A Computational Study of Spectral Behavior in Physics-Informed Neural Networks

PhD Student: Andrei-Ionuț Mohuț
Coordinator: Prof. Dr. Călin-Adrian Popa

Politehnica University of Timișoara
Department of Computers and Information Technology

June 3, 2025

Motivation

- Differential equations are fundamental tools in modeling physical systems — from quantum mechanics to fluid dynamics.
- Analytical solutions are often intractable; numerical methods provide approximations but introduce discretization and stability challenges.
- Advances in deep learning offer new, flexible tools to solve differential equations through data-driven or physics-informed models.
- Spectral theory connects linear operators, eigenvalue problems, and function spaces — crucial in understanding wave behavior, stability, and modal dynamics.
- Recent research suggests that spectral insights can help design better neural architectures and explain training behavior (e.g., spectral bias).
- This project aims to explore the intersection of deep learning, differential equations, and spectral theory through computational experiments.

Universitatea Politehnica Timișoara

Summary

Universitatea Politehnica Timișoara

A Single Dense Neuron for Regression Tasks

Artificial Intelligence (AI): Simulates intelligent behavior (e.g., reasoning, planning).

Machine Learning (ML): Learns patterns from data for prediction and decision-making.

Deep Learning (DL): Uses multi-layer neural networks to model complex functions.

$DL \subset ML \subset AI$

Examples:

- AI: Robotics, game agents
- ML: Spam filters, recommendations
- DL: Image classification, text-to-speech

House Price Prediction (Single Neuron) Input features:

- x_1 Size (m^2)
- x_2 Number of rooms
- x_3 Location score
- x_4 Year built
- x_5 Distance to center (km)

Single house prediction:

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + b$$

Matrix form for n houses:

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

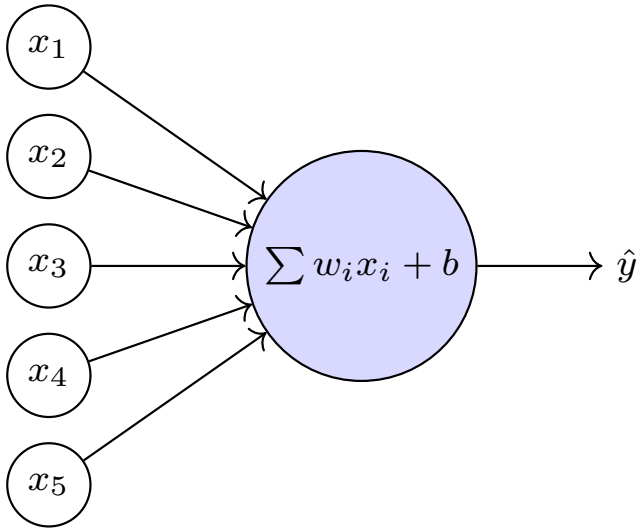
$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{15} \\ x_{21} & x_{22} & \cdots & x_{25} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{n5} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_5 \end{bmatrix}$$

Output $\hat{\mathbf{y}} \in \mathbb{R}^n$ gives predicted prices.

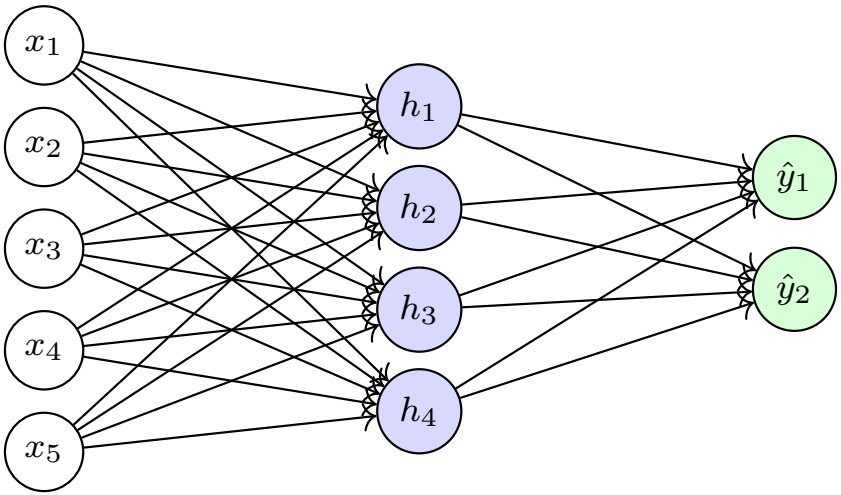
Universitatea Politehnica Timișoara

From a Single Neuron to a Small Neural Network

Single Neuron (Fully Connected)



Neural Network (1 Hidden Layer, 2 Outputs)



Single Neuron Equation:

$$\hat{y} = \sum_{i=1}^{5} w_i x_i + b \quad \text{or} \quad \hat{y} = \mathbf{x}^\top \mathbf{w} + b$$

Neural Network (Layered Form):

$$\mathbf{h} = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

$$\hat{\mathbf{y}} = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2$$

Shapes:

- $\mathbf{x} \in \mathbb{R}^5$
- $\mathbf{W}_1 \in \mathbb{R}^{4\times 5}$, $\mathbf{b}_1 \in \mathbb{R}^4$
- $\mathbf{W}_2 \in \mathbb{R}^{2\times 4}$, $\mathbf{b}_2 \in \mathbb{R}^2$
- $\hat{\mathbf{y}} \in \mathbb{R}^2$

Universitatea Politehnica Timișoara

Why Layers Alone Are Not Enough

Without activation functions:

$$\mathbf{h} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1, \quad \hat{\mathbf{y}} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

Substituting:

$$\hat{\mathbf{y}} = \mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = \underbrace{\mathbf{W}_2\mathbf{W}_1}_{\mathbf{W}_{\text{eq}}}\mathbf{x} + \underbrace{\mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2}_{\mathbf{b}_{\text{eq}}}$$

$$\Rightarrow \quad \hat{\mathbf{y}} = \mathbf{W}_{\text{eq}}\mathbf{x} + \mathbf{b}_{\text{eq}}$$

Conclusion: multiple layers without nonlinearities collapse to a single linear map.

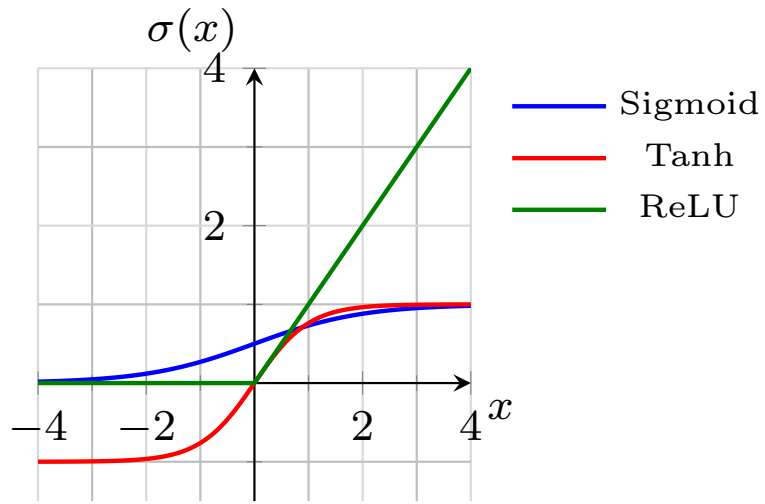
To model non-linear relationships:

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \quad \hat{\mathbf{y}} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

Now the network can learn non-linear functions.

Common Nonlinearities:

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tanh: $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- ReLU: $\text{ReLU}(x) = \max(0, x)$



Nonlinearities add expressive power beyond linear systems.

Neural Networks as Universal Approximators

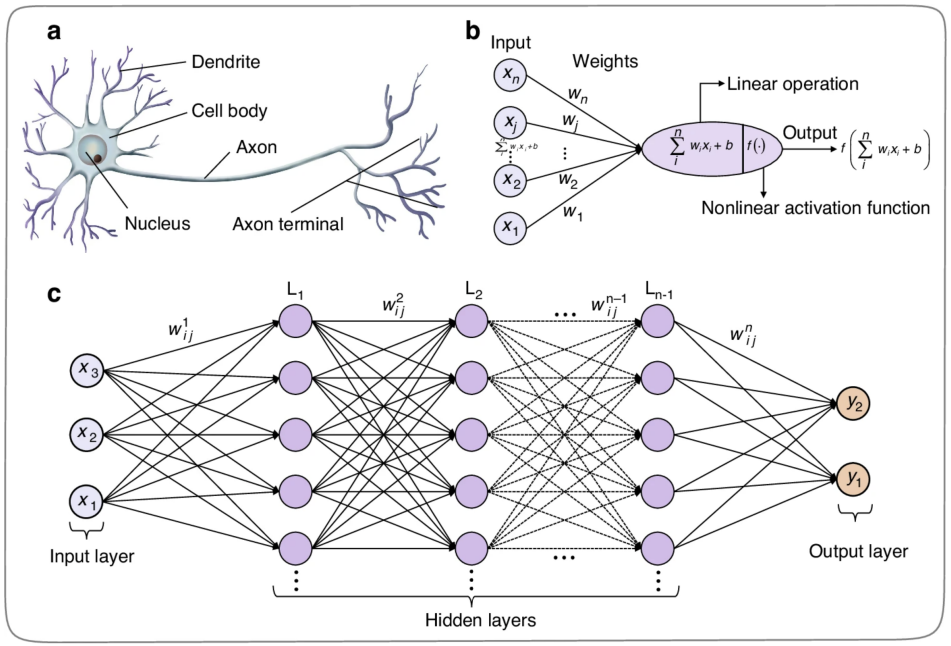


Figure 1: Biological and mathematical abstraction of a neural network

Source: Fu, T., Zhang, J., Sun, R. et al. Optical neural networks: progress and challenges. *Light Sci Appl* **13**, 263 (2024). https://doi.org/10.1038/s41377-024-01590-3

Universal Approximation Theorem

Consider a fixed activation function $\sigma : \mathbb{R} \to \mathbb{R}$ that is not a polynomial. For any integers d, D, any continuous function $f : \mathbb{R}^d \to \mathbb{R}^D$, any compact set $K \subset \mathbb{R}^d$, and any $\epsilon > 0$, there exists a neural network function $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^D$ such that:

$$f_\epsilon = W_2 \circ \sigma \circ W_1$$

where W_1, W_2 are affine maps and:

$$\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \epsilon$$

Implication: Neural networks can approximate any continuous function on compact domains arbitrarily well.

Universitatea Politehnica Timișoara

Learning as Optimization in Neural Networks

Training data:

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}, \quad x^{(i)} \in \mathbb{R}^d, \; y^{(i)} \in \mathbb{R}^D$$

Prediction:

$$\hat{y}^{(i)} = f_\theta(x^{(i)}), \quad \theta = \{W_k, b_k\}$$

Loss (MSE):

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \|\hat{y}^{(i)} - y^{(i)}\|^2$$

Training (Gradient Descent):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t)$$

Why not solve analytically?

- Non-convex loss landscape
- Millions of parameters
- No closed-form for $\nabla_\theta \mathcal{L} = 0$

Gradient Computation:
Backpropagation applies the chain rule layer-by-layer:

$$\frac{\partial \mathcal{L}}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial a_k} \cdot \frac{\partial a_k}{\partial W_k}$$

where a_k is the activation at layer k

Implementation:

- Gradients are computed via **automatic differentiation**
- **No manual derivatives** needed

Hardware Acceleration:

- Matrix operations run on GPUs/TPUs
- Enables fast training over large datasets

Goal: Learn parameters θ that minimize loss and generalize to unseen data

Universitatea Politehnica Timișoara

PINNs: Solving PDEs through Optimization

Physics-Informed Neural Networks (PINNs) solve PDEs by embedding physical laws into the loss function. The network approximates $u(x,t)$ by minimizing residuals of the governing equations, boundary, and initial conditions.

Governing PDE: Describes the physical system (e.g., heat, wave, Burgers' equation).

$$\mathcal{D}[u(x,t);\lambda] = f(x,t), \quad x \in \Omega, \quad t \in [0,+\infty)$$

Boundary Conditions: Applied on the spatial domain boundary $\partial\Omega$.

$$\mathcal{B}_k[u(x,t)] = g_k(x,t), \quad x \in \Gamma_k \subseteq \partial\Omega$$

Initial Conditions: Needed for time-dependent problems to set initial state.

$$u(x,0) = u_0(x), \quad \left.\frac{\partial u(x,t)}{\partial t}\right|_{t=0} = u_0'(x)$$

Total Loss Function: Combines all physics constraints into one objective.

$$L(\omega) = L_p(\omega) + L_b(\omega) + L_i(\omega)$$

Universitatea Politehnica Timișoara

PINNs: Solving PDEs through Optimization

The loss terms guide the network to approximate a solution that respects:

- the PDE dynamics at interior points,
- the boundary values at domain edges,
- and the initial state at $t = 0$.

PDE Residual Loss: Enforces the differential equation at collocation points.

$$L_p(\omega) = \frac{1}{N_p} \sum_{i=1}^{N_p} \|\mathcal{D}[NN(x_i, t_i; \omega)] - f(x_i, t_i)\|^2$$

Boundary Loss: Penalizes mismatch between predicted and exact boundary values.

$$L_b(\omega) = \sum_k \frac{1}{N_b} \sum_{j=1}^{N_b} \|\mathcal{B}_k[NN(x_j, t_j; \omega)] - g_k(x_j, t_j)\|^2$$

Initial Loss: Ensures the network respects initial conditions of u and $\partial u/\partial t$.

$$L_i(\omega) = \|NN(x, 0) - u_0(x)\|^2 + \left\| \frac{\partial NN(x, 0; \omega)}{\partial t} - u_0'(x) \right\|^2$$

Universitatea Politehnica Timișoara

PINNs: Solving PDEs through Optimization

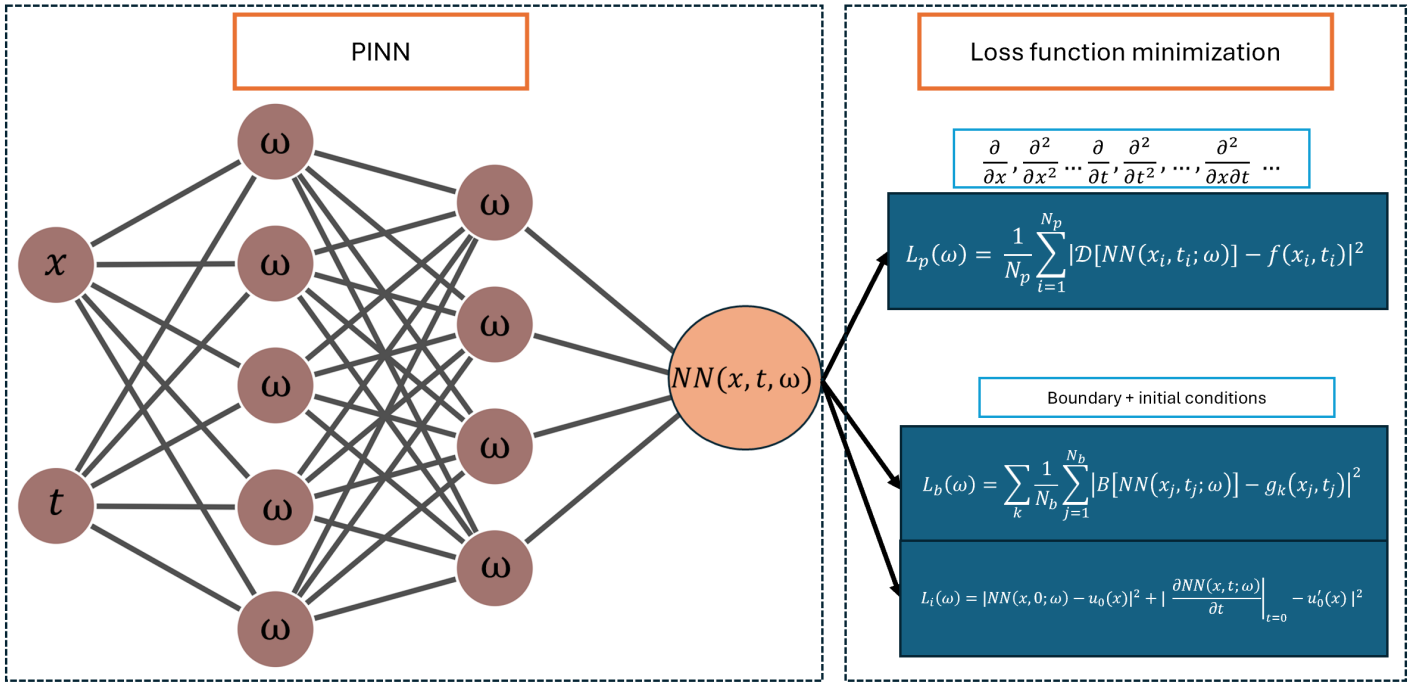


Figure 2: The network minimizes the residuals of the governing equation, boundary conditions, and initial conditions through loss functions $L_p(\omega)$, $L_b(\omega)$, and $L_i(\omega)$, respectively.

Universitatea Politehnica Timișoara

Why PINNs Are Different from Traditional Neural Networks

Traditional Neural Networks:

- Require labeled data: $(x^{(i)}, y^{(i)})$
- Learn to interpolate from training data
- Generalize only within the domain of seen data

Physics-Informed Neural Networks (PINNs):

- Do not require a labeled dataset
- Instead, use physical laws (PDEs) to generate training signals
- Can extrapolate to unseen domains by obeying the governing equations
- Directly minimize residuals of the PDE + boundary/initial conditions

Universitatea Politehnica Timișoara

Problem Setup: Solving an ODE with PINNs

We consider the first-order ODE:

$$\frac{du}{dt} = \sin(t) - u, \quad u(0) = 1$$

Analytical solution:

$$u(t) = \frac{1}{2}(\sin(t) - \cos(t)) + \frac{3}{2}e^{-t}$$

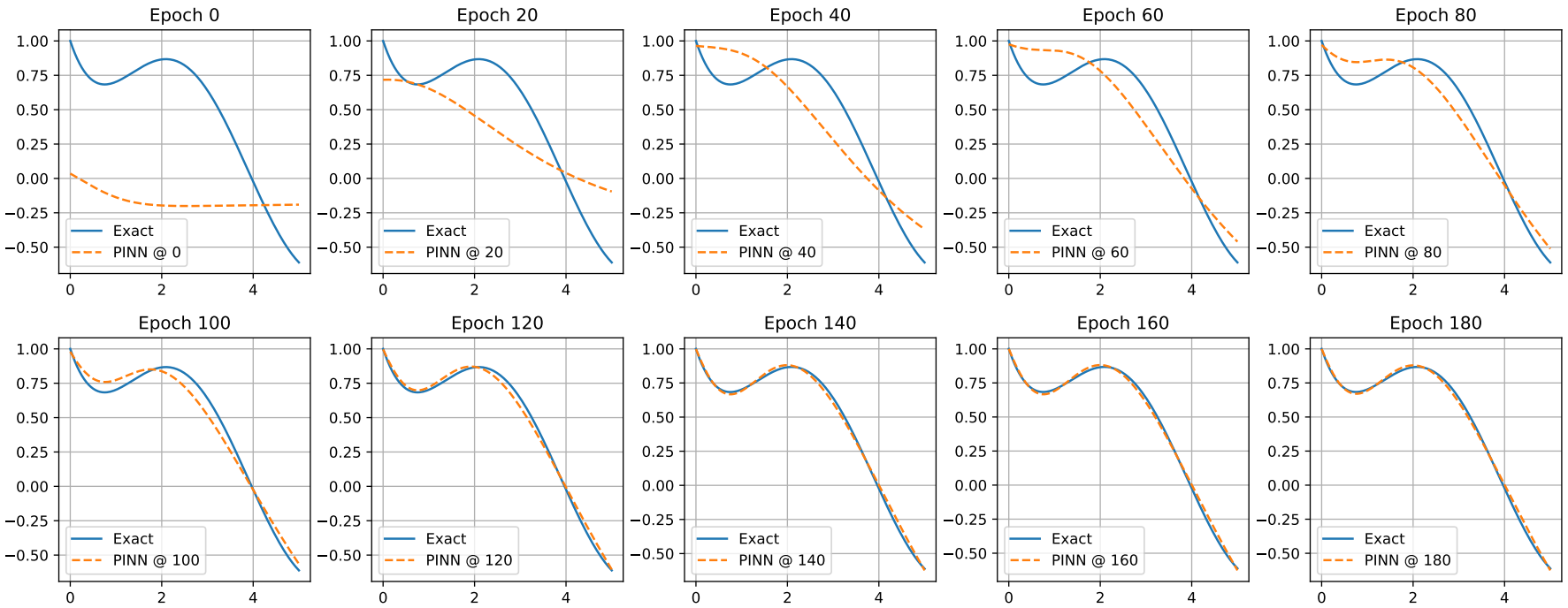
- Domain: $t \in [0, 5]$
- Used for benchmarking PINNs against Euler, RK2, RK4

PINN Setup:

- Neural network: 2 layers, 128 neurons each, Tanh activation
- Physics loss = PDE residual + initial condition
- Trained for 10k and 20k epochs on collocation points in $[0, 5]$

Universitatea Politehnica Timișoara

Learning Dynamics of the PINN



- PINN solution shown every 20 epochs, from epoch 0 to 180
- Initial residual is large; network gradually learns the dynamics
- Visible convergence around epoch 100–150

Universitatea Politehnica Timișoara

Accuracy Comparison

Method	MSE	Max Error
Euler	1.9888×10^{-5}	8.4372×10^{-3}
RK2	1.7121×10^{-9}	7.8139×10^{-5}
RK4	1.0440×10^{-18}	1.8441×10^{-9}
PINN @10k	1.8149×10^{-7}	1.6839×10^{-3}
PINN @20k	1.1398×10^{-8}	4.0000×10^{-4}

Table 1: Comparison of mean squared and maximum error for all methods.

Observations:

- RK4 yields highest accuracy (used as benchmark)
- PINN improves significantly with longer training
- PINNs learn a **continuous function** over time
- Can be evaluated at any t — even outside training domain
- Traditional methods compute only on fixed grids

Universitatea Politehnica Timișoara

Lane-Emden Equation: A Showcase for PINNs

- The Lane-Emden equation is a dimensionless form of the Poisson equation used in astrophysics to model polytropic stellar structures.
- It describes the structure of a self-gravitating, spherically symmetric star in hydrostatic equilibrium, where pressure and density obey the relation: $P \propto \rho^{1+1/n}$.

$$\frac{1}{\xi^2}\frac{d}{d\xi}\left(\xi^2\frac{d\theta}{d\xi}\right) + \theta^n = 0$$

- Here, ξ is the dimensionless radius, $\theta(\xi)$ is the normalized density, and n is the polytropic index.
- Boundary conditions at the stellar center:

$$\theta(0) = 1, \quad \theta'(0) = 0$$

- Only a few analytical solutions exist:

$$\theta_{n=0} = 1 - \frac{\xi^2}{6}, \quad \theta_{n=1} = \frac{\sin\xi}{\xi}, \quad \theta_{n=5} = \frac{1}{\sqrt{1+\frac{\xi^2}{3}}}$$

Historical references: Lane (1870), Emden (1907), Chandrasekhar (1957)

PINN Formulation for the Lane-Emden Equation (1/2)

- The classical Lane-Emden equation models stellar structure under a polytropic assumption:

$$\frac{d^2\theta}{dt^2} + \frac{2}{t}\frac{d\theta}{dt} + \theta^n = 0$$

- We reformulate it as a 2D PINN problem:

$$\theta(t, n), \quad \text{with } t = \text{radius}, \quad n = \text{polytropic index}$$

  This enables the neural network to generalize across families of stellar models, not just one.

- Derivatives $\partial\theta/\partial t$ and $\partial^2\theta/\partial t^2$ are computed via automatic differentiation.
- The residual function enforced at each collocation point (t_i, n_j) is:

$$R(t_i, n_j) = \frac{\partial^2\theta}{\partial t^2} + \frac{2}{t_i}\frac{\partial\theta}{\partial t} + \theta^{n_j}$$

Universitatea Politehnica Timișoara

PINN Loss Formulation for the Lane-Emden Equation

- The total loss combines:
  - the residual of the PDE
  - boundary conditions at $t = 0$

$$\mathcal{L} = \mathcal{L}_{\text{residual}} + \lambda_1 \mathcal{L}_{\theta(0,n)} + \lambda_2 \mathcal{L}_{\frac{\partial\theta}{\partial t}(0,n)}$$

$$\mathcal{L}_{\text{residual}} = \frac{1}{N_b} \sum_{i=1}^{m} \sum_{j=1}^{p} R(t_i, n_j)^2$$

$$\mathcal{L}_{\theta(0,n)} = \sum_{j=1}^{p} (\theta(0, n_j) - 1)^2$$

$$\mathcal{L}_{\frac{\partial\theta}{\partial t}(0,n)} = \sum_{j=1}^{p} \left(\frac{\partial\theta}{\partial t}(0, n_j) \right)^2$$

Here, λ_1 and λ_2 are weighting coefficients controlling the emphasis on boundary conditions relative to the residual. These may be tuned manually or learned adaptively during training. The conditions reflect physical requirements:

$$\theta(0, n) = 1, \quad \frac{d\theta}{dt}(0, n) = 0$$

Universitatea Politehnica Timișoara

PINN Solutions to the Lane-Emden Equation

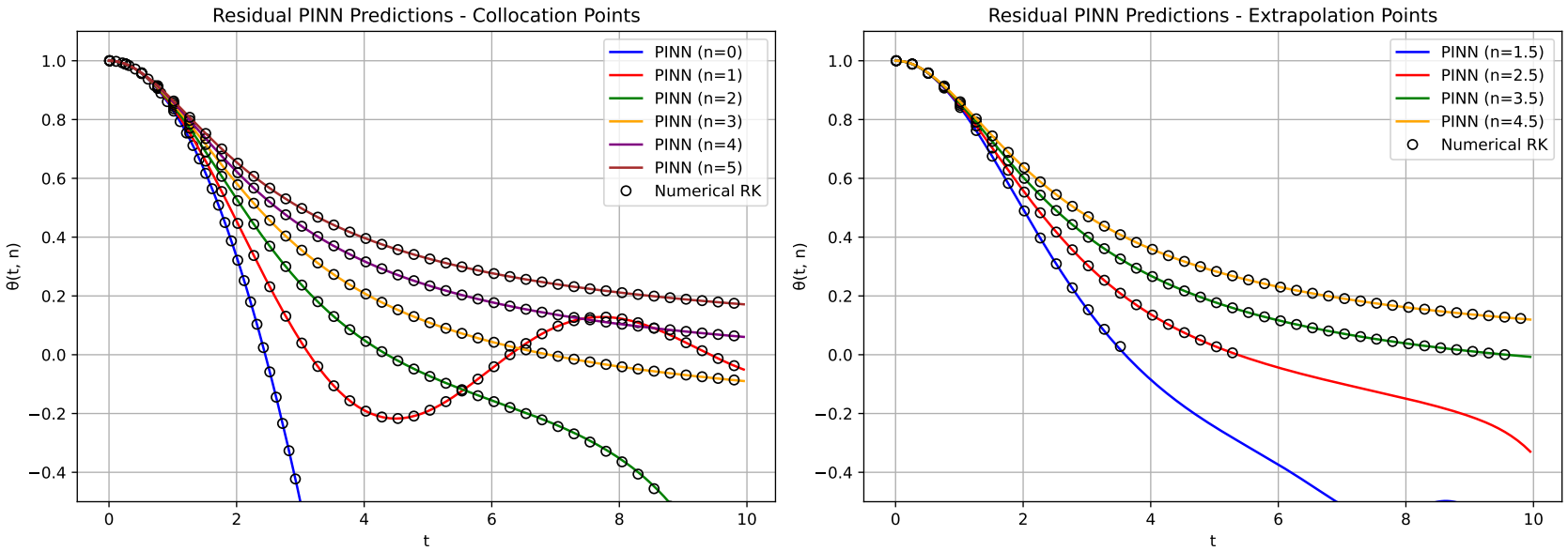


Figure 3: Residual-based PINN predictions (solid lines) versus classical Runge-Kutta (RK) solutions (dashed lines) for the Lane-Emden equation. **Left:** PINN interpolates accurately over collocation points for integer indices $n = 0$ to 5. **Right:** The PINN generalizes to non-integer indices $n = 1.5$ to 4.5 beyond the trained region. PINNs exhibit strong interpolation and extrapolation capabilities for singular ODEs.

Source: A.-I. Mohut, C.-A. Popa, *Residual-Based PINNs for Accurate Solutions of the Lane-Emden Equation*, in preparation (2025).

Universitatea Politehnica Timișoara

Spectral PINN: 1D Laplace Eigenproblem

- We consider the eigenvalue problem:

$$-\frac{d^2u}{dx^2} = \lambda u, \quad x \in (0,1), \quad u(0) = u(1) = 0$$

- The true eigenvalues and eigenfunctions are:

$$\lambda_n = n^2\pi^2, \quad u_n(x) = \sin(n\pi x), \quad n = 1, 2, 3, \ldots$$

- The PINN is trained to minimize:

$$\mathcal{L} = \|u''(x) + \lambda u(x)\|^2 + \|u(0)\|^2 + \|u(1)\|^2$$

- We optimize both $u_\theta(x)$ and λ jointly.
- Goal: observe how well the PINN captures different eigenmodes and how frequency affects convergence.

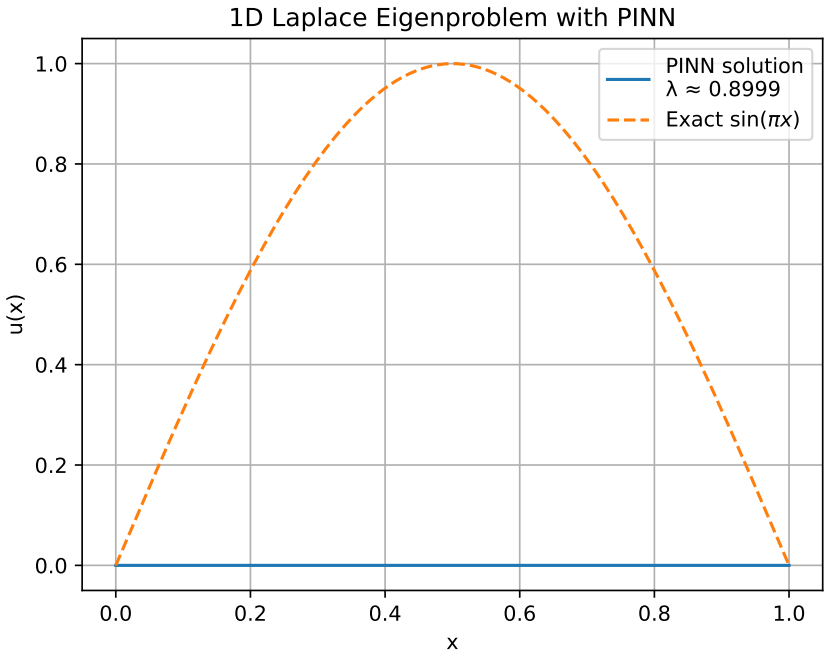
Universitatea Politehnica Timișoara

Spectral PINN Failure Case: Trivial Solution

- The loss function penalizes deviation from the PDE and the Dirichlet boundary conditions:

$$\mathcal{L} = \|u''(x) + \lambda u(x)\|^2 + \|u(0)\|^2 + \|u(1)\|^2$$

- We optimize both the function $u_\theta(x)$ and the eigenvalue λ jointly using automatic differentiation and Adam optimizer.
- However, in this example, the PINN collapsed to a trivial solution $u(x) \approx 0$, which minimizes the loss but is not an eigenfunction.



Spectral PINN: Normalization & Implicit BCs

- Instead of enforcing boundary conditions via the loss, we embed them into the neural network output via an **ansatz**:

$$u(x) = x(1-x) \cdot \hat{u}_\theta(x) \Rightarrow u(0) = u(1) = 0 \text{ automatically}$$

- To avoid trivial solutions $u(x) \approx 0$, we normalize the output:

$$\mathcal{L}_{\text{norm}} = \left(\|u\|^2 - 1\right)^2$$

- The total loss becomes:

$$\mathcal{L} = \|u''(x) + \lambda u(x)\|^2 + \alpha \cdot \mathcal{L}_{\text{norm}}$$

  where α is a weighting parameter (e.g., 100).

- This yields a valid eigenfunction shape and eigenvalue without explicitly enforcing boundary conditions.

Universitatea Politehnica Timișoara

Spectral PINN: Normalization & Implicit BCs

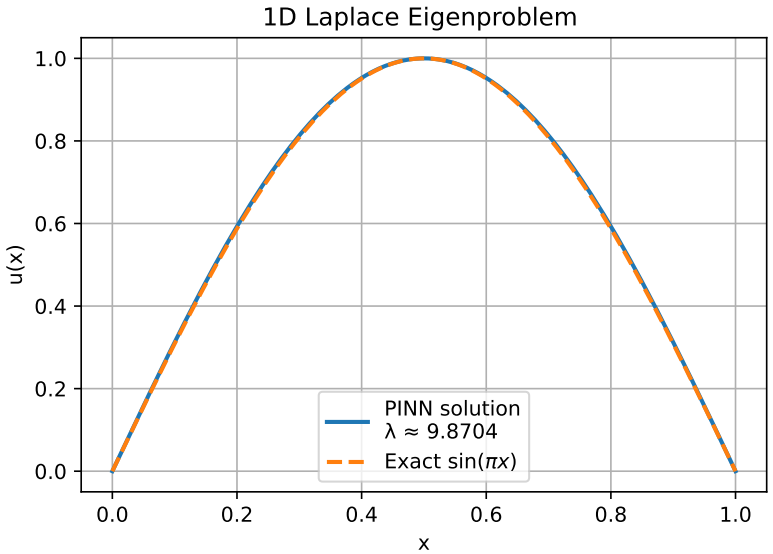
- Eigenfunctions are only defined up to sign: both $\sin(\pi x)$ and $-\sin(\pi x)$ are valid.
- We align the sign during training using a **phase constraint**:

$$\mathcal{L}_{\text{phase}} = \text{ReLU}(-u(0.5))^2$$

- For fair comparison with ground truth, we normalize:

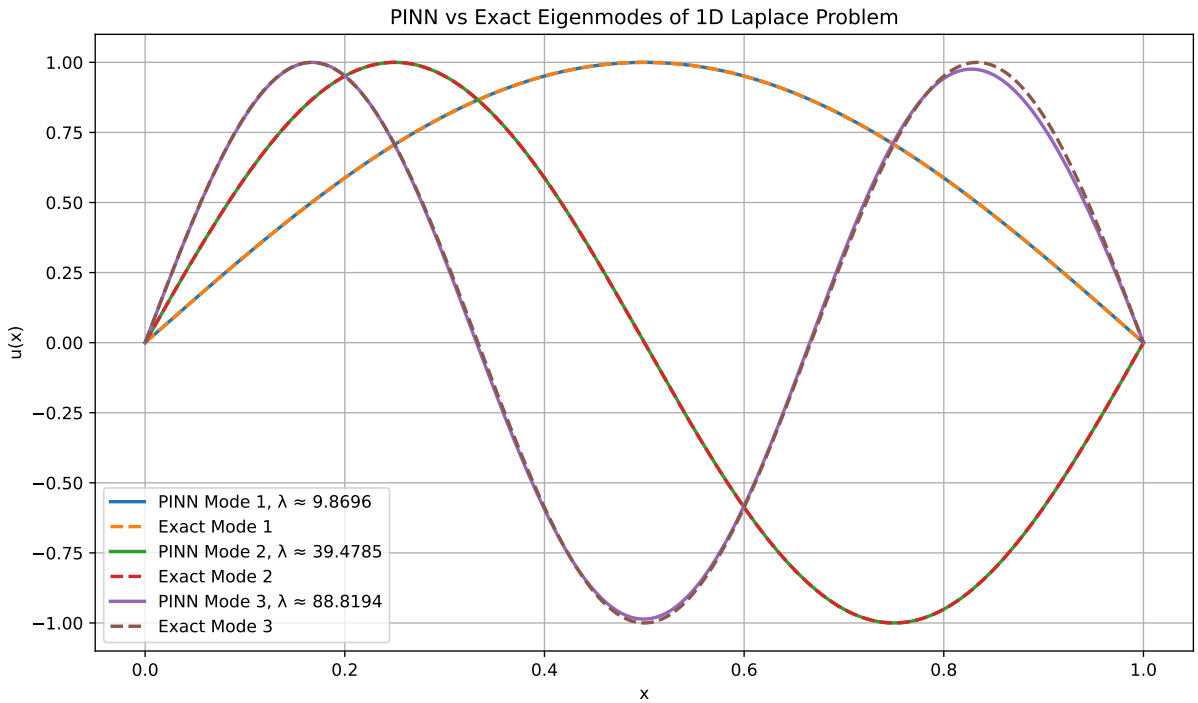
$$u(x) \leftarrow \frac{u(x)}{\max |u(x)|}$$

- The PINN now learns the correct frequency and shape, with clean output thanks to ansatz and normalization.



Spectral PINN: Learning Higher Eigenmodes

- To compute higher modes ($n > 1$), we train separate PINNs with:
  1. **Implicit BCs:** Ansatz $u(x) = x(1-x) \cdot \hat{u}_\theta(x)$
  2. **Normalization:** Enforce $\|u\|_{L^2} \approx 1$
  3. **Orthogonality:** Enforce $\langle u_n, u_k \rangle \approx 0$ for all $k < n$
- Sign ambiguity is expected (both $\sin(n\pi x)$ and $-\sin(n\pi x)$ are valid eigenfunctions). We postprocess by flipping u_n to align with the ground truth.



Universitatea Politehnica Timișoara

Eigenvalue Problem: Setup and Numerical Methods

Problem:

$$-u''(x) = \lambda u(x), \quad x \in [0, 2], \quad u(0) = 0, \quad u'(2) = 0$$

- Analytical solution: $u(x) = \sin\left(\frac{\pi x}{4}\right), \quad \lambda = \left(\frac{\pi}{4}\right)^2 \approx 0.6169$

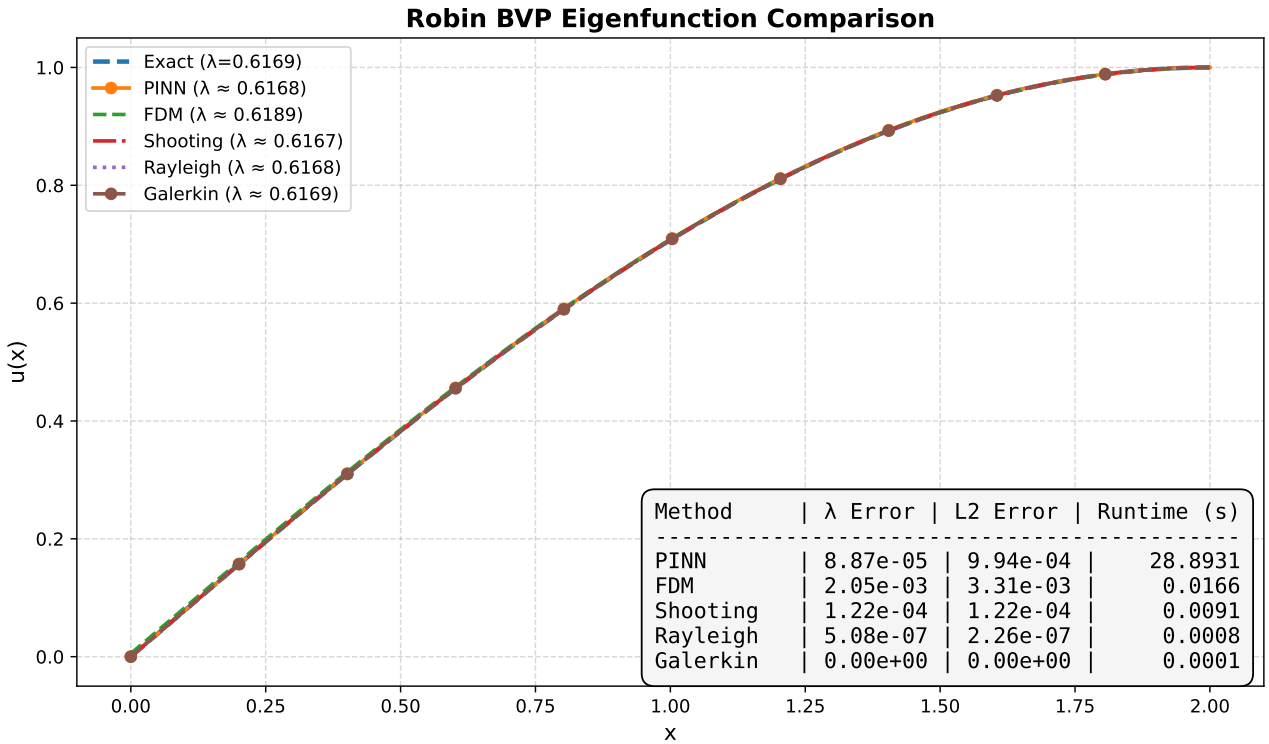
Methods Compared:

- **PINN** (Physics-Informed Neural Network)
- **FDM** (Finite Differences)
- **Shooting Method**
- **Rayleigh Quotient**
- **Galerkin Method**

Each method aims to compute the lowest eigenvalue λ_1 and associated eigenfunction $u(x)$, compared to the known analytical ground state.

Universitatea Politehnica Timișoara

Numerical Results and Accuracy



- All methods approximate the ground state eigenvalue accurately.
- Galerkin and Rayleigh methods nearly exact for this specific problem.

Universitatea Politehnica Timișoara

Strengths and Weaknesses of Methods

Method	**Strengths**	**Weaknesses**
PINN	Flexible, can handle complex geometries and physics; learns λ automatically	High runtime; requires tuning and many epochs
FDM	Easy to implement; fast runtime	Less accurate near boundaries; needs fine grids
Shooting	Very accurate for 1D problems; fast	Hard to generalize to higher dimensions or multiple eigenmodes
Rayleigh	Extremely fast; accurate with good trial function	Accuracy depends heavily on choice of trial function
Galerkin	Can achieve exact result with appropriate basis	Requires symbolic or analytic knowledge of solution space

Table 2: Qualitative comparison of numerical methods

Universitatea Politehnica Timișoara

Spectral Bias in PINNs: Motivation & Setup

- **Goal:** Demonstrate the **spectral bias** of PINNs — their tendency to learn low-frequency components of a solution faster than high-frequency ones.
- We construct a known multi-frequency solution using only odd modes:

$$u(x) = \sum_{n=1,3,5,7,9} \frac{1}{n^2} \sin(n\pi x)$$

- We plug this into:

$$-u''(x) = f(x), \quad x \in (0,1), \quad u(0) = u(1) = 0$$

to obtain a custom ODE with known forcing term $f(x)$, ensuring full control over the solution.

Universitatea Politehnica Timișoara

PINN Setup and Frequency Tracking

- We solve the equation using a PINN with no ansatz:
  - Boundary conditions $u(0) = u(1) = 0$ are imposed directly via the loss.
- The total loss includes:

$$\mathcal{L} = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{BC}}$$

- During training, the PINN solution is projected onto the Fourier basis:

$$\text{coeff}_n = \langle u_\theta, \sin(n\pi x) \rangle$$

  and compared to the true amplitudes $a_n = \frac{1}{n^2}$.
- A mode is considered converged when:

$$\left| \frac{\text{coeff}_n - a_n}{a_n} \right| < 1\% \quad \text{for at least 100 epochs}$$

Universitatea Politehnica Timișoara

Results: Convergence of Fourier Modes

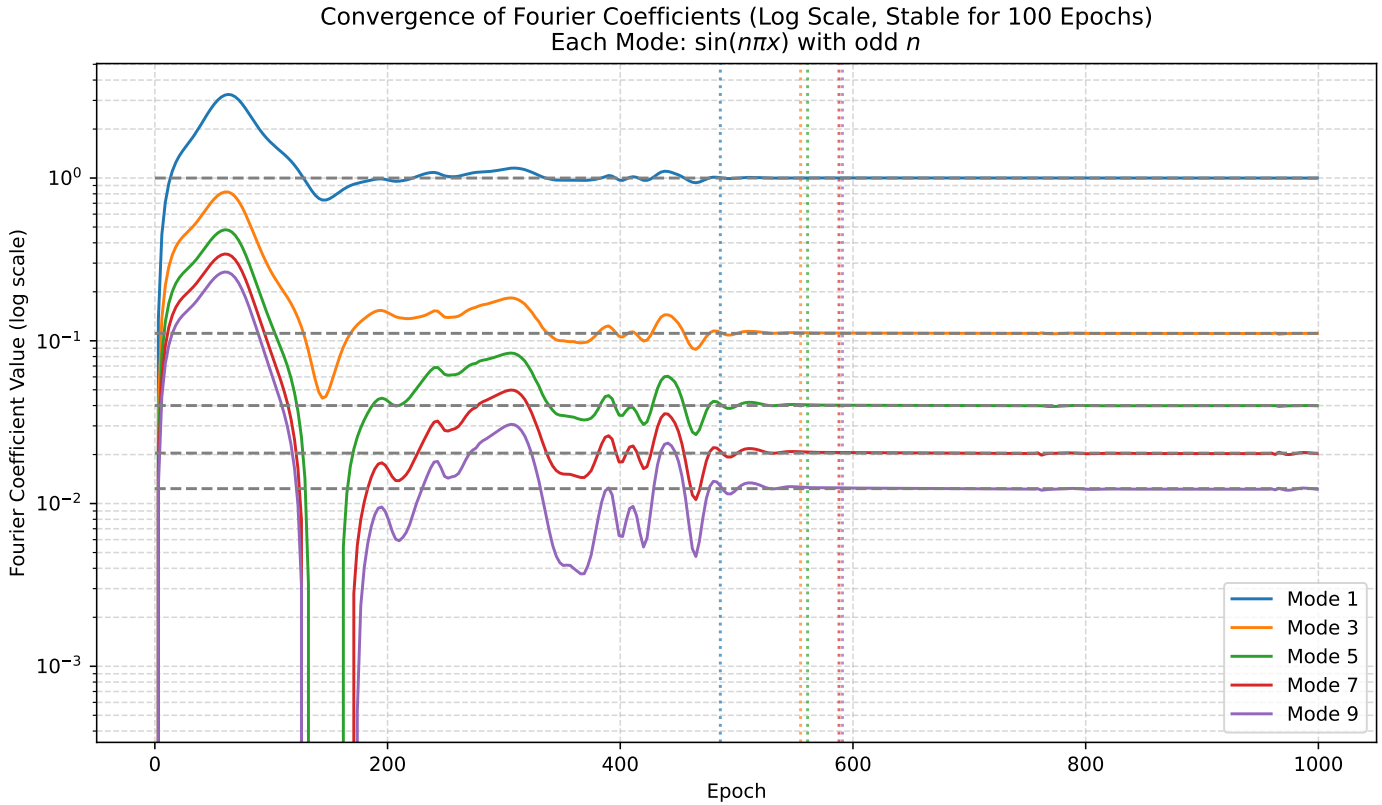


Figure 4: Learned Fourier amplitudes (solid lines) compared to target values (gray dashed). Dotted vertical lines show when each mode's error drops and stays below 1% for 100 epochs.

Interpreting Spectral Bias in PINNs

- **Spectral bias observed:** lower-frequency modes (e.g., $\sin(\pi x)$) converge first.
- Higher modes (e.g., $\sin(7\pi x), \sin(9\pi x)$) take significantly longer to stabilize.
- This behavior reveals a frequency-ordering in how PINNs absorb information.
- **This is spectral bias:**
  - A known behavior of neural networks trained via gradient descent
  - More pronounced with smooth activations (e.g., `tanh`)
- **Why it matters:**
  - Explains why PINNs may struggle with high-frequency PDEs
  - Suggests using enriched input encodings (e.g., Fourier features, positional encoding)

Universitatea Politehnica Timișoara

Random Fourier Feature (RFF) Embedding for PINNs

- **Motivation:** Due to spectral bias, PINNs learn low-frequency features faster — making high-frequency learning inefficient.
- **Solution:** Use a **Random Fourier Feature (RFF)** embedding to enrich the input space with high-frequency content.
- The original input vector $x = (t, n)$ is transformed as:

$$\Phi(x) = \begin{bmatrix} \cos(Bx) \\ \sin(Bx) \end{bmatrix}$$

  where:

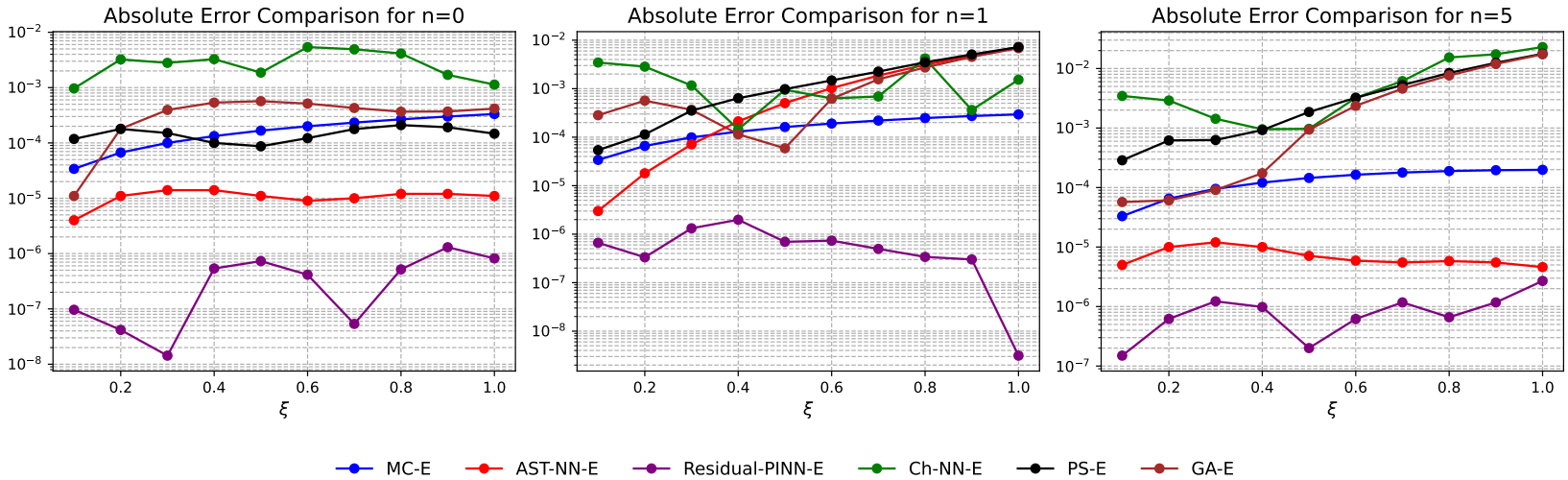
  - $B \in \mathbb{R}^{D \times 2}$, with $B_{ij} \sim \mathcal{N}(0, s^2)$
  - s is a hyperparameter controlling the frequency scale

- The PINN is then trained on $\Phi(t, n)$ instead of (t, n), allowing the network to access multi-scale representations.
- **Effect:** Accelerates learning of oscillatory (high-frequency) components that would otherwise converge slowly due to spectral bias.

Universitatea Politehnica Timișoara

Improving Lane-Emden PINNs with RFF Embeddings

- **Goal:** Reduce absolute error in PINN solutions of the Lane-Emden equation by mitigating spectral bias.
- **Method:** Use a Random Fourier Feature (RFF) embedding to enrich the input (t, n) space with high-frequency components.
- **Observation:** RFF-PINNs yield substantial accuracy improvements across all tested polytropic indices n, including singular and stiff regimes.



Key Takeaways

- **Neural networks** can approximate any continuous function — but nonlinear activations are critical to their power.
- **PINNs** embed differential equations directly into the loss function — allowing solution of ODEs/PDEs without labeled data.
- PINNs are **flexible** but **optimization-heavy** — high runtime and convergence sensitivity are key challenges.
- **Spectral PINNs** solve eigenvalue problems by combining physics constraints, normalization, and ansatz-driven architectures.
- **Spectral bias** causes PINNs to learn low-frequency modes first — high-frequency accuracy requires enriched encodings.
- **Random Fourier Features (RFFs)** improve high-frequency learning and address limitations of smooth activations like `tanh`.

Universitatea Politehnica Timișoara

State of the Art: PINNs and Spectral Approaches

- **PINNs** have been extended to:
  - Multiphysics (e.g., Navier-Stokes, Maxwell, Schrödinger)
  - Irregular and multi-domain geometries
  - Inverse problems and data assimilation
- **Key Innovations:**
  - *Adaptive weighting* of loss terms
  - *Domain decomposition* for scaling to large domains
  - *Fourier feature embeddings* for frequency-rich solutions
  - *Complex PINN architectures*
- **Limitations:**
  - Struggle with stiff or chaotic dynamics
  - Require extensive hyperparameter tuning
  - Slow convergence compared to classical solvers

Universitatea Politehnica Timișoara

References I

M. Raissi, P. Perdikaris, G. Karniadakis.
Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear PDEs.
Journal of Computational Physics, 378, 686–707, 2019.

M. Tancik et al.
Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains.
NeurIPS, 2020.

G. Cybenko.
Approximation by Superpositions of a Sigmoidal Function.
Mathematics of Control, Signals and Systems, 1989.

K. Hornik.
Approximation Capabilities of Multilayer Feedforward Networks.
Neural Networks, 1991.

A. J. Meade, A. A. Fernandez.
The Numerical Solution of Linear Ordinary Differential Equations by Feedforward Neural Networks.
Mathematical and Computer Modelling, 19:1–25, 1994.
`https://api.semanticscholar.org/CorpusID:122130053`

Universitatea Politehnica Timișoara

References II

B. van Milligen, V. Tribaldos, J. Jiménez.
Neural Network Differential Equation and Plasma Equilibrium Solver.
Physical Review Letters, 75:3594–3597, 1995.
10.1103/PhysRevLett.75.3594

I. Lagaris, A. Likas, D. Fotiadis.
Artificial Neural Networks for Solving Ordinary and Partial Differential Equations.
IEEE Transactions on Neural Networks, 9(5):987–1000, 1998.
10.1109/72.712178

I. Lagaris, A. Likas, D. Papageorgiou.
Neural-Network Methods for Boundary Value Problems with Irregular Boundaries.
IEEE Transactions on Neural Networks, 11(5):1041–1049, 2000.
10.1109/72.870037

Universitatea Politehnica Timișoara

Thank you!

andrei.mohut@student.upt.ro

Universitatea Politehnica Timișoara