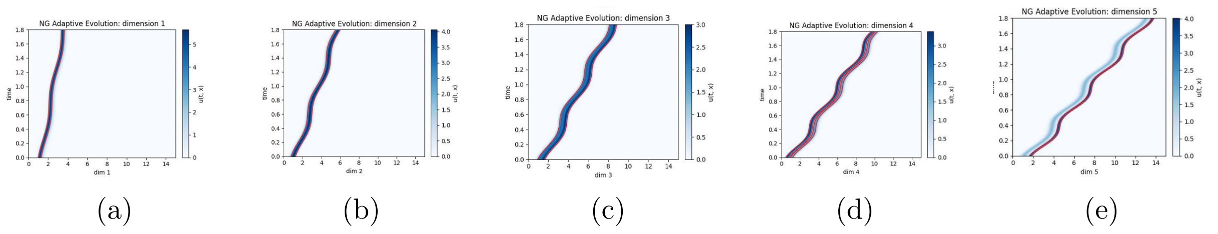


Figure 12: Adaptive NG Solution for Advection Equation with Time-Only Varying Advection Speed

6.3.2 Advection with Time- and Space-Dependent Coefficients

In this experiment, we solve a **high-dimensional advection equation** where the velocity field (also called the advection speed) varies over both **time** and **space**. This means that the rate and direction of movement of the quantity we study change depending on where and when we look.

The velocity at which the quantity moves, denoted as $a_{st}(t, x)$, is constructed by multiplying three factors together:

$$a_{st}(t, \mathbf{x}) = a_s \odot (\sin(a_v \pi t) + 3) \odot (\mathbf{x} + 1)/10,$$

- A vector a_s, which sets the base speed in each dimension of the space.
- A time-dependent oscillating term $\sin(\pi a_v t) + 3$, which causes the velocity to fluctuate periodically over time. The "+3" ensures that this term stays positive (no negative velocities).
- A space-dependent term $\frac{x+1}{10}$, which varies the velocity depending on the location x.

Putting these together, the advection speed changes smoothly over time and space, creating a complex flow that affects how the initial distribution moves.

The starting state of the system, $u_0(x)$, is given by adding together two Gaussian distributions (bell-shaped curves) scaled down by a factor of $\frac{1}{10}$:

$$u_0(x) = \frac{1}{10} p_1(x) + \frac{1}{10} p_2(x)$$

Each $p_i(x)$ represents a **probability density function** describing a localized "packet" in the high-dimensional space. They represent where the quantity is initially concentrated.

Scaling by $\frac{1}{10}$ prevents very large values that can cause numerical instability or scaling issues during computations.

Each Gaussian is characterized by:

- A **mean vector** μ that tells where the center of the packet is in the 5-dimensional space.
- A **covariance matrix** Σ that describes the spread or uncertainty around the mean in each dimension.

The means are:

$$\mu_1 = 2 - \frac{1}{12}\begin{bmatrix}-1\\2\\-3\\4\\-5\end{bmatrix}, \quad \mu_2 = 1.8 - \frac{1}{12}\begin{bmatrix}1\\-2\\3\\-4\\5\end{bmatrix}$$

The covariance matrices are diagonal (i.e., there is no correlation between dimensions), with small variances to ensure that the packets are tightly clustered.

$$\Sigma_1 = \text{diag}\left(\frac{3}{50}, \frac{3}{50}, \frac{3}{50}, \frac{3}{50}, \frac{3}{50}\right), \quad \Sigma_2 = \text{diag}\left(\frac{3}{100}, \frac{3}{100}, \frac{3}{100}, \frac{3}{100}, \frac{3}{100}\right)$$

This means packet 1 is slightly more spread out than packet 2.

The problem cannot be solved exactly (no closed-form formula), so the **Neural Galerkin method** is used — a technique that approximates the solution using neural networks and projects the PDE dynamics onto a set of basis functions.

The approach to test the method's accuracy is:

- Integrate the equation forward in time from $t = 0$ to $t = 1$, tracking how the initial packets move.
- Then reverse the flow by **inverting the time-dependent velocity** (think of playing the flow backward).
- Integrate backward from $t = 1$ to $t = 0$.

If the method is accurate, after this forward-backward integration the solution should return very close to the initial condition.

To solve the problem, integrals involving the solution's representation must be estimated. This is done by **sampling points** in the high-dimensional space.

- At each time step, **5,000 samples** are drawn from an **adaptive probability distribution** tailored to the current state of the solution. This helps focus computational effort where the solution has significant values.
- To estimate the **error or residual** of the PDE solution (i.e., how well the PDE is satisfied), a much larger sample of **100,000 points** is drawn from a mixture of Gaussian distributions centered at the neural network's nodes. This allows a robust Monte Carlo estimate of the error.
- The time-space marginals of the solution show how the spatial dependence in velocity causes the packets to deform and move non-uniformly.
- Comparing the initial condition and the final state after forward and backward integration shows **good agreement**, validating the numerical method's stability and accuracy.
- The residual error remains small, confirming that the Neural Galerkin approach effectively captures the local dynamics in this complex transport problem.

- In contrast, using a uniform (non-adaptive) sampling strategy fails to capture the details properly, showing the importance of adaptive sampling.

This experiment demonstrates that the Neural Galerkin method, combined with adaptive sampling, can effectively solve a challenging high-dimensional advection problem with complex, time- and space-dependent velocities. By verifying the reversibility of the solution and monitoring residuals, the accuracy and stability of the method are confirmed even in scenarios where no exact solution is available.

7 Extended Formulation for Second Time Derivative – Degree Two Equation

Problem Setup and Derivation(Single Degree)

We consider the use of a neural network $U(\theta(t), x)$ to approximate the solution $u(t, x)$ of a time-dependent partial differential equation (PDE) of the form:

$$\partial_t u = f(t, x, u)$$

Here:

- $\theta(t) \in \mathbb{R}^d$ denotes the time-varying parameters of the neural network,
- $x \in X \subset \mathbb{R}^n$ is the spatial variable,
- $U(\theta(t), x)$ is the neural approximation to the true solution $u(t, x)$.

The key idea is that we do not use any training data. Instead, we enforce that the neural network output satisfies the PDE by minimizing the residual — the error in the governing equation. The time-dependence of U is implicit through $\theta(t)$. Therefore, when computing the time derivative of U, we apply the chain rule:

$$\partial_t U(\theta(t), x) = \nabla_\theta U(\theta, x) \cdot \dot{\theta}(t)$$

This expression for the first derivative is essential in linking the PDE with the dynamics of the network parameters.

Step-by-Step Derivation

Step 1: Define the residual
We define the residual function as the difference between the left and right sides of the PDE, evaluated using the network:

$$r_t(\theta, \dot{\theta}, x) = \nabla_\theta U(\theta, x) \cdot \dot{\theta}(t) - f(t, x, U(\theta, x))$$

Step 2: Define the loss functional

To enforce the residual is small over the spatial domain X, we define the squared residual loss:

$$J_t(\theta, \dot{\theta}) = \frac{1}{2} \int_X \left(r_t(\theta, \dot{\theta}, x) \right)^2 \, d\nu(x)$$

where $\nu(x)$ is the spatial measure (e.g., uniform or empirical over sample points).

Step 3: Reformulate as a least squares problem

Let:

$$\eta = \dot{\theta}(t), \quad A(x) = \nabla_\theta U(\theta, x), \quad b(x) = f(t, x, U(\theta, x))$$

Then:

$$J(\eta) = \frac{1}{2} \int_X (A(x) \cdot \eta - b(x))^2 \, d\nu(x)$$

Step 4: Compute the gradient of the loss

Taking the gradient with respect to η:

$$\nabla_\eta J = \int_X A(x)(A(x) \cdot \eta - b(x)) \, d\nu(x)$$

Step 5: Set the gradient to zero (first-order optimality condition)

Setting $\nabla_\eta J = 0$, we obtain:

$$\int_X A(x)A(x)^T \, d\nu(x) \cdot \eta = \int_X A(x)b(x) \, d\nu(x)$$

Substituting back $\eta = \dot{\theta}(t)$:

$$\int_X \nabla_\theta U(\theta, x) \nabla_\theta U(\theta, x)^T \, d\nu(x) \cdot \dot{\theta}(t) = \int_X \nabla_\theta U(\theta, x) f(t, x, U(\theta, x)) \, d\nu(x)$$

Step 6: Define final expressions and ODE

Define:

$$M(\theta) = \int_X \nabla_\theta U(\theta, x) \otimes \nabla_\theta U(\theta, x) \, d\nu(x)$$

$$F(t, \theta) = \int_X \nabla_\theta U(\theta, x) f(t, x, U(\theta, x)) \, d\nu(x)$$

Then the final system becomes:

$$M(\theta) \cdot \dot{\theta}(t) = F(t, \theta)$$

This is the ordinary differential equation (ODE) that governs the evolution of the neural network parameters $\theta(t)$, ensuring that the network output satisfies the original PDE by minimizing the squared residual over the spatial domain.

Problem Setup and Derivation (Second Time Derivative – Degree Two Equation)

We consider a time-dependent partial differential equation of the form:

$$\partial_t^2 u(t, x) = f(t, x, u(t, x), \nabla_x u(t, x))$$

defined on a spatial domain $x \in X \subset \mathbb{R}^d$ and time $t \in [0, T]$. We approximate the solution using a parameterized surrogate:

$$u(t, x) \approx U(\theta(t), x),$$

where $\theta(t) \in \mathbb{R}^n$ is a vector of time-dependent parameters, and $U : \mathbb{R}^n \times X \to \mathbb{R}$ is a smooth function.

Differentiating this surrogate with respect to time:

$$\frac{d}{dt} U(\theta(t), x) = \nabla_\theta U(\theta(t), x) \cdot \dot{\theta}(t),$$

$$\frac{d^2}{dt^2} U(\theta(t), x) = \nabla_\theta U(\theta(t), x) \cdot \ddot{\theta}(t) + \dot{\theta}(t)^\top \nabla^2_{\theta\theta} U(\theta(t), x) \dot{\theta}(t).$$

We define the residual as the mismatch between the surrogate and the PDE dynamics:

$$r_t(x) = \partial_t^2 U(\theta(t), x) - f(t, x, U(\theta(t), x), \nabla_\theta U(\theta(t), x) \cdot \dot{\theta}(t)).$$

Let us denote:

$$g(x) := \nabla_\theta U(\theta, x), \quad H(x) := \nabla^2_{\theta\theta} U(\theta, x), \quad w(x) := g(x) \cdot \dot{\theta}(t), \quad U := U(\theta, x),$$

so that the residual becomes:

$$r_t(x) = g(x) \cdot \ddot{\theta}(t) + \dot{\theta}(t)^\top H(x) \dot{\theta}(t) - f(t, x, U, w(x)).$$

The objective functional is defined as:

$$J_t(\theta, \dot{\theta}, \ddot{\theta}) = \frac{1}{2} \int_X |r_t(x)|^2 \, d\nu(x),$$

where ν is a measure on X.

Gradient with Respect to $\ddot{\theta}(t)$

The gradient of J_t with respect to $\ddot{\theta}(t)$ is:

$$\nabla_{\ddot{\theta}} J_t = \int_X r_t(x) \cdot \nabla_{\ddot{\theta}} r_t(x) \, d\nu(x).$$

Since $r_t(x)$ depends linearly on $\ddot{\theta}$ via $g(x) \cdot \ddot{\theta}$, we have:

$$\nabla_{\ddot{\theta}} r_t(x) = g(x),$$

$$\nabla_{\ddot{\theta}} J_t = \int_X r_t(x) \cdot g(x) \, d\nu(x).$$

Substituting the full expression for $r_t(x)$:

$$r_t(x) = g(x) \cdot \ddot{\theta} + \dot{\theta}^\top H(x) \dot{\theta} - f(t, x, U, w),$$

$$\nabla_{\ddot{\theta}} J_t = \int_X \Big(g(x) \cdot \ddot{\theta} + \dot{\theta}^\top H(x) \dot{\theta} - f(t, x, U, w) \Big) g(x) \, d\nu(x).$$

Breaking into terms:

$$M(\theta) := \int_X g(x) \otimes g(x) \, d\nu(x), \quad C(\theta, \dot{\theta}) := \int_X (\dot{\theta}^\top H(x) \dot{\theta}) g(x) \, d\nu(x),$$

$$F(t, \theta, \dot{\theta}) := \int_X f(t, x, U, w(x)) \cdot g(x) \, d\nu(x).$$

Then setting the gradient to zero:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) = F(t, \theta, \dot{\theta}). \tag{a}$$

Gradient with Respect to $\dot{\theta}(t)$

We are minimizing the cost functional

$$J_t = \frac{1}{2} \int_X r_t(x)^2 \, d\nu(x)$$

with respect to $\dot{\theta}(t)$, where the residual is defined as

$$r_t(x) = \underbrace{\nabla_\theta U(\theta, x) \cdot \ddot{\theta}}_{(A)} + \underbrace{\dot{\theta}^\top \nabla^2_{\theta\theta} U(\theta, x) \cdot \dot{\theta}}_{(B)} - \underbrace{f(t, x, U, \nabla_\theta U \cdot \dot{\theta})}_{(C)}.$$

The gradient of the residual $r_t(x)$ with respect to $\dot{\theta}$ is computed as follows. Term (A) has no dependence on $\dot{\theta}$, so it contributes zero. The second term yields

$$\nabla_{\dot{\theta}} \left[\dot{\theta}^\top \nabla^2_{\theta\theta} U(\theta, x) \cdot \dot{\theta} \right] = 2 \nabla^2_{\theta\theta} U(\theta, x) \cdot \dot{\theta}.$$

For the third term, we define $w(x) = \nabla_\theta U(\theta, x) \cdot \dot{\theta}$, and by the chain rule we obtain

$$\nabla_{\dot{\theta}} f(t, x, U, w(x)) = \frac{\partial f}{\partial w} \cdot \nabla_\theta U(\theta, x).$$

Hence, the total gradient is

$$\nabla_{\dot{\theta}} r_t(x) = 2 \nabla^2_{\theta\theta} U(\theta, x) \cdot \dot{\theta} - \frac{\partial f}{\partial w} \cdot \nabla_\theta U(\theta, x).$$

Applying the chain rule to the objective functional, we get

$$\nabla_{\dot{\theta}} J_t = \int_X r_t(x) \cdot \nabla_{\dot{\theta}} r_t(x) \, d\nu(x),$$

which becomes

$$\nabla_{\dot{\theta}} J_t = \int_X r_t(x) \cdot \left[2 \nabla^2_{\theta\theta} U(\theta, x) \cdot \dot{\theta} - \frac{\partial f}{\partial w} \cdot \nabla_\theta U(\theta, x) \right] d\nu(x).$$

This is a vector equation that must be solved for the optimal value of $\dot{\theta}(t)$ at each time t.

Setting this to zero gives:

$$\int_X r_t(x) \cdot \left(2H(x)\dot{\theta} - \frac{\partial f}{\partial w} \cdot g(x) \right) d\nu(x) = 0. \tag{b}$$

Coupled Optimality System

The necessary conditions for optimality are given by the coupled system:

$$M(\theta)\ddot{\theta} = F(t, \theta, \dot{\theta}) - C(\theta, \dot{\theta}), \tag{a}$$

$$\int_X r_t(x) \cdot \left(2H(x)\dot{\theta} - \frac{\partial f}{\partial w} \cdot g(x) \right) d\nu(x) = 0. \tag{b}$$

These equations define a coupled nonlinear system in $(\dot{\theta}, \ddot{\theta})$ that must be solved at each time t, given $\theta(t)$.

Solution Methodology

Step 1: Fix $\theta(t)$
We assume that $\theta(t)$ is known at the current time t. This allows us to evaluate the spatial functions $g(x)$, $H(x)$, and $U(\theta, x)$ without ambiguity. These will be treated as known inputs.

Initial Guess for $\dot{\theta}(t)$

The selection of an appropriate initial value for $\dot{\theta}(t)$ plays a critical role in ensuring the convergence and stability of iterative schemes employed to minimize the residual functional $J_t(\theta, \dot{\theta}, \ddot{\theta})$. The strategy for choosing $\dot{\theta}^{(0)}$ is problem-dependent and is guided by the availability of prior information or structural properties of the underlying dynamical system. Commonly employed initialization strategies include:

1. Zero Initialization When no prior information is available, it is standard to assume the system begins at rest:

$$\dot{\theta}^{(0)} = 0. \tag{1}$$

This is particularly effective when $t = 0$ and the residual functional is convex or well-behaved.

2. Finite-Difference Approximation If historical data is available (e.g., from earlier time steps), a first-order approximation is:

$$\dot{\theta}(t_k) \approx \frac{\theta(t_k) - \theta(t_{k-1})}{\Delta t}. \tag{2}$$

This is effective in discretized time domains.

3. Exponential Moving Average In systems with stiff or oscillatory dynamics, a smoothed estimate is:

$$\dot{\theta}^{(0)} = \beta \dot{\theta}_{\text{prev}} + (1 - \beta) \cdot \frac{\theta(t_k) - \theta(t_{k-1})}{\Delta t}, \tag{3}$$

where $\beta \in [0, 1)$ is a smoothing parameter.

4. Gradient-Based Descent In learning contexts, initialize along the negative gradient direction:

$$\dot{\theta}^{(0)} = -\eta \nabla_\theta L(\theta), \tag{4}$$

where $\eta > 0$ is a learning rate and $L(\theta)$ is a scalar loss functional.

5. Stochastic Prior Sampling In Bayesian or multi-start optimization methods:

$$\dot{\theta}^{(0)} \sim \mathcal{N}(0, \sigma^2 I), \tag{5}$$

where $\sigma > 0$ is the standard deviation.

Step 2: Initialize
Set the initial guess $\dot{\theta}^{(0)}(t)$ using one of the above strategies.

Step 3: Solve the Acceleration Equation
Given $\dot{\theta}^{(k)}$, compute the acceleration from:

$$\ddot{\theta}^{(k)} = M^{-1}(\theta)\left[F(t,\theta,\dot{\theta}^{(k)}) - C(\theta,\dot{\theta}^{(k)})\right]. \tag{3}$$

This will be used to evaluate the residual $r_t(x)$.

Step 4: Solve the Nonlinear Integral Equation
With $\ddot{\theta}^{(k)}$, compute the residual:

$$r_t^{(k)}(x) = g(x)\cdot\ddot{\theta}^{(k)} + (\dot{\theta}^{(k)})^T H(x)\dot{\theta}^{(k)} - f(t,x,U,g(x)\cdot\dot{\theta}^{(k)}). \tag{6}$$

Substitute into the governing equation:

$$\int_X r_t^{(k)}(x)\cdot\left(2H(x)\dot{\theta}^{(k+1)} - \frac{\partial f}{\partial w}\cdot g(x)\right) d\nu(x) = 0. \tag{4}$$

Define the nonlinear operator:

$$\mathcal{G}(\dot{\theta}) := \int_X r_t(x)\cdot\left(2H(x)\dot{\theta} - \frac{\partial f}{\partial w}\cdot g(x)\right) d\nu(x). \tag{7}$$

Then solve $\mathcal{G}(\dot{\theta}) = 0$ using:
(a) Newton–Raphson Method:

$$\dot{\theta}^{(k+1)} = \dot{\theta}^{(k)} - \left(\nabla_{\dot{\theta}}\mathcal{G}\right)^{-1}\mathcal{G}(\dot{\theta}^{(k)}). \tag{8}$$

(b) Gradient Descent Method:

$$\dot{\theta}^{(k+1)} = \dot{\theta}^{(k)} - \alpha\nabla_{\dot{\theta}}\mathcal{G}, \tag{9}$$

where $\alpha > 0$ is a learning rate.

Step 5: Iterate to Convergence
Repeat Steps 3 and 4 until the convergence criteria are satisfied:

$$\|\dot{\theta}^{(k+1)} - \dot{\theta}^{(k)}\| < \varepsilon, \quad \|\ddot{\theta}^{(k+1)} - \ddot{\theta}^{(k)}\| < \varepsilon, \tag{10}$$

where $\varepsilon > 0$ is a prescribed tolerance.

Example Usage

Consider the general second-order nonlinear partial differential equation:

$$\partial_t^2 u(x,t) = f\left(x,t,u(x,t),\nabla_x u(x,t),\nabla_x^2 u(x,t)\right) \tag{11}$$

where:

- $u(x,t)$: Unknown scalar field defined over space $x \in \mathbb{R}^n$ and time $t \in \mathbb{R}$.
- $\partial_t^2 u(x,t)$: Second partial derivative of u with respect to time t; it represents the temporal acceleration of the system.

- $f(\cdot)$: A nonlinear function of:
  - x, t: The spatial and temporal coordinates.
  - $u(x,t)$: The function value at point (x,t).
  - $\nabla_x u(x,t)$: The **gradient** of u with respect to spatial variables x, i.e., the vector of first-order spatial derivatives.
  - $\nabla_x^2 u(x,t)$: The **Hessian matrix** of second-order spatial derivatives. In many physical models, this is replaced by the **Laplacian** of u, denoted as:

    $$\Delta u(x,t) = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial x_i^2}$$

    The Laplacian Δu captures the net curvature or diffusion of the function in space and is a key operator in many physical systems such as diffusion, wave propagation, and quantum mechanics.

This general form includes many important nonlinear PDEs, such as nonlinear wave equations, reaction-diffusion systems, and damped oscillatory systems with spatial interaction represented by the Laplacian.

Surrogate Model

We approximate the solution using a time-dependent parametric surrogate:

$$u(x,t) \approx U(\theta(t), x), \quad \theta(t) \in \mathbb{R}^n \tag{12}$$

where U is a smooth function in both arguments.

Define:

$$\begin{aligned} g(x) &:= \nabla_\theta U(\theta(t), x) \in \mathbb{R}^n \\ H(x) &:= \nabla_{\theta\theta}^2 U(\theta(t), x) \in \mathbb{R}^{n\times n} \\ w(x) &:= g(x) \cdot \dot{\theta}(t) \in \mathbb{R} \\ U &:= U(\theta(t), x) \\ \nabla_x U &:= \nabla_x U(\theta(t), x) \in \mathbb{R}^d \\ \nabla_x^2 U &:= \nabla_x^2 U(\theta(t), x) \in \mathbb{R}^{d\times d} \end{aligned}$$

The time derivatives of the surrogate are:

$$\frac{d}{dt} U(\theta(t), x) = g(x) \cdot \dot{\theta}(t) \tag{13}$$

$$\frac{d^2}{dt^2} U(\theta(t), x) = g(x) \cdot \ddot{\theta}(t) + \dot{\theta}(t)^\top H(x) \dot{\theta}(t) \tag{14}$$

Residual and Objective Functional

We define the residual:

$$r_t(x) := g(x) \cdot \ddot{\theta}(t) + \dot{\theta}(t)^\top H(x) \dot{\theta}(t) - f(x, t, U, \nabla_x U, \nabla_x^2 U) \tag{15}$$

The residual minimization objective is:

$$J_t(\theta, \dot{\theta}, \ddot{\theta}) := \frac{1}{2} \int_X |r_t(x)|^2 \, d\nu(x) \tag{16}$$

Gradient with Respect to $\ddot{\theta}(t)$

Since $r_t(x)$ is linear in $\ddot{\theta}(t)$:

$$\nabla_{\ddot{\theta}} r_t(x) = g(x) \tag{17}$$

Thus:

$$\nabla_{\ddot{\theta}} J_t = \int_X r_t(x) \cdot g(x)\, d\nu(x) \tag{18}$$

We define the following quantities:

$$M(\theta) := \int_X g(x) \otimes g(x)\, d\nu(x) \tag{19}$$

$$C(\theta, \dot{\theta}) := \int_X (\dot{\theta}^\top H(x)\dot{\theta}) g(x)\, d\nu(x) \tag{20}$$

$$F(t, \theta, \dot{\theta}) := \int_X f(x, t, U, \nabla_x U, \nabla_x^2 U) g(x)\, d\nu(x) \tag{21}$$

Setting the gradient to zero:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) = F(t, \theta, \dot{\theta}) \tag{22}$$

Gradient with Respect to $\dot{\theta}(t)$

The derivative of the residual with respect to $\dot{\theta}(t)$ is:

$$\nabla_{\dot{\theta}} r_t(x) = 2H(x)\dot{\theta} - \nabla_{\dot{\theta}} f(x, t, U, \nabla_x U, \nabla_x^2 U) \tag{23}$$

Using the chain rule for the second term:

$$\nabla_{\dot{\theta}} f = \frac{\partial f}{\partial U} \cdot g(x) + \sum_{j=1}^{d} \frac{\partial f}{\partial (\nabla_x U)_j} \cdot \nabla_\theta \left(\frac{\partial U}{\partial x_j} \right) + \sum_{j,k=1}^{d} \frac{\partial f}{\partial (\nabla_x^2 U)_{jk}} \cdot \nabla_\theta \left(\frac{\partial^2 U}{\partial x_j \partial x_k} \right)$$

Simplifying this, we observe that the surrogate U depends on $\theta(t)$ through its parametric form, so its derivatives with respect to x also depend on θ. Thus, applying the chain rule to each term:

The first term involves the gradient

$$g(x) := \nabla_{\dot{\theta}} U = \frac{\partial U}{\partial \theta} \cdot \nabla_{\dot{\theta}} \theta(t)$$

If we treat θ and $\dot{\theta}$ as independent during differentiation (i.e., not integrating in time), then $\nabla_{\dot{\theta}} \theta(t) = 0$, and hence $g(x) = 0$. Therefore, the first term vanishes unless f depends explicitly on $\dot{\theta}$ or on $\frac{dU}{dt}$.

For the second term, we note that each spatial derivative of U also depends on θ, so

$$\nabla_\theta \left(\frac{\partial U}{\partial x_j} \right) = \frac{\partial^2 U}{\partial \theta\, \partial x_j}$$

which is a matrix in $\mathbb{R}^n$. Thus, the second term simplifies to

$$\sum_{j=1}^{d} \frac{\partial f}{\partial (\nabla_x U)_j} \cdot \frac{\partial^2 U}{\partial \theta\, \partial x_j}$$

Similarly, the third term becomes

$$\sum_{j,k=1}^{d} \frac{\partial f}{\partial (\nabla_x^2 U)_{jk}} \cdot \frac{\partial^3 U}{\partial \theta \, \partial x_j \, \partial x_k}$$

Combining the results, we find that the total derivative $\nabla_{\dot{\theta}} f$ reduces to contributions involving second- and third-order derivatives of the surrogate U with respect to θ and the spatial variables x_j, x_k, weighted by the sensitivities of f with respect to $\nabla_x U$ and $\nabla_x^2 U$. Thus, the final expression for the gradient of f with respect to $\dot{\theta}$ is given by:

$$\nabla_{\dot{\theta}} f = \underbrace{0}_{\text{if } U \text{ has no explicit dependence on } \dot{\theta}} + \sum_{j=1}^{d} \frac{\partial f}{\partial (\nabla_x U)_j} \cdot \frac{\partial^2 U}{\partial \theta \, \partial x_j} + \sum_{j=1}^{d} \sum_{k=1}^{d} \frac{\partial f}{\partial (\nabla_x^2 U)_{jk}} \cdot \frac{\partial^3 U}{\partial \theta \, \partial x_j \, \partial x_k} \tag{24}$$

Substituting into the gradient:

$$\nabla_{\dot{\theta}} J_t = \int_X r_t(x) \cdot \left(2H(x)\dot{\theta} - \nabla_{\dot{\theta}} f \right) d\nu(x) \tag{25}$$

Setting to zero yields:

$$\int_X r_t(x) \cdot \left(2H(x)\dot{\theta} - \nabla_{\dot{\theta}} f \right) d\nu(x) = 0 \tag{26}$$

Coupled Optimality System

Equations (22) and (26) form the coupled nonlinear optimality system:

$$M(\theta)\ddot{\theta} = F(t, \theta, \dot{\theta}) - C(\theta, \dot{\theta}) \tag{c}$$

$$\int_X r_t(x) \cdot \left(2H(x)\dot{\theta} - \nabla_{\dot{\theta}} f \right) d\nu(x) = 0 \tag{d}$$

These equations can be solved simultaneously at each time t, given $\theta(t)$, to determine $\dot{\theta}(t)$ and $\ddot{\theta}(t)$.

8 Conclusion

In conclusion, by combining deep learning with adaptive, physics-informed data generation, the Neural Galerkin scheme offers a substantial improvement in the solution of high-dimensional partial differential equations. By strategically allocating computational resources to areas of complexity or rapid change, its dynamic training process overcomes the drawbacks of conventional static sampling techniques. In addition to improving accuracy, this novel method opens the door for effective, scalable solutions in high-dimensional problem spaces.

These characteristics make the Neural Galerkin scheme especially helpful for high-dimensional problems that were previously unsolvable through computation. It has the potential to be applied universally to PDEs with different sizes and levels of complexity. However, this method is still in its infancy and needs more theoretical knowledge, better computational methods, and algorithmic optimization. It is a dynamic and promising field for further investigation in scientific computing and machine learning, but high-resolution research efforts are necessary to improve its scalability, robustness, and practical applicability.

All implementation codes and reports can be found at `https://github.com/AYUSHIIESTS/NG_Adaptive`.

9 References

1. Bruna, J., Peherstorfer, B., & Vanden-Eijnden, E. (2024). Neural Galerkin schemes with active learning for high-dimensional evolution equations. *Journal of Computational Physics, 496*, 112588.

2. Bruna, J., Peherstorfer, B., & Vanden-Eijnden, E. (2022). Neural Galerkin schemes with active learning for high-dimensional evolution equations.

3. Gu, Y., & Ng, M. K. (2021). Deep adaptive basis Galerkin method for high-dimensional evolution equations with oscillatory solutions.

4. Peherstorfer, B. (2022, May 4). Neural Galerkin schemes with active learning for high-dimensional evolution equations. *Seminar, NYU.*

5. FutureMojo. (n.d.). NLP Demystified: Essential Training Techniques for Neural Networks. *YouTube playlist and Colab notebook.*

6. Bruna, J., Peherstorfer, B., & Vanden-Eijnden, E. (2024). Neural Galerkin schemes with active learning for high-dimensional evolution equations. *ScienceDirect.*

7. Gu, Y. (n.d.). Deep Adaptive Galerkin Method.

8. Peherstorfer, B. (n.d.). GitHub Repository for Neural Galerkin implementation.

9. SIAM Journal Article Related to Neural Galerkin Methods.