

Lecture 3: Introduction to PINNS

Chris Budd¹

¹University of Bath

Seattle, June 2025

Some papers to look at

- Raissi, M., P. Perdikaris, and G. E. Karniadakis. 2019. “Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations.” *Journal of Computational Physics* 378: 686–707. <https://doi.org/https://doi.org/10.1016/j.jcp.2018.10.045>.
- Grossman et. al. *Can PINNS beat the FE method?*
- Shin et. al. *On the convergence of PINNS for linear elliptic and parabolic PDEs*
- Wang et. al. *When and why PINNS fail to train: a neural tangent kernel perspective*
- Kiyani *Which Optimizer Works Best for Physics-Informed Neural Networks and Kolmogorov-Arnold Networks?*
- Chen et. al. *Sidecar: A Structure-Preserving Framework for Solving Partial Differential Equations with Neural Networks*
- Russell et. al. *Two-point boundary value problems*

Motivation: Solving ODEs and PDEs

Seek to solve ODE/PDE problems of the form

$$u_t = F(x, u, \nabla u, \nabla^2 u) \quad \text{with BC}$$

eg. ODE (IVP or BVP)

$$\frac{du}{dt} = u^2, u(0) = 1, \quad -\epsilon^2 \frac{d^2 u}{dx^2} + u = 1, \quad u(0) = u(1) = 0.$$

eg. PDE

$$-\Delta u = f(x), \quad iu_t + \Delta u + u|u|^2 = 0, \quad u_t = \Delta u + f(x, u) \quad x \in R^n$$

Classical Method 1. Finite Differences

Work with a set of **point values** for a function **never with a function directly**
IVP/BVP:

$$U_j^n \approx u(n\Delta t, j\Delta x), \quad \Delta t, \Delta x \ll 1$$

eg.

$$u_t = u_{xx} + u^3$$

IMEX Crank-Nicholson method:

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} = \frac{1}{2\Delta x^2} \left(U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1} + U_{j+1}^n - 2U_j^n + U_{j-1}^n \right) + (U_j^n)^3.$$

- Have error estimates of the form

$$\|U_j^n - u(n\Delta t, j\Delta x)\| < C(u)\Delta t^p \Delta x^q, \quad \Delta t, \Delta x \rightarrow 0$$

- Can reduce $C(u)$ and increase p, q using an adaptive approach.
- Lots of software
- Have to recover the function from the point values
- Awkward in higher dimensions!

Classical Method 2: Finite Elements

Express $u(x, t)$ as a Galerkin approximation:

$$u(t, x) \approx U(t, x) = \sum_{i=0}^N U_i(t) \phi_i(x)$$

with $\phi_i(x)$ **Not** globally differentiable *locally supported, piece-wise polynomial spline functions*

- Work with a function $U(x) \in H^1$
- Require U to satisfy a *weak form* of the PDE (see Lecture 4)
- Have guaranteed error estimates of the form

$$\|u - U\|_{H^1} < C(u)N^{-\alpha}$$

- Lots of software (see Lecture 4)
- Can reduce $C(u)$ and increase α using an adaptive approach.
- Awkward in higher dimensions!

Classical Method 3: Collocation

Express $u(x, t)$ as a function approximation:

$$u(t, x) \approx U(t, x) = \sum_{i=0}^N U_i(t) \phi_i(x)$$

with $\phi_i(x)$:

Locally differentiable locally supported, piece-wise polynomial spline functions such as Lagrange polynomial interpolants

- Work with a function $U(x) \in C_{loc}^n$
- Require U to satisfy the PDE **exactly** at a carefully chosen set of **collocation points**

- Have guaranteed error estimates of the form

$$\|u - U\|_{C_2} < C(u)N^{-\alpha}$$

- Can reduce $C(u)$ and increase α using VERY carefully chosen **collocation points**
- Implemented in `python` as `solvebvp`, `Colsys`
- Impossible in higher dimensions!

Issues with classical methods

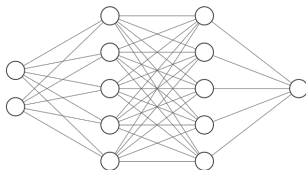
- Well tested and good convergence theories
- Accuracy of the computation depends crucially on the choice and shape of the mesh points
- Generally require practice to implement (although FireDrake etc. makes this easier)
- Two-fold curse of dimensionality

PINNS

Physics Informed Neural Networks for solving PDEs: advertised as "Mesh free methods".

Use a Deep Neural Net of width W and depth L to give a nonlinear functional approximation to $u(\mathbf{x})$ with input x .

$$y(\mathbf{x}) = DNN(\mathbf{x})$$



$y(\mathbf{x})$ is constructed via a combination of linear transformations and nonlinear/semi-linear activation functions.

Example: Shallow 1D neural net

$$y(\mathbf{x}) = \sum_{i=0}^{W-1} c_i \sigma(a_i \mathbf{x} + \mathbf{b}_i)$$

Often take

$$\sigma(z) = \text{ReLU}^3(z) \equiv z_+^3, \quad \text{or} \quad \sigma(z) = \tanh(z)$$

As a result we can define y_{zz} for every point z

Operation of a 'traditional' PINN

Want to solve a ODE/PDE in \mathbf{x}

- PINNS are similar to collocation methods
- Assume that $y(\mathbf{x})$ has strong regularity eg. C^2
- Differentiate $y(\mathbf{x})$ **exactly** using the chain rule
- Evaluate the **PDE residual** at N_r **collocation points** \mathbf{X}_i , :chosen to be uniformly spaced, or **samples from a random distribution**
- Train the neural net to minimise a **loss function** L combining the PDE residual and boundary and initial conditions. May use **Epochs linked to the random training samples**.

Eg 1. Solution of regular two-point BVPs by PINNs

Consider the two-point BVP with Dirichlet boundary conditions:

$$-u_{xx} = f(x, u, u_x), \quad x \in [0, 1] \quad u(0) = a, \quad u(1) = b.$$

Define output of the PINN by $y(x)$ and residual $r(x) := y_{xx} + f(x, y, y_x)$.
Train the coefficients θ of the NN by minimising the loss function

$$L = \frac{1}{N_r} \sum_i^{N_r} |r(X_i^r)|^2 + \frac{\beta}{2} (|y(0) - a|^2 + |y(1) - b|^2),$$

where $\{X_i^r\}_i^{N_r}$ are the **collocation points** (possibly randomly) placed in $(0, 1)$.

Eg 2. Solution of regular second order IVPs by PINNs

Consider the second order IVP with initial conditions:

$$-u_{xx} = f(x, u, u_x), \quad x \in [0, 1] \quad u(0) = a, \quad u'(0) = b.$$

Define output of the PINN by $y(x)$ and residual $r(x) := y_{xx} + f(x, y, y_x)$.
Train the coefficients θ of the NN by minimising the loss function

$$L = \frac{1}{N_r} \sum_i^{N_r} |r(X_i^r)|^2 + \frac{\beta}{2} (|y(0) - a|^2 + |y'(0) - b|^2),$$

where $\{X_i^r\}_i^{N_r}$ are the **collocation points** (possibly randomly) placed in $(0, 1)$.

Eg 3. Solution of regular parabolic PDEs by time-stepping PINNs

Consider the **semilinear parabolic PDE** with Dirichlet boundary conditions:

$$u_t = u_{xx} + f(x, u, u_x), \quad x \in [0, 1] \quad u(0) = a, \quad u(1) = b$$

and initial condition

$$u(x, 0) = u_0(x).$$

Implicit time-stepping scheme $U^n(x) \approx u(n\Delta t, x)$.

$$\frac{U^{n+1}(x) - U^n(x)}{\Delta t} = U_{xx}^{n+1} + f(x, U^{n+1}, U_x^{n+1}) \equiv F(U^{n+1}).$$

- Start with $U^0(x) = u_0(x)$
- For $n > 0$: Define output of the PINN by $y(x)$ and residual

$$r(x) := U^n(x) + \Delta t F(y).$$

- The PINN is trained by minimising the loss function

$$L = \frac{1}{N_r} \sum_i^{N_r} |r(X_i^r)|^2 + \frac{1}{2} (|y(0) - a|^2 + |y(1) - b|^2),$$

where $\{X_i^r\}_i^{N_r}$ are the **collocation points** placed in $(0, 1)$.

- Set $U^{n+1} = y(x)$ and repeat.

Eg 4. Solution of regular parabolic PDEs by a full PINN

Consider the same semilinear parabolic PDE

$$u_t = u_{xx} + f(x, u, u_x), \quad x \in [0, 1] \quad u(0) = a, \quad u(1) = b.$$

Bold approach!

- Have NN with $y(t, x)$ a function of t and x
- Take N_r collocation points Z_i in **space and time**
- Residual $r = u_t - u_{xx} - f(x, u, u_x)$
- Minimise

$$L = \frac{1}{N_r} \sum_i^{N_r} |r(Z_i^r)|^2 + \frac{\beta}{2} (|y(0) - a|^2 + |y(1) - b|^2) \\ + \gamma \|y(0, x) - u_0(x)\|^2.$$

Problematic as space and time play very different roles in the PDE

Numerical results for: $-u'' = \pi^2 \sin(\pi x)$.

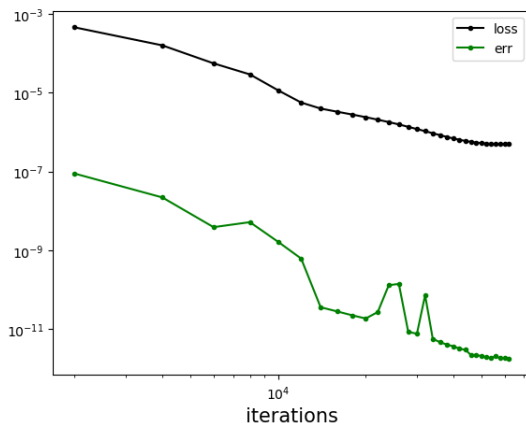


Figure: Residual based Loss and L^2 error of the PINN solution for $N_r = 100$

Numerical results: $u(x) = \sin(\pi x)$

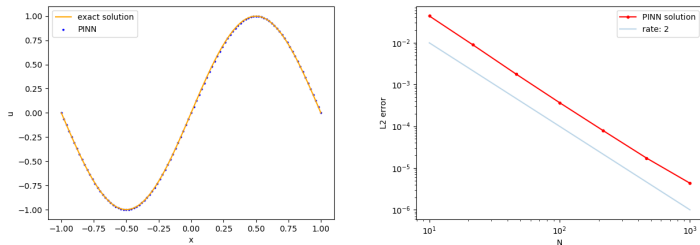


Figure: (Left) PINN with depth $L = 2$ and width $W = 30$. $N_r = 100$ uniformly distributed quadrature points, activation function: \tanh , optimizer: Adam with $lr = 1e - 3$ (Right) Convergence rate for 1st order interpolant

Eg 2. Singular Reaction-Diffusion Equation

Solve $-\varepsilon^2 u_{xx} + u = 1 - x$ on $[0, 1]$ $u(0) = u(1) = 0$

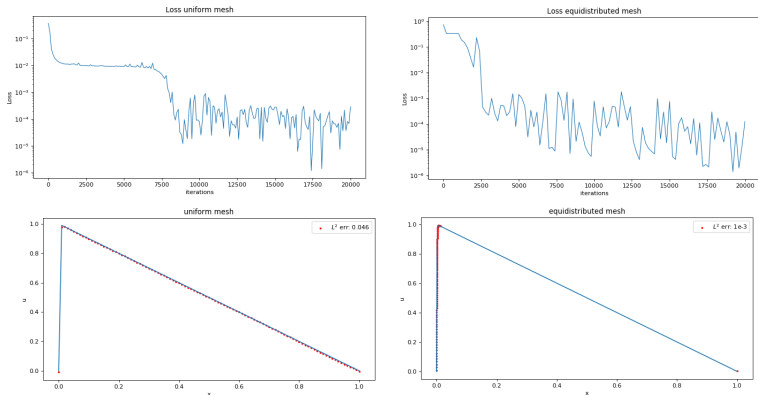


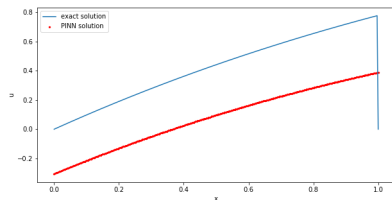
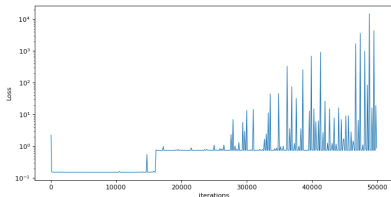
Figure: PINN (tanh) trained for 20000 epochs, $N_r = 101$, Adam optimizer with $lr = 1e - 3$. (left) Uniform collocation points (right) Adapted collocation points **much faster training**

Eg 3. Bad news: Convection-dominated equation

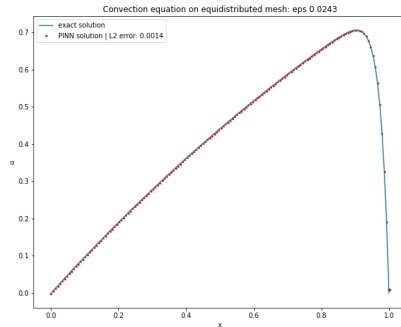
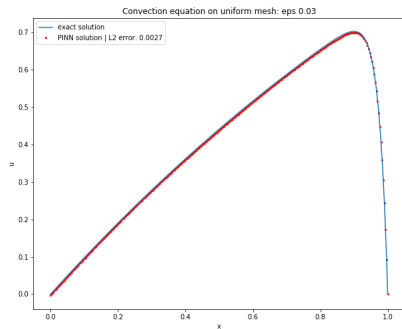
PINNs often fail to train when the solution of the BVP exhibits singular and convective behaviour [Krishnapriyan, Aditi et. al., (2021)]:

$$-\varepsilon u_{xx} + \left(1 - \frac{\varepsilon}{2}\right) u_x + \frac{1}{4} \left(1 - \frac{1}{4}\varepsilon\right) u = e^{-x/4} \text{ on } [0, 1] \quad u(0) = u(1) = 0$$

$$u(x) = \exp^{\frac{-x}{4}} \left(x - \frac{\exp^{-\frac{1-x}{\varepsilon}} - \exp^{-\frac{1}{\varepsilon}}}{1 - \exp^{-\frac{1}{\varepsilon}}} \right)$$



Numerical results: Convection Equation



General questions for consideration

A PINN is based on a NN so we can expect in theory to achieve the expressivity and highly accurate approximation results. But do we see this in practice?

- ① When do and don't PINNs work, and why?
- ② How do these answers depend on (i) the problem (ii) choice of activation function, optimisation, collocation points, conditioning etc
- ③ Can we develop a useful convergence theory for a PINN using tools from approximations theory, bifurcation theory, numerical analysis etc.
- ④ How does a PINN compare to a finite element method?

Convergence theory for PINNS

[Shin, Darbon and Kaniardarkis, 2020], [Jiao, Lai, Lo, Wang, Yang, 2024]

- PINN error is a combination of approximation error and training/optimization error
- Show that a PINN (depth L width W) can be constructed with low approximation error which reduces as the complexity of the PINN increases.
- Hope that the optimization error can be reduced to acceptable levels.
- Try things out on simple problems

BUT

- **Non convex (no guarantee of uniqueness or convergence of training)**
- PINNs are nonlinear function approximators. **No equivalent of Cea's lemma giving a bound on the solution error.**
- Solutions can sometimes have no connection to reality!

Simple convergence theory for any collocation method including a PINN

Consider the DE:

$$-u_{xx} = f(x), \quad u(0) = f, u(1) = g.$$

There exists an exact solution using a Green's function $G(x, y)$

$$u(x) = G * f = \int_0^1 G(x, y) f(y) dy.$$

Suppose we have N collocation points Y_i then we can approximate the integral by a quadrature

$$u(x) = \sum_{i=1}^N w_i G(x, Y_i) f(Y_i) + \mathcal{O}(N^{-\alpha})$$

for some α depending on the weights w_i and the smoothness of u .

Now let $U(x)$ be any solution of the collocation problem

$$-U_{xx}(Y_i) = f(Y_i).$$

It follows that

$$\sum_{i=1}^N w_i G(x, Y_i) f(Y_i) = - \sum_{i=1}^N w_i G(x, Y_i) U_{xx}(Y_i) = U(x) + \mathcal{O}(N^{-\beta})$$

for some β depending on the weights w_i and the smoothness of u .
Hence we have the convergence result

$$u(x) = U(x) + \mathcal{O}(N^{-\alpha}) + \mathcal{O}(N^{-\beta}).$$

Can do better .. see [Russell et. al.]

Implementation of collocation methods and PINNS

A standard collocation method for the problem $-u_{xx} = f(x)$ is a linear approximation of the form:

$$u(x) \approx U(x) = \sum a_j \phi_j(x).$$

The collocation conditions at the points X_i lead to a linear equation:

$$\sum_j A_{i,j} a_j = f_i$$

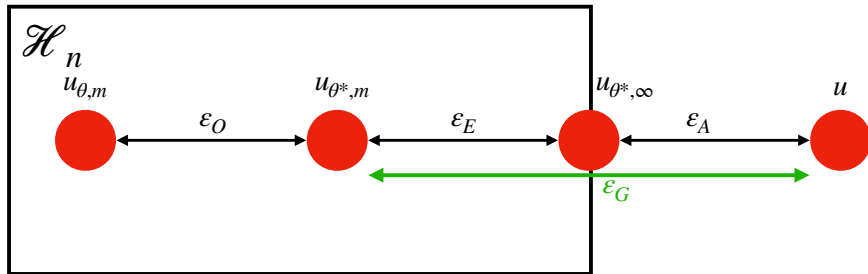
$$A_{i,j} = -\epsilon^2 \phi_j(X_i) + \phi_j(X_i), \quad f_i = f(X_i).$$

These linear equations are (in general) well conditioned and have a unique solution

Compare with the PINN formulation for the same problem

- PINN is a nonlinear approximation
- It may have better expressivity
- The collocation equations become an optimisation problem
- We may not find a good optimum

Detailed PINN Convergence Theory 1



Detailed PINN Convergence Theory 2

Underlying solution: $u(x)$

H_n : Class of functions approximated by NN with n degrees of freedom

- $u_{\theta,m}$: Approximation found in practice after some training. Is less accurate than
- $u_{\theta_m}^*$: Approximation obtained by perfect optimisation with finite collocation points. Is less accurate than
- $u_{\theta^*,\infty}$: Approximation obtained by perfect optimisation with infinite collocation points.

Measured error is $\|u_{\theta,m} - u\|$

- $\|u_{\theta,m} - u_{\theta^*,m}\|$: Optimisation error \mathcal{E}_O : Hard to control and depends on the optimiser and the initialisation: See Lecture 2
- $\|u_{\theta^*,m} - u_{\theta^*,\infty}\|$: Estimation error \mathcal{E}_E : Main results on this
- $\|u_{\theta^*,\infty} - u^*\|$: Approximation error \mathcal{E}_A : See Lecture 2

Training data

[Shin et. al] prove

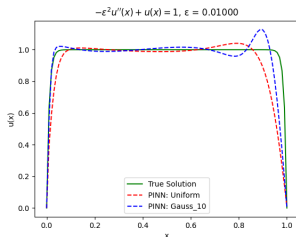
Theorem *If there is a random set of training data and n samples are chosen for training. Then the optimal minimiser h_n over this set converges to the best approximation \hat{h} as $n \rightarrow \infty$.*

Results 1: Aengus Roberts

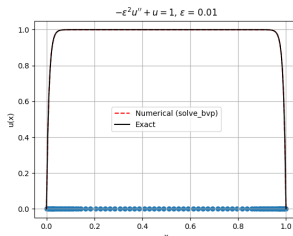
Compare standard collocation and a PINN for the ODE using ADAM

$$-\epsilon^2 u_{xx} + u = 1, \quad u(0) = u(1) = 0.$$

This has a boundary layer at $x = 0, x = 1$ of width ϵ .

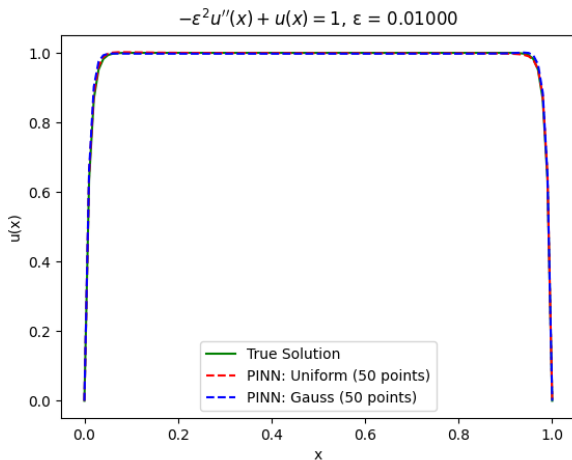


PINN



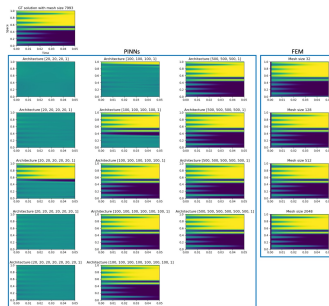
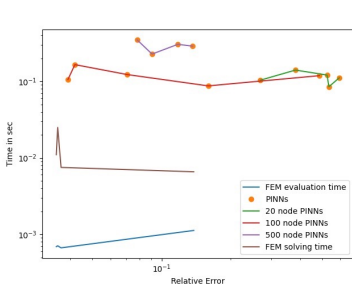
Collocation

Using the LBFGS optimiser



Results by Grossman et. el. 1

(Solution of the Allen-Cahn Equations)



Results by Grossman et. el. 2

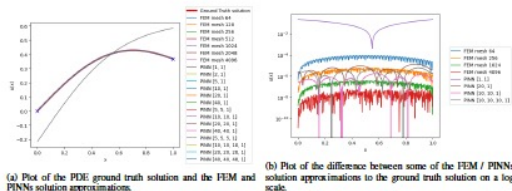


Figure 1: Plot for 1D Poisson equation solution.

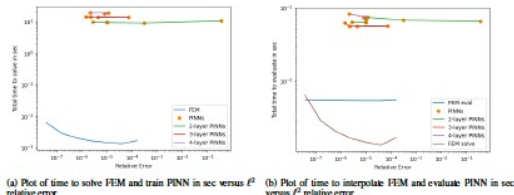


Figure 2: Plot for 1D Poisson equation of time in sec versus L^2 relative error.

How and why do PINNS struggle to train .. and how can we fix this?

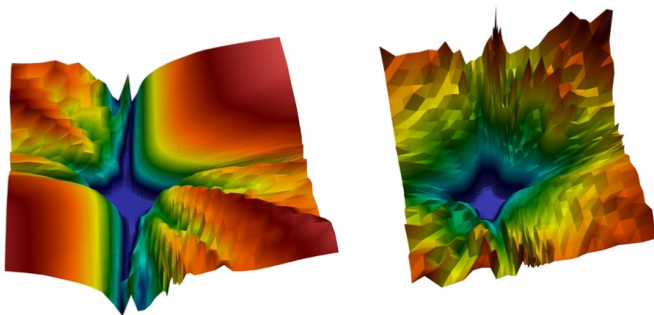
A subject of intensive research! See [Wang et. al.] for a summary:
Observe

- PINNS train especially badly when they have high frequency features
- Loss function has multi-scale interactions, leading to ill-conditioning and stiffness in the gradient flow dynamics and a stringent stability requirement on the loss rate
- Standard networks show **spectral bias** and cannot learn high frequencies. This has been seen for example in **weather forecasting**

They advocate the use of **Neural Tanjent Kernels** in the optimisation process.

[Kiyani et. al.] extend this with the development of better optimisers eg. **Broyden methods**, for PINNS

Loss landscape is highly non-convex, [Kiyani et. al.]



Bringing in the physics

Many differential equations have **conservation laws** eg.

$$u_t = u_{xx}, \quad u : \text{periodic on } [0, 1] \implies \int_0^1 u \, dx = C$$

$$iu_t + u_{xx} + u|u|^2 = 0, \quad u : \text{periodic on } [0, 1] \implies \int_0^1 |u|^2 \, dx = C$$

A standard PINN **Does not usually learn the conservation law!**

However, it can be built into the architecture either (i) through the loss function or (better) (ii) through the architecture see [Chen et. al.]

This generally leads to much more accurate solutions.

Application 1: PINNs for Euler's equations on \mathbb{T}^2

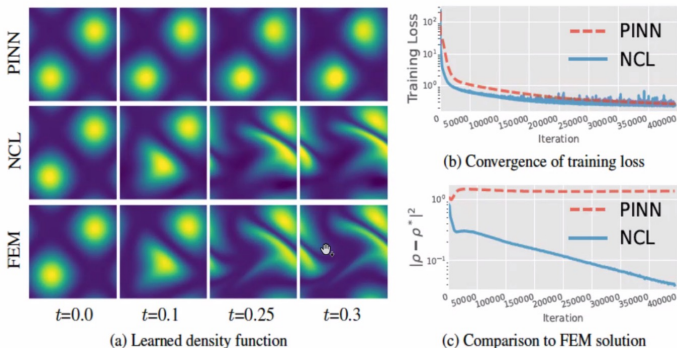


Figure 4: While both PINN and NCL models minimize the training loss effectively and fit the initial conditions, the PINN fails to learn the dynamics of the advected density. When compared to a gold standard FEM solution, our NCL model nicely exhibits linear convergence to the solution.