Physics-Informed Neural Networks (PINNs)

Theory and Applications in Computational Physics

Eklavya Goyal

TU München

January 29, 2025

What are Physics-Informed Neural Networks (PINNs)?

Definition:
PINNs are a class of neural networks that incorporate the laws of physics (PDEs) directly into the training process.

PINNs embed physical laws through:

$$\mathcal{L}_{\text{total}} = \lambda_1 \mathcal{L}_{\text{physics}} + \lambda_2 \mathcal{L}_{\text{data}}, \tag{1}$$

where λ_1 and λ_2 are weighting factors.

Physics-Informed ~~Neural Networks~~ Loss Functions

The physics loss ensures that the NN prediction satisfies the governing PDE:

$$\mathcal{L}_{\text{physics}} = \frac{1}{N_r} \sum_{i=1}^{N_r} |\mathcal{F}(\hat{u}(x_i, t_i))|^2 , \tag{2}$$

where $\mathcal{F}$ is the residual of the PDE, $\hat{u}(x, t)$ is the NN prediction of the solution, and N_r is the no. of collocation points.
The data loss ensures that the neural network predictions are close to observed or boundary/initial condition values:

$$\mathcal{L}_{\text{data}} = \frac{1}{N_d} \sum_{i=1}^{N_d} |u(x_i, t_i) - \hat{u}(x_i, t_i)|^2 , \tag{3}$$

where $u(x, t)$ represents true observed values, and N_d is the number of data points.

Damped Harmonic Oscillator (DHO)

Under-damped state, i.e. $\delta < \omega_0$, where $\delta = \frac{\mu}{2m}$ and $\omega_0 = \sqrt{\frac{k}{m}}$

$$L = m\frac{d^2}{dt^2} + \mu\frac{d}{dt} + k \rightarrow Lu(t) = 0$$

Initial conditions:

$$x(0) = x_0 = 1$$
$$\dot{x}(0) = v_0 = 0$$

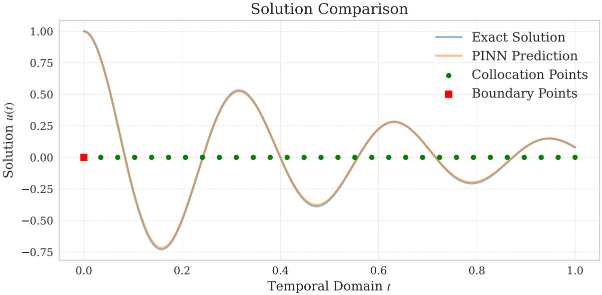
Constants:

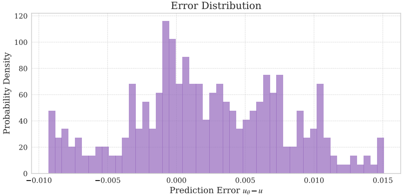
$$m = 1, \quad d = 2, \quad \omega_0 = 20,$$
$$\mu = 4 \quad k = 400$$

PINN Loss Function

$$\mathcal{L} = \underbrace{\frac{1}{N}\sum_{i}^{N}|LNN(t_i;\theta)|^2}_{\text{Physics residual}} + \underbrace{|NN(0;\theta) - x_0|^2 + |\frac{d}{dt}NN(0;\theta) - v_0|^2}_{\text{Initial conditions}}$$

Results - DHO





4 layer Fully-Connected network with 2 Hidden layers, each with 32 Neurons.
Tanh activation function. ADAM optimiser. $\text{lr} = 10^{-3}, 25k$ Epochs

DHO with noisy data and parameter inversion

Here, I tried to invert the PINN by first generating some noisy data, then letting ADAM optimize the value for mu.

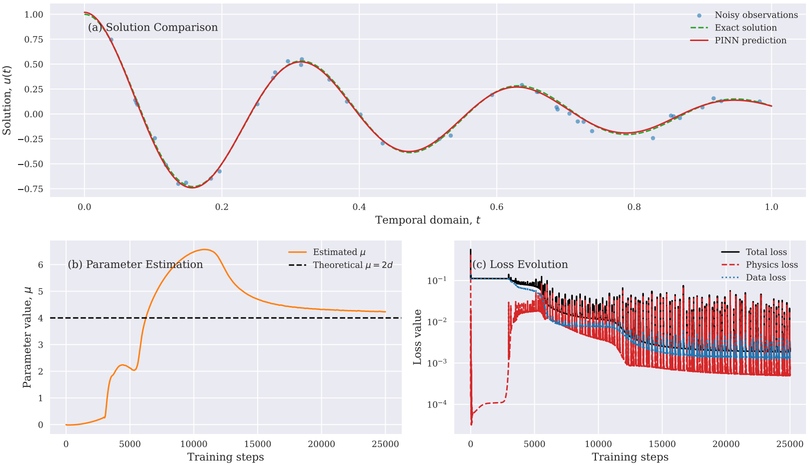
$$\mathcal{L}(\theta, \mu) = \frac{1}{N} \sum_{i}^{N} (L(\mu) NN(t_i; \theta))^2 + \frac{\lambda}{M} \sum_{j}^{M} (NN(t_j; \theta) - u_{obs}(t_j))^2$$

4 layer Fully-Connected network with 2 Hidden layers, each with 32 Neurons.
Tanh activation function. ADAM optimiser:

```
optimiser = torch.optim.Adam(list(pinn.parameters())+[mu],
lr = 1e-3)
```

lr $= 10^{-3}, 25k$ Epochs

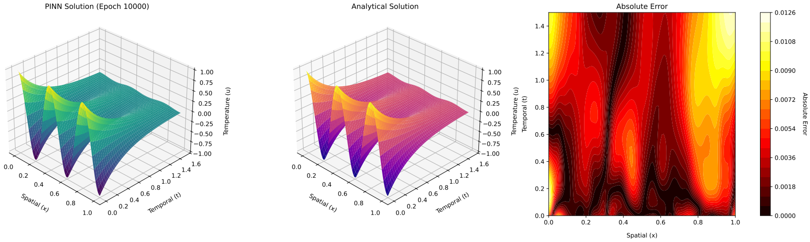
Results - DHO with noisy data, optimizing for μ



The estimated value μ_{est} converges to the theoretical value $\mu = 2d = 4$

1D Heat Equation

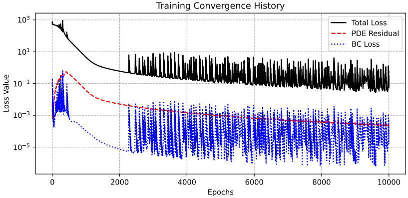
$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad (x,t) \in [0,L] \times [0,T], \alpha = 0.01$$



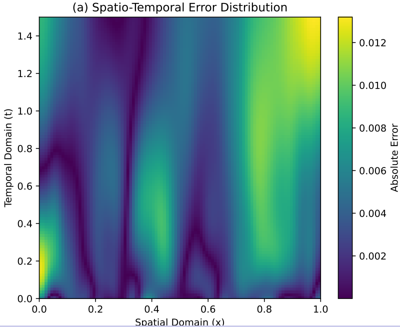
Initial and Boundary conditions

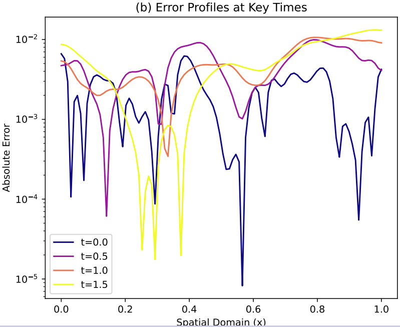
- **IC:** $u(x,0) = \cos(5\pi x)$
- **BC:** $\frac{\partial u}{\partial x}\big|_{x=0,x=L} = 0$ (Neumann BC's)

Results - 1D Heat equation



FCN with 4 Hidden layers, each with 64 Neurons.
Tanh. ADAM optimiser.
$lr = 10^{-3}, 10k$ Epochs. 5000 Collocation Points with 1000 points at the boundaries and 1000 at t=0.





2D Heat Equation

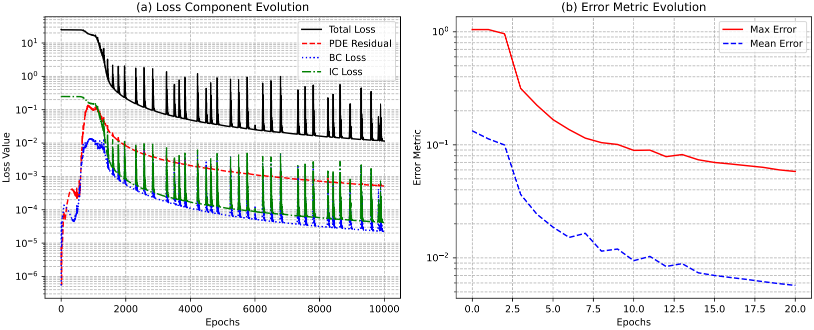
$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \quad (x, y, t) \in [0, L_x] \times [0, L_y] \times [0, T], \alpha = 0.01$$

Initial and Boundary conditions

- **IC:** $u(x, y, 0) = \cos(3\pi x) \cos(3\pi y)$
- **BC:** $\frac{\partial u}{\partial x} = \frac{\partial u}{\partial y} = 0$ (Neumann BC's)

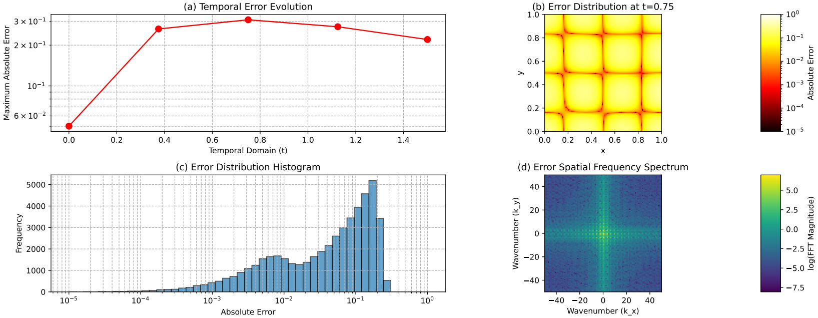
Here's a quick animation for this model.

2D Heat Equation - Results I



FCN with 3 Hidden layers, each with 64 Neurons.
SiLU activation. ADAM optimiser. lr $= 10^{-3}$ with lr scheduler, $\gamma = 0.9$ and step size $= 10^3$. $10k$ Epochs. 5000 Collocation Points, 5000 points at the boundaries$(x = 0, y = 0, x = L_x, y = L_y)$ and 5000 at t=0.

2D Heat Equation - Results II



where k_x and k_y are computed as follows: (2D FFT)

$$e(x, y) = u_{\text{num}} - u_{\text{exact}} \quad \hat{E}(k_x, k_y) = \mathcal{F}\{e(x, y)\}$$

Ideas for improvement: More collocation points, 1 or 2 more layers or 128 neurons in each hidden layer.