Case Studies on Physics-Informed Neural Networks
AM 205 Final Project

Florian Juengermann Marius Merkle Marcel Torne

Contents

1 Introduction

Differential equations are ubiquitous in science and engineering and their solutions are crucial for many applications as they can help us understand various scientific disciplines described by applied mathematics. Unfortunately, differential equations are notoriously hard to solve. For almost all practically relevant differential equations, exact analytical solutions are unknown. Therefore, numerical solvers that yield solutions with accuracy up to a certain degree have been developed for several decades. Despite these tremendous efforts, high-performance computing clusters may still need prohibitively long (weeks to months) to carry out sophisticated numerical simulations. Therefore, the design of new efficient algorithms for differential equations is of primary importance and could have revolutionary effects in both academia and industry.

With the rise of artificial intelligence, the application of machine learning algorithms to scientific computing has received a lot of attention. Recently, physics-informed neural networks (PINNs) have been proposed as a new framework to solve differential equations numerically with artificial neural networks [Raissi et al., 2019].

Throughout this report, we will explore several methods of modeling both ordinary and partial differential equations (ODE/PDE) with PINNs. We'll apply those to three different problems out of which we have seen two in the graduate course AM 205: Numerical Methods, for which this is the final report.

2 Physics-Informed Neural Networks (PINN)

PINNs have sparked a growing number of works in the domain of physics-informed deep learning since their introduction by Raissi et al. [2019]. They combine two powerful concepts of deep learning. First, it is well-known that a single hidden layer network can approximate any continuous function up to arbitrary accuracy [Hornik et al., 1989]. Therefore, we can construct an artificial neural network (ANN) that receives independent variables such as time and space coordinates of a differential equation as an input and outputs the variable of interest (e.g. temperature or pressure). Second, several deep learning libraries such as PyTorch have automatic differentiation capabilities Paszke et al. [2017]. That is, we can differentiate the output with respect to the input and construct derivative terms that come up in a differential equation.

Following the notation of Raissi et al. [2019], let us define a general non-linear differential equation of the form

$$u_t + \mathcal{N}[u] = 0 \tag{1}$$

where $u(x,t)$ denotes the solution, u_t is the first time derivative and $\mathcal{N}$ is a non-linear operator on u given by the differential equation. Given some spatio-temporal domain $(x,t) \in [x_{min}, x_{max}] \times [0,T]$ a boundary-initial value problem can be constructed by combining the differential equation in Equation 1 with boundary conditions (BC) $u(x_{min},t) = u_{b_{min}}(t)$, $u(x_{max},t) = u_{b_{max}}(t)$ and an initial condition (IC) $u(x,0) = u_0(x)$.

Next, we can discretize the domain to obtain a labeled dataset on the spatio-temporal boundary $\{x_i^b, t_i^b, u_i^b\}_{i=1}^{N_b}$ given by the initial and boundary conditions and set of collocation points $\{(x_i^c, t_i^c)\}_{i=1}^{N_c}$ for which the differential equation residual should vanish

$$f(x_i^c, t_i^c) := u_t(x_i^c, t_i^c) + \mathcal{N}[u(x_i^c, t_i^c)] \tag{2}$$

We can then formulate a PINN as solving an optimization problem to minimize the following loss function:

$$\mathcal{L} = \underbrace{\frac{1}{N_c}\sum_{i=1}^{N_c}\left(f(x_i^c, t_i^c)\right)^2}_{\mathcal{L}_{DE}} + \underbrace{\frac{1}{N_b}\sum_{i=1}^{N_b}\Big(u(x_i^b, t_i^b) - u_i^b\Big)^2}_{\mathcal{L}_{IC/BC}} \tag{3}$$

where the first term corresponds to the differential equation residual and the second term to the initial and boundary conditions

2.1 Soft versus Hard Constraints

Note that the optimization problem given in Equation 3 has multiple objectives to satisfy at the same time. We will refer to that scenario as enforcing ICs and BCs as soft constraints. It would be great if the PINN would minimize each of the terms individually, but the network can only be trained in a way to minimize the *sum* of them. This can bring several difficulties in the training process and can lead to undesired solutions where the initial and boundary conditions are not properly satisfied. The magnitude of the individual loss terms often differs and multiple solutions have been proposed to weigh the loss terms differently [van der Meer et al., 2021]. However, this weighted loss requires a lot of manual tuning of the weights.

Instead, a more elegant approach is to reparametrize the neural network output in such a way that ICs and BCs are satisfied by construction. An exemplary architecture is shown in Figure 1. For example, to satisfy the differential equation $\frac{\mathrm{d}x}{\mathrm{d}t} = cx$ along with $x(t_0) = x_0$, we can transform the neural network output to $x(t) = NN(t) \cdot t + x_0$ which simplifies to $x(0) = x_0$ in the case of $t = 0$ and therefore satisfies the initial condition by construction. Then, the loss function simplifies to

$$\mathcal{L} = \underbrace{\frac{1}{N_c}\sum_{i=1}^{N_c}\left(f(x_i^c, t_i^c)\right)^2}_{\mathcal{L}_{DE}} \tag{4}$$

so the PINN optimizes a single loss term related to the differential equation residual.

We will make use of this reparametrization trick for the three-body problem in section 3 and the convection-diffusion equation in section 4.

2.2 Weighting Function

For time-dependent systems, oftentimes, errors accumulate over time so errors early on have a larger influence than errors of the same magnitude at a later stage. For that reason, we want to weigh the differential equation residuals according to the time. We motivate this by analyzing the simple ODE $\frac{\mathrm{d}x}{\mathrm{d}t} = g(t, x)$ as it was done in Flamant et al. [2020]. Let $x^*(t)$ describe the solution to the ODE and let $\hat{x}(t)$ be the numerical approximation. The global error $\epsilon(t) = x^*(t) - \hat{x}(t)$ can be bounded by an exponential factor of the maximum local error $\epsilon_L^*(t) = \max_{t_0 \leq t' \leq t} |\epsilon_L(t')|$:

$$|\epsilon(t)| \leq \frac{\epsilon_L^*(t)}{L_g}\left(e^{L_g(t-t_0)} - 1\right) \tag{5}$$

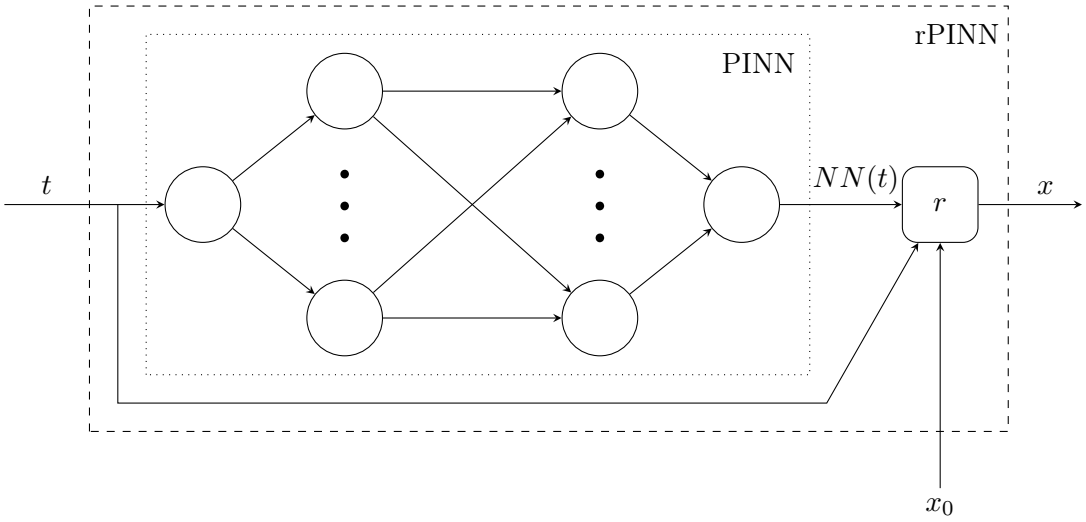


Figure 1: Modified network that enforces initial or boundary conditions through an additional layer.

where L_g is the Lipschitz constant of g. A proof is given in Flamant et al. [2020].

So, to minimize the global error, the local error at the beginning should be exponentially lower than at the end. The local error at a collocation point (x, t) is governed by the residual of the differential equation $f(x, t)$. By applying the weighting function $e^{-\lambda t}$ to each residual loss, the total residual loss becomes

$$\mathcal{L}_{DE} = \frac{1}{N_c} \sum_{i=1}^{N_c} e^{-\lambda t_i^c} \left(f(x_i^c, t_i^c)\right)^2 \tag{6}$$

where λ is a hyperparameter we choose problem-dependent.

2.3 Curriculum Learning

In some instances, learning the solution for the entire domain at once can be challenging so we use curriculum learning to learn the function in small steps. Curriculum learning as introduced by Bengio et al. [2009] builds on the intuition that it helps to start with a simple objective and then increase the complexity with time. In the case of an ODE, the PINN predictions for large times t are most likely far from the truth at the beginning of the training. So there is little use in optimizing the function so that it locally fulfills the ODE when the output needs to be corrected on a macro scale first. In subsubsection 3.2.3 we have a look at such an example in more detail.

To deal with that, we first train the network on a subset of the collocation points close to the known initial conditions and then gradually expand the training domain. For example, we could first only consider collocation points with $t_i^c \leq 0.05 \cdot t_{max}$ representing only the first 5% of simulation time.

3 Three-Body Problem

As the first case study, we look at the asteroid collision problem discussed in homework 3 of the AM 205 course. In this setting, we want to predict the trajectory of an asteroid of negligible mass in the gravity field of the Earth and the Moon.

3.1 Problem Setup

For simplicity, we assume the asteroid's movement is restricted to the 2D plane of the circular orbits of Earth and Moon. We use a scaled and rotating frame of reference so that the Earth is at the fixed location $(0, 0)$ and the Moon is at the fixed location $(1, 0)$. For the asteroid, we observe its initial position $\boldsymbol{p_0} = (x_0, y_0)$ and its initial velocity $\boldsymbol{v_0} = (u_0, w_0)$.

The equations of motion can be derived from the *Jacobi integral* that stays constant over time:

$$J(x, y, u, w) = (x - \mu)^2 + y^2 + \frac{2(1-\mu)}{\sqrt{x^2 + y^2}} + \frac{2\mu}{\sqrt{(x-1)^2 + y^2}} - u^2 - w^2 \tag{7}$$

where $\mu = 0.01$ is the ratio of the Moon's mass to the total mass of the Earth and Moon. Now, the equations of motion are given by the system of ODEs:

$$\frac{\partial x}{\partial t} = -\frac{1}{2}\frac{\partial J}{\partial u} = u \tag{8}$$

$$\frac{\partial y}{\partial t} = -\frac{1}{2}\frac{\partial J}{\partial v} = w \tag{9}$$

$$\frac{\partial u}{\partial t} = w + \frac{1}{2}\frac{\partial J}{\partial x} = w + x - \mu - \frac{x(1-\mu)}{(x^2+y^2)^{\frac{3}{2}}} - \frac{(x-1)\mu}{((x-1)^2+y^2)^{\frac{3}{2}}} \tag{10}$$

$$\frac{\partial w}{\partial t} = -u + \frac{1}{2}\frac{\partial J}{\partial y} = -u + y - \frac{y(1-\mu)}{(x^2+y^2)^{\frac{3}{2}}} - \frac{y\mu}{((x-1)^2+y^2)^{\frac{3}{2}}} \tag{11}$$

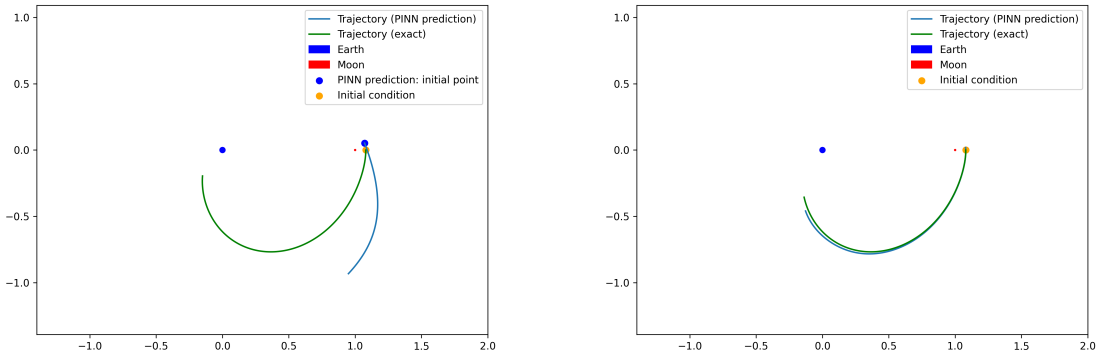
This formulation is beneficial for classical numerical methods, but not for PINNs. As there are only first-order derivatives, these ODEs can be solved using simple classical methods such as the forward Euler method. Implementing this formulation with PINN leads to a network that takes a time t as input and outputs four values x, y, u, w that must fulfill these four ODEs. However, note that the velocity in x-direction u is just the time derivative of x and w is the time derivative of y. As a neural network outputs an infinitely differentiable function, automatic differentiation directly gives the derivatives of x and y with respect to the input t. So instead, we construct a PINN that takes time t as input and outputs the position x, y of the asteroid at that time. This way, we only need to fulfill two ODEs:

$$\frac{\partial^2 x}{\partial t^2} = \frac{\partial x}{\partial t} + x - \mu - \frac{x(1-\mu)}{(x^2+y^2)^{\frac{3}{2}}} - \frac{(x-1)\mu}{((x-1)^2+y^2)^{\frac{3}{2}}} \tag{12}$$

$$\frac{\partial^2 y}{\partial t^2} = -\frac{\partial y}{\partial t} + y - \frac{y(1-\mu)}{(x^2+y^2)^{\frac{3}{2}}} - \frac{y\mu}{((x-1)^2+y^2)^{\frac{3}{2}}} \tag{13}$$

3.2 Optimizations

If we try to learn a solution to this ODE system, the results are fairly disappointing. Figure 2 shows one example of a learned trajectory after 500 epochs of training. We identified two main problems.



(a) no optimizations

(b) all optimizations

Figure 2: Trajectory learned by a PINN compared to an accurate numerical solution. (a) shows the result without optimizations, in (b) we use hard constraints for the initial condition, a weighting function for the loss, and curriculum learning.

First, this is a highly chaotic system, meaning that the slightest disturbance in the initial conditions can lead to a completely different trajectory. So we must make sure that our network precisely learns the initial conditions. For that, we use the reparameterization trick from subsection 2.1. Second, in this time-dependent system, errors accumulate. So an error at the beginning of the trajectory has a much bigger impact than an error towards the end. To compensate for that, we introduce a weighting function for the loss and implement curriculum learning.

3.2.1 Enforce Initial Conditions

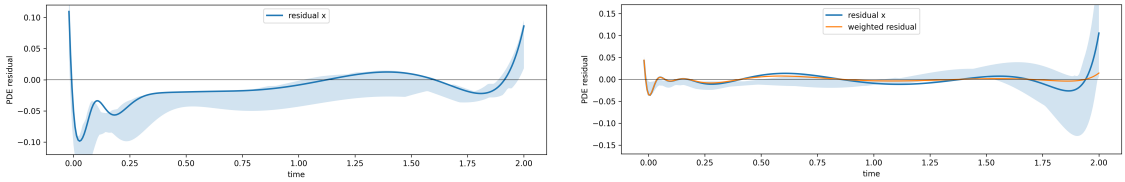
As shown in subsection 2.1, we can reparametrize the network output $\boldsymbol{NN}(t)$ to enforce that our initial conditions $\boldsymbol{p}(0) = \boldsymbol{p_0}$, $\frac{\partial \boldsymbol{p}}{\partial t} = \boldsymbol{v_0}$ are exactly met. We choose

$$\hat{\boldsymbol{p}}(t) = t^2 \cdot \boldsymbol{NN}(t) + t \cdot \boldsymbol{v_0} + \boldsymbol{p_0} \tag{14}$$

such that the initial conditions are satisfied by design.

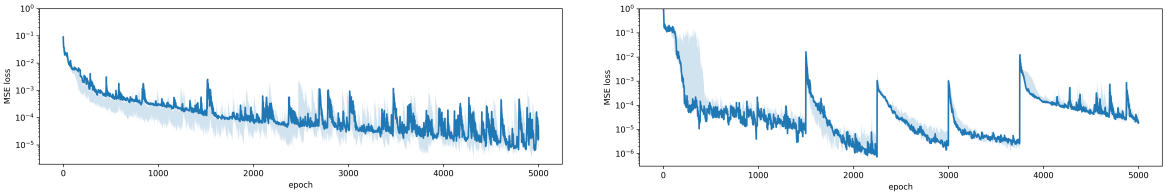
3.2.2 Loss-Weighting Function

In subsection 2.2 we gave an intuition that time-dependent systems with accumulating errors, local errors at the beginning are worse than errors at the end of the trajectory. In Figure 3a, however, we see that the residuals are especially high at the beginning of the trajectory. So we implement the weighting function for the loss as described in subsection 2.2 with parameter $\lambda = 1$. With that, we receive a significant reduction in the residual terms in the beginning. Figure 3b shows the results where the blue line indicates the residual and the orange line shows the residual after multiplying with the weighting function. As expected, the weighted residual stays roughly constant over the simulation time, while the pure residual is exponentially smaller in the beginning than in the end.



(a) without loss weighing (b) exponentially weighed loss

Figure 3: Residual of x (Equation 12) over simulation time t. (a) shows the result without weighing function. In (b) a negative exponential weighting function is used as described in subsection 2.2. The orange line describes the loss after multiplying with the weighing function. Results are from 10 runs, the lines show one of those runs, the shaded region indicates one standard deviation.



(a) regular training (b) curriculum learning

Figure 4: MSE loss over the training time for regular training (a) compared to curriculum learning (b). The blue line shows the result from one run, the shaded region indicates one standard deviation.

3.2.3 Curriculum Learning

As a final optimization, we implemented curriculum learning as described in subsection 2.3. We split up the simulation time into five chunks and train on the first $5\%, 15\%, 40\%, 65\%, 100\%$ of simulation time. Empirically, we found that it is beneficial to assign different numbers of epochs to the chunks, so we spend more training epochs on the first and the last chunks. Behavior during training is shown in Figure 4b. We see that each time we extend the training domain, the error spikes but then quickly decreases again. However, the loss plateaus at increasing levels. In comparison to the training without curriculum learning, we do not see an improvement in the loss. After 5000 epochs, the weighted MSE loss is $3.9 \pm 1.2 \cdot 10^{-5}$ compared to $1.6 \pm 0.8 \cdot 10^{-5}$ for regular training.

3.3 Error Analysis

After finishing training, we analyze the error in the trajectory (Figure 4b). In our case, we have an alternative numeric solution so we can directly observe the global error as seen in Figure 2. However, in general, we do not have access to this reference solution and must evaluate the error on internal metrics. The natural metric we can use is the residual of the differential equation. In Figure 3, we plot the residual of Equation 12 over the simulation time t which is a measure of the local error. The global error can probably be bounded by the integral of the absolute value of the residuals. For example, De Ryck and Mishra [2021] show an error bound for PINNs for the Kolmogorov PDEs. A general bound of the global error based on the residual is the topic of current research and beyond the scope of our project.

4 Ensemble Learning for the Convection-Diffusion Equation

Remark: These two approaches have been developed in collaboration with Pavlos Protopapas and his lab.

In this section, we would like to exploit the flexible framework of PINNs and outline two approaches that target ensemble learning. That is, instead of learning a particular instance of a differential equation, the goal is to efficiently and accurately generate solutions to an entire family of differential equations. Such a family may be described by any parameter, e.g. physical quantities such as the Reynolds number or material properties. The two covered ensemble learning approaches are solution bundles in subsection 4.2 and bottleneck freeze in subsection 4.3.

4.1 Problem Setup

We study the well-known convection-diffusion equation, which comes up in several applications including fluid dynamics in the form of the Navier-Stokes equation

$$\frac{\partial T}{\partial t} = \frac{\partial T}{\partial x} + \frac{\partial T}{\partial y} - \frac{\partial^2 T}{\partial x^2} - \frac{\partial^2 T}{\partial y^2} + f(x, y, t) \tag{15}$$

on the domain $(x, y, t) \in [-1, 1] \times [-1, 1] \times [0, 1]$. To accurately evaluate our numerical PINN results, we manufacture a solution. More specifically, we pick a temperature distribution of our choice, which is a combination of trigonometric cosine functions in space and exponential decay in time.

$$T(x) = e^{-\alpha t} cos\left(\frac{\pi x}{2}\right) cos\left(\frac{\pi y}{2}\right) \tag{16}$$

Once we plug that equation and its derivatives into Equation 15, we obtain the following source term

$$\begin{aligned} f(x, y, t) = e^{-\alpha t} \Bigg(& cos\left(\frac{\pi x}{2}\right) cos\left(\frac{\pi y}{2}\right) \left(\frac{\pi^2}{2} - \alpha\right) \\ & - \frac{\pi}{2} \left(sin\left(\frac{\pi x}{2}\right) cos\left(\frac{\pi y}{2}\right) + cos\left(\frac{\pi x}{2}\right) sin\left(\frac{\pi y}{2}\right) \right) \end{aligned} \tag{17}$$

The family of partial differential equation is parametrized by the exponential rate of decay α and is then given by Equation 15 and Equation 17. We provide the PINN only with partial differential equation along with zero Dirichlet boundary conditions and train it to reconstruct Equation 16 as closely as possible.

4.2 Solution Bundles

4.2.1 Background

The concept of a solution bundle is to learn all the solutions for a variety of values for α at the same time Flamant et al. [2020]. This is achieved by inserting the parameter α alongside the independent variables (x, y, t) into the PINN. We also have to specify a range of values on which the equation family should be learned. Then, we can proceed with training the PINN as usually with the only difference is that the PDE residual corresponding to each input will use the provided value of α in its differential equation residual. A graphical visualization of a solution bundle architecture used to solve the convection-diffusion equation is given in Figure 5.

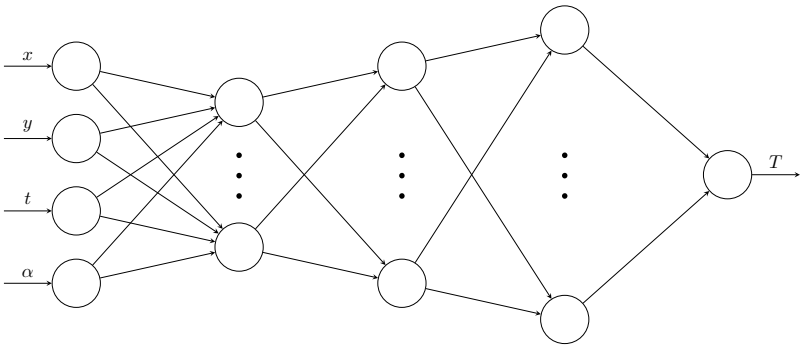


Figure 5: Exemplary PINN as a solution bundle where α is inserted alongside the standard inputs (x, y, t).

	$t = 0.333$	$t = 0.667$	$t = 1.000$
$\alpha = 1.0$	0.38%	0.38%	0.44%
$\alpha = 1.25$	0.19%	0.51%	0.79%
$\alpha = 1.5$	0.24%	0.69%	1.02%

Table 1: Maximum relative residual in percent for three points in time t and three different rates of decay α.

4.2.2 Training Process and Results

For the scope of this report, we train a solution bundle with $\alpha \in [1, 1.5]$. We use a bunch of tricks to accelerate convergence:

First, we weigh the loss function exponentially in time to give higher importance to points that are encountered early on. Second, we train on a "noisy uniform grid" that is resampled ten times over the training process. To do so, we construct a uniform grid in space and time and sample random noise from a Gaussian distribution which is added independently for each coordinate (x, y, t).

We initialize a PINN with four hidden layers of an increasing number of hidden neurons $(10, 20, 30, 40)$. After training the solution bundle for $5,000$ epochs with an initial learning rate of 10^{-3} with the Adam optimizer, we can quantitatively evaluate the PINN predictions for T in Table 1 (visualizations of the residuals can be found in the appendix).

The residuals are computed with respect to the exact solution given in Equation 16 and are in the range of $\mathcal{O}(10^{-3})$. We can observe that errors tend to increase with t which confirms the trends we have observed in the three-body problem. Residuals are also slightly increasing as α increases which is reasonable if you consider the functional form of Equation 16: for larger rates of decay, the PINN needs to approximate sharper gradients in the challenging temperature field. We have empirically observed that these residuals can be further decreased with more complex network architecture, e.g. a deeper (more layers) or wider (more hidden neurons per layer), or more extensive training.

We can therefore conclude that the solution bundle has converged and is now able to accurately provide the solution for the differential equation in Equation 15 together with Equation 17. Notice that inference is cheap after the training process: For any given α, the evaluation of the spatio-

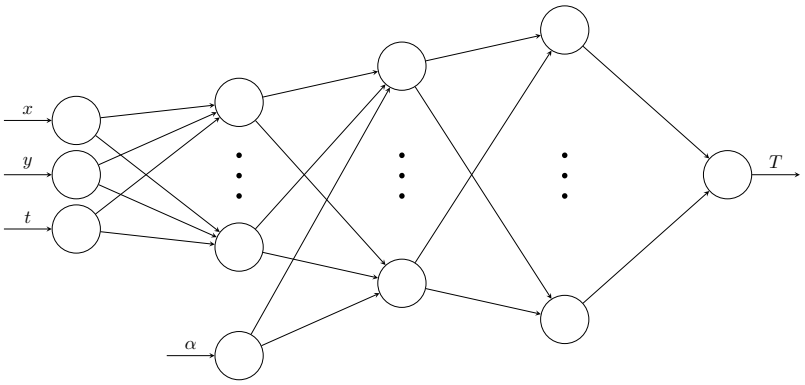


Figure 6: Exemplary PINN as a solution bundle where α is inserted at the second hidden layer.

temporal solution is obtained by a single forward pass of the PINN. This is computationally much more efficient than a conventional numerical method that requires resolving the PDE instance when a new value of α is provided. The solution bundle has in contrast learned a functional dependence $T(\alpha)$ and is therefore able to provide a solution without any further training.

4.2.3 Parameter Studies on the Network Architecture

We have seen in the section before that a PINN can learn a PDE family. However, we iteratively trained the PINN on distinct but specific instances by inputting the parameter α in the input layer.

Here, we want to propose a setup that splits the PINN into two parts: a general part that extracts common patterns of the PDE family and a second part specific to α. We construct such a PINN by inserting α at some hidden layer instead of the input layer. One can equivalently imagine that the parameter is appended to some hidden layer nodes and the concatenation of both serves as an input to the next layer. The intuition behind this approach is that the PINN "does not know" the value of the parameter until it is inserted and is therefore forced to extract general patterns depending on (x, y, t). A possible such architecture is given in Figure 6.

We benchmark the results of inserting α at each of the four hidden layers to the baseline of treating α as standard input. In Figure 7 you can observe the loss curves of all five scenarios during training.

We observed that the results are very sensitive to random initialization and therefore averaged the relative solution errors in the L_2 norm over 5 runs. We can observe that inserting at the final layer results in the worst performance. This is reasonable because the dependence of α on the exact solution given by Equation 16 is nonlinear. The insertion at some hidden layers is relatively similar, but the lowest error is achieved when inserting the parameter alongside the standard inputs.

Therefore, the PINN did not learn a general and specific part in the way we desired. We can conclude that a solution bundle should be trained in such a way that a differential equation parameter is treated as standard input.

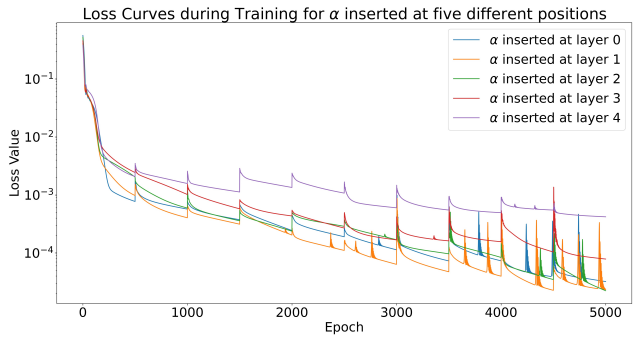


Figure 7: Five mean squared error losses during training time. One function corresponds to a different layer at which α is inserted.

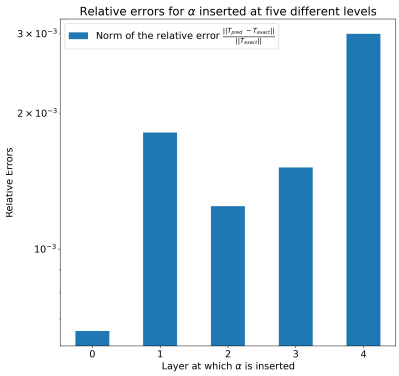


Figure 8: Relative solution errors of T measured in the L_2 norm.

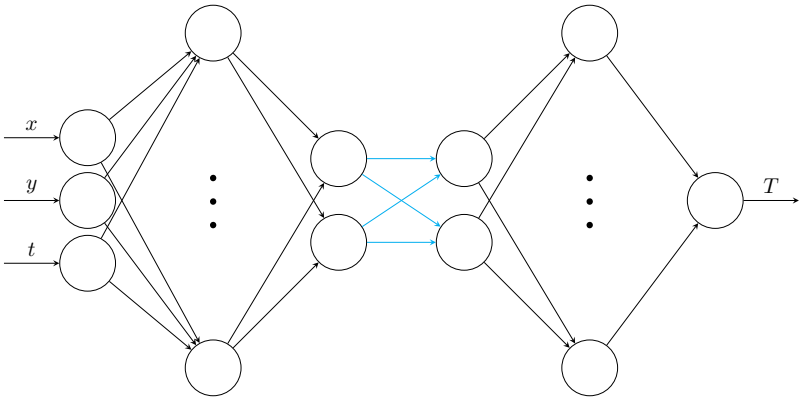


Figure 9: Exemplary PINN as a bottleneck freeze where the bottleneck layers are frozen during retraining.

4.3 Bottleneck Freeze

4.3.1 Background

We experimented with a different approach for ensemble learning that is closely related to transfer learning. In this scenario, we train a PINN on a single PDE instance with a fixed value α_0. After training, we store numerical values for the network's parameters (i.e. weights and biases) along with its architecture. This trained network will then serve as an initialization for a PINN trained on a PDE with $\alpha_1 \neq \alpha_0$ but $\alpha_1 \approx \alpha_0$, i.e. it will be trained on a "similar" PDE instance. During training, we will freeze the parameters corresponding to the bottleneck layers, so their numerical values will not change. The intuition behind that approach is that we expect the PINN to extract some structure inherent to the PDE that is generalizable across the PDE family. The subsequently trained network should then have superior performance.

Figure 9 shows a possible PINN with two bottleneck layers, i.e. layers with very few hidden nodes (in this case two) that compress information. For the scope of this report, we use a network with five hidden layers of widths $(20, 20, 2, 2, 20, 20)$.

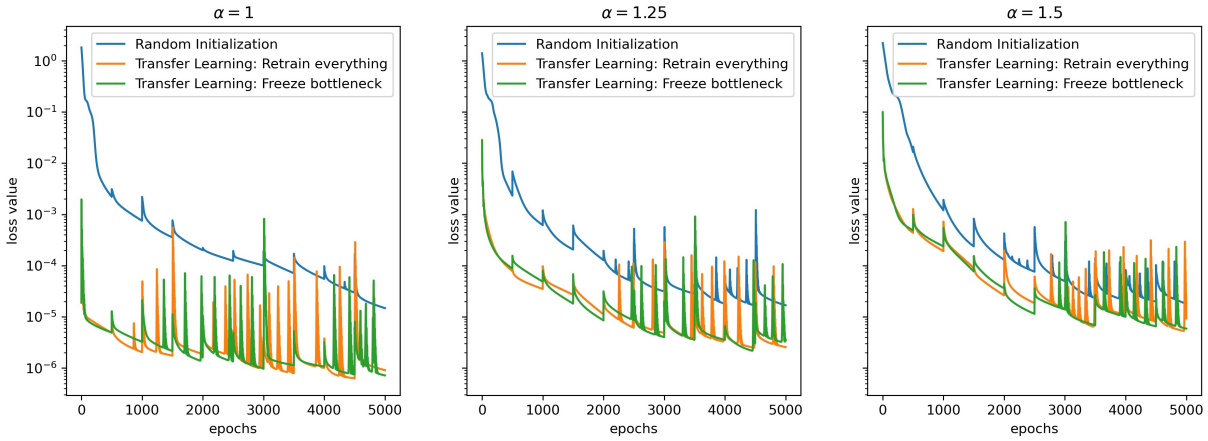


Figure 10: Training Process for three values of $\alpha_1 = 1$ (left), 1.25 (middle), 1.5 (right) for three different training procedures.

4.3.2 Training Process and Results

The training process follows several of the principles employed for the solution bundle (see subsubsection 4.2.2 including Adam optimizer, a noisy uniform grid, and an exponentially decaying loss function.

First, we train a PINN on the convection diffusion equation given by Equation 15 on $\alpha_0 = 1$ and store the network's parameters. We then train new PINNs on three different values for $\alpha_1 = 1, 1.25, 1.5$ and employ three training procedures on each of them:

1. Random initialization: the saved model is disregarded and weights and biases are initialized at random. This model serves as a baseline for comparing accuracy and efficiency.

2. Transfer learning without freeze: we initialize the new PINN with the saved parameters and retrain the entire network starting from the final model trained on α_0

3. Transfer learning with freeze: equivalent to approach 2 with the difference that the network's parameters in the bottleneck layers are frozen. That is, these four weights and two biases will not be updated during training, so their final value will be equivalent to the final value in the model trained on α_0

The convergence during training can be observed in Figure 10: we can see that approach 1 is consistently outperformed by both transfer learning approaches. The latter is superior in terms of initial accuracy, speed of convergence (sharper decline of the mean squared error loss in the beginning) as well as final accuracy. The effect of freezing the parameters in the bottleneck is relatively minor but leads to a slightly lower training loss after $5,000$ epochs.

4.4 Comparison

Two ensemble learning approaches have been described in this chapter: solution bundles and the bottleneck freeze. The following table aims to give a concise summary of their approaches:

Strategy	Solution Bundles	Bottleneck Freeze
Inputs	(x, y, t, α)	(x, y, t)
Training	expensive (learn family)	cheap (learn specific instance)
Evaluation	very cheap (forward pass)	cheap (requires retraining)

Table 2: Comparison of solution bundles and bottleneck freeze across different evaluation metrics.

Note that the adjectives very cheap, cheap, and expensive should be interpreted relative to each other and not in absolute terms.

5 Pierce Hall

Previously in this paper, the problems where PINNs have been used were fairly simple problems, with simple boundary and initial conditions. However, we were curious how this method would adapt when we apply it to more complex problems with more complex boundary conditions such as for the study of fluid dynamics in racing hydrofoil sailboats. Hence, we decided to try to bring the reality of solving such complex systems with PINNs closer. However, the setting of racing sailboats seemed too complicated for the scope of this project and we decided to go to a simpler problem, a simplified version of the Pierce Hall problem seen in Homework 4 of AM205.

5.1 Problem Setup

The simplified instance of the Pierce Hall problem that we are going to solve using PINNs consists of the study of the pressure field inside a room. The pressure field $p(x, y, t)$ satisfies the wave equation

$$\frac{\partial^2 p}{\partial t^2} - c^2 \nabla^2 p = 0 \tag{18}$$

where $\nabla^2 = \nabla \cdot \nabla$ denotes the Laplace operator. Originally, the Pierce Hall problem simulated this pressure field in one of the floors of the Pierce Hall building at Harvard University. However, we will limit ourselves to simulate this field in a squared room of dimension 5 by 5, surrounded by walls and discretized using 5 by 5 points in a grid (with a total of 25 points). Furthermore, we will solve this problem for the time between 0 and 1 seconds using 100 time steps. Finally, there is a speaker in the center of the room, at point $(x = 2, y = 2)$, that produces a pressure wave that will travel through the room.

As we mentioned before, the goal of trying to solve such a problem was to verify whether these PINNs work for problems with many more boundary conditions. We proceed by stating the boundary conditions in this problem. First, we have two types of initial conditions. The first is that the field at $t = 0$ will be 0 (Equation 19). Second, the derivative in time at $t = 0$ has again to be 0 (Equation 20). These two conditions have to be satisfied for all points inside the room.

$$p(x, y, 0) = 0 \tag{19}$$

$$\frac{\partial p(x, y, 0)}{\partial t} = 0 \tag{20}$$

Furthermore, this problem contains also boundary conditions. The first is the boundary condition at the speaker in the center of the room. This pressure is dictated by Equation 21. This pressure wave is different than the one present in the original problem which was $p_0 \sin(wt)$. The reason is that the derivative at $t = 0$ at this point using the original pressure wave would be non-zero which conflicts with the initial condition (Equation 20). This is not a problem using traditional numerical methods, however, it was a problem training the PINN and we had to change it to a pressure wave that would have a derivative equal to zero at $t = 0$. Finally, we have Neumann boundary conditions at the walls so that the derivatives at the wall satisfy the boundary condition (Equation 22), where $\mathbf{n}$ is a unit vector normal to the wall.

$$p_{center} = p_0 \cdot (1 - \cos(wt)) \tag{21}$$

$$\mathbf{n} \cdot \nabla p = 0 \tag{22}$$

Finally, the parameters of the equation were chosen to bound the pressure field close to [-1,1]. The reason is that it will be easier for the PINN to output values in this interval than if it has to output values in a much larger interval. Hence, the values of the parameters are the following: $c = 3.4$, the angular frequency is $w = 5\ s^{-1}$, the pressure amplitude is $p_0 = 1\ Pa$.

5.2 PINN Configuration

5.2.1 Network Architecture

The architecture of the PINN is a multi-layer perceptron that has 3 nodes in the input layer, corresponding to x, y, and t. Continuing with 3 fully-connected hidden layers each with 20 neurons and finishing with a single output node corresponding to the pressure field at point $x, y, t : p(x, y, t)$. A visualization of this neural network can be found in the appendix.

5.2.2 Training Dataset

The training dataset was composed of 5 different subsets. These correspond to the different terms that we want to minimize: the residuals for the initial conditions, boundary conditions, and PDE.

1. **Initial points**: 5 by 5 points evenly spaced in the grid with with $t = 0$ for which we want the pressure field and the partial derivative with respect to time to be equal to 0 ($N_{x,y} = 25$).

2. **PDE** 5 by 5 points evenly spaced in the grid with 100 evenly distributed time points between 0 and 1, for which the PDE equation has to be satisfied ($N_{x,y,t} = 2500$).

3. **Wall boundary conditions** 5 points evenly distributed on each one of the walls for which the Neumann boundary condition has to be satisfied for 100 evenly distributed time points between 0 and 1 ($N_{x_{wall},y_{wall},t} = 2000$).

4. **Speaker boundary conditions** 1 point in the center of the grid corresponding to the speaker, repeated for 100 evenly distributed time points between 0 and 1, where the boundary condition regarding the pressure field generated by the speaker has to be satisfied ($N_t = 100$).

The dataset decomposition presented above is an indicator of the different error terms that we try to minimize to learn the desired solution. As explained earlier in the introduction we will use each point of the dataset and compute the loss term with respect to the expected outcome. Hence, the final loss function that the PINN will have to minimize will be the following: The architecture of the PINN is a multi-layer perceptron that has 3 nodes in the input layer, corresponding to x, y, and t. Continuing with 3 fully-connected hidden layers each with 20 neurons and finishing with a single output node corresponding to the pressure field at point $x, y, t : p(x, y, t)$. A visualization of this neural network can be found in the appendix.

$$\mathcal{L}_{total} = \mathcal{L}_{initial} + \mathcal{L}_{initial\_derivatives} + \mathcal{L}_{PDE} + \mathcal{L}_{wall} + \mathcal{L}_{speaker} \tag{23}$$

Where each of the terms are the following:

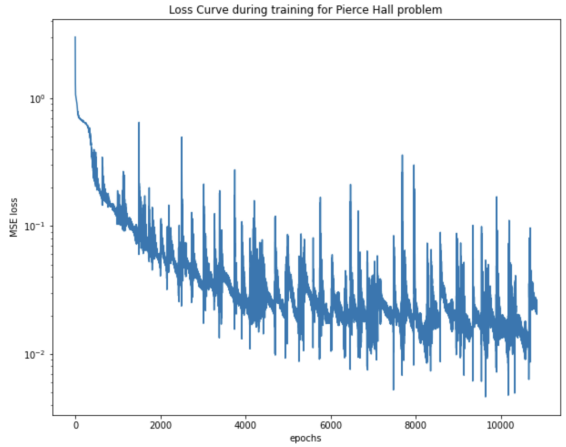


Figure 11: Overall loss curve when training the PINN for the Pierce Hall problem.

Figure 12: Decomposed loss curve when training the PINN for the Pierce Hall problem.

$$
\begin{aligned}
\mathcal{L}_{initial} &= \frac{1}{N_{x,y}} \sum_{x,y} p(x,y,t_0)^2 \\
\mathcal{L}_{initial\_derivatives} &= \frac{1}{N_{x,y}} \sum_{x,y} \left(\frac{\partial p(x,y,t_0)}{\partial t} \right)^2 \\
\mathcal{L}_{PDE} &= \frac{1}{N_{x,y,t}} \sum_{x,y,t} \left(\frac{\partial^2 p}{\partial^2} - c^2 \nabla^2 p \right)^2 \\
\mathcal{L}_{wall} &= \frac{1}{N_{x_{wall},y_{wall},t}} \sum_{x_{wall},y_{wall},t} (\mathbf{n} \nabla p)^2 \\
\mathcal{L}_{speaker} &= \frac{1}{N_t} \sum_{t} (p(x_{speaker}, y_{speaker}, t) - p_0(1 - \cos(wt)))^2
\end{aligned}
\tag{24}
$$

5.2.3 Model Training

The training was performed using Adam optimizer with an initial learning rate of 0.01 and run for around 11,000 epochs. The loss function used was the mean-squared error. It resulted in being hard to learn such a model. We had to introduce several tricks during training to speed up the training and achieve lower losses. The first trick was to add an exponentially decaying learning rate through the epochs, with a discount factor of 0.99. Then, we also added gradient clipping to a maximal norm of 0.1. The reason why we added the second one was because we were seeing many peaks in the loss curve during previous training, and using gradient clipping we managed to reduce them. The total training time was around 40 minutes.

In Figure 11 we can see the overall loss curve during training, and in Figure 12 we can see the decomposed loss for each one of the loss terms. We observe that even after gradient clipping we continue to see these high peaks during the loss, this means there is still room for improving the learning but it also is an indicator of the complexity of learning such models. Furthermore, we see that the loss decreases which means that it learned. We also see that it starts to converge but

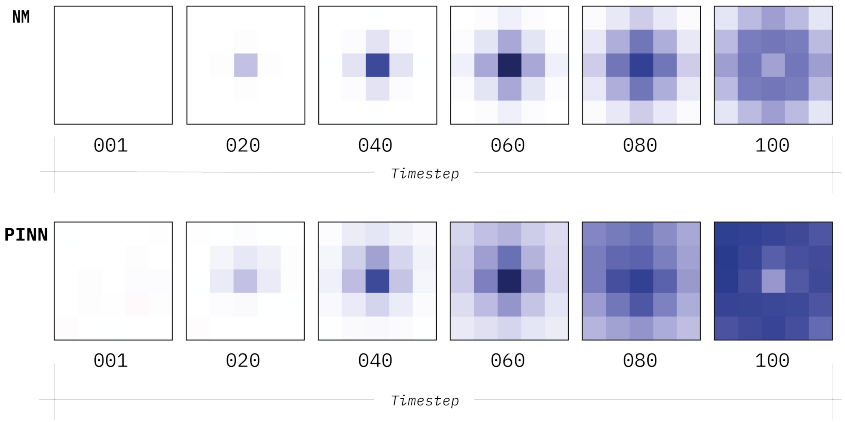


Figure 13: Comparison between numerical method and PINN solutions.

it does not seem like it completely converged meaning that maybe with more training time and computing resources we could achieve a lower loss.

5.3 Numerical Methods

We will compare the PINN solution to a solution obtained through a traditional numerical method solution. We chose the same method as the one used to solve the original problem in the homework. This consists in using the following two-dimensional discretization:

$$\frac{P_{j,k}^{n+1} - 2P_{j,k}^{n-1} + P_{j,k}^{n-1}}{\Delta t^2} - c^2 \frac{P_{j+1,k}^{n} + P_{j,k+1}^{n} - 4P_{j,k}^{n} + P_{j-1,k}^{n} + P_{j,k-1}^{n}}{h^2} = 0 \tag{25}$$

where $P_{j,k}^{n}$ is the numerical approximation of $p(i, j, n\Delta t)$ with time step Δt. Furthermore, we imposed the initial conditions (Equation 19, Equation 20, Equation 21) by forcing them. The last boundary condition in Equation 22 was satisfied by using the ghost node approach: when considering a point (j, k) that references an orthogonal neighbor (j', k') that is in a wall, then we set $P_{j',k'}^{n} = P_{j,k}^{n}$.

5.4 Results

5.4.1 Comparison of PINNs and Traditional Numerical Method

In Figure 13 we can see the frames representing the pressure field around the room at certain timesteps. We observe that until step 60, we have a similar shape for the pressure field, even though the PINN outputs higher values. However, at steps 80 and 100 we see that the shapes of the pressure field are very off. In Figure 14, we can see this difference between two methods in numerical values and we observe that in general, the error in each point of the grid increases exponentially with time. This is a recurring problem when training PINNs that we have already seen before. Furthermore, we observe that the speaker point, in the center of the grid, maintains its correct value, which can also be numerically seen in Figure 15.

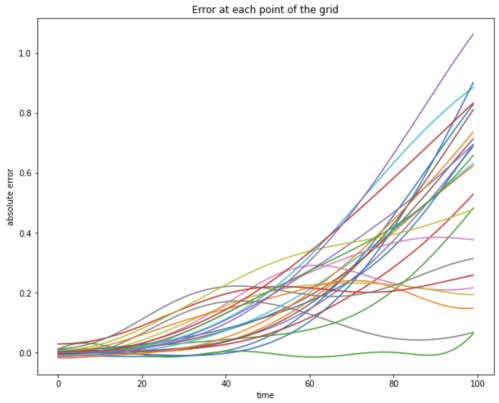


Figure 14: Difference between numerical method and PINN solutions for all points in the grid.

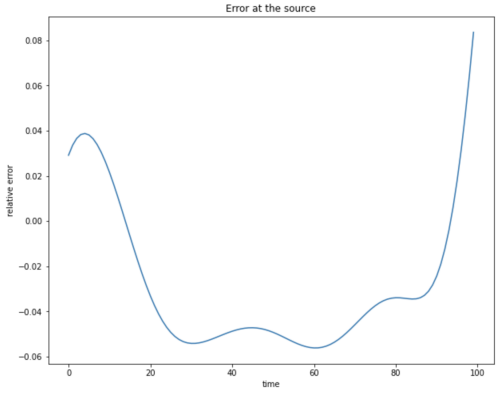


Figure 15: Difference between numerical method and PINN solutions for the speaker point.

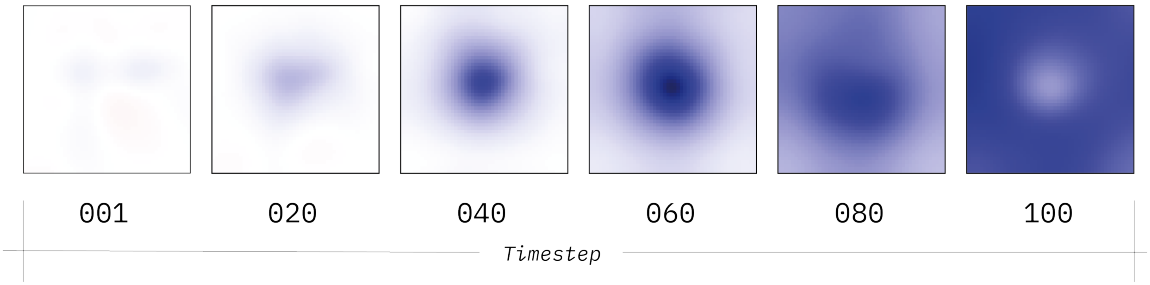


Figure 16: Increase in resolution by 100, and output given by PINN without any extra training.

5.4.2 Increasing Pixel Resolution

Even though, as we have seen before, the PINN did not yet achieve the same accuracy as the numerical method, we can already exploit some of the benefits of this method. For example, since the PINN gives us a continuous function, we can sample from it at any given point, and not just the points used during training, something that is not possible with the traditional numerical method we used. Hence, we decided to increase the resolution by a factor of 10 for both x and y. In Figure 16, we can see some timesteps with this increased pixel resolution. We see the behavior is very reasonable and this was achieved with 0 additional training time.

5.4.3 Increasing Time Resolution

In this section, we decided to do the same as in the previous section. However, we increased the time resolution by 2. We did not add the frames since this increase in time resolution cannot be appreciated through them. However, we plotted the differences in Figure 17 and Figure 18, we see these two figures show continuous functions and almost identical to Figure 14 and Figure 15 which indicates that the trained PINN properly interpolates the solution in time. As before, these results were obtained with no additional training cost.

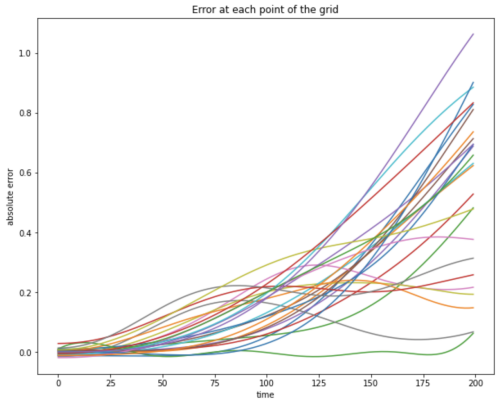


Figure 17: Difference between numerical method and PINN solutions for all points in the grid and doubling time resolution.

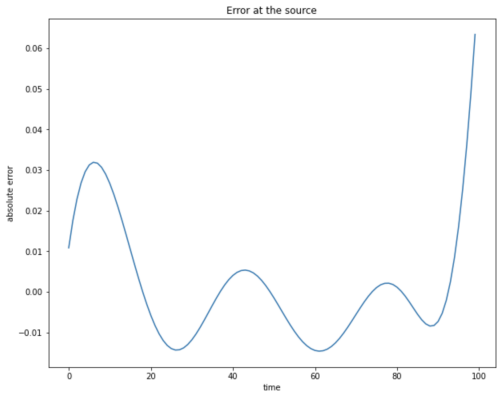


Figure 18: Difference between numerical method and PINN solutions for the speaker point and doubling time resolution.

5.4.4 Extending Solution in Time

In the two sections above, we increased the resolution of x, y, and t, however, this increase was made inside the interval of values used during training. We wanted to analyze what happens when we leave the training interval. We tried this by sampling the PINN solution for 1 extra second after the times used during training. In Figure 19 we see the solution for some timesteps between 0 and 2 seconds. We observe that after timestep 100, it does not make much sense anymore. We can extract the same conclusion by looking at Figure 20, where after the timestep 100, corresponding to 1 second, the errors are all over the place. Hence, we can conclude as expected, that this PINN method did not generalize to any timestep, i.e. it cannot extrapolate in time. And hence, the solution is only relatively valid inside the training interval.

5.4.5 Improvements

Finally, some techniques have been shown to help learn better solutions when using PINNs. In our project, we were not able to try them all. However, we wanted to present one of the techniques we tried that brought improvements. This consisted in choosing randomly selected timesteps inside the training interval instead of having them fixed. These timesteps were re-selected every 10 epochs during training, which means that the training dataset was changing every 10 epochs.

In Figure 21, we can see the comparison between the version using the constant timesteps and the improved method using re-sampling every 10 timesteps. We see the improvement is indeed significant for such a small change. This gives us hope in that if we add extra techniques we could greatly improve the performance and hence there is also hope in PINNs will someday work for more complex problems.

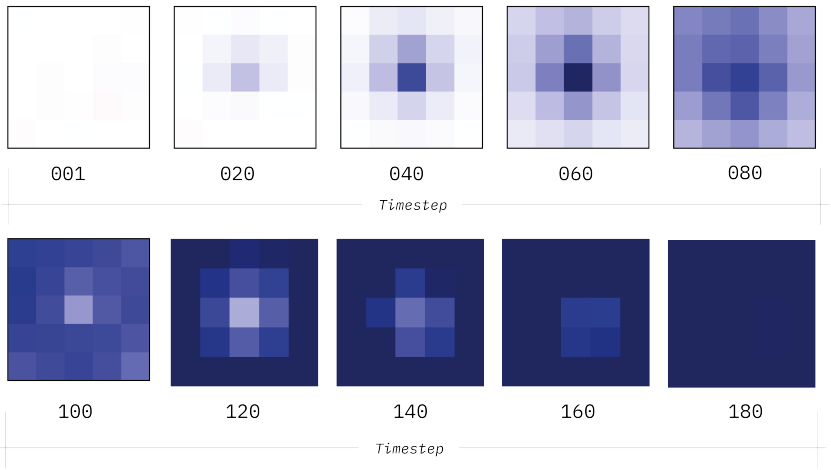


Figure 19: Extend simulation in time by a factor of 2, and output given by PINN without any extra training.

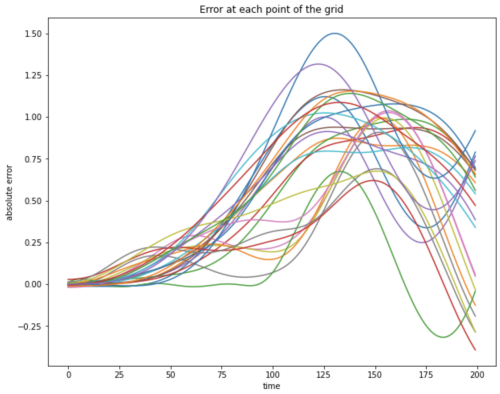


Figure 20: Difference between numerical method and PINN solutions for all points in the grid and extending time sampled by 2.

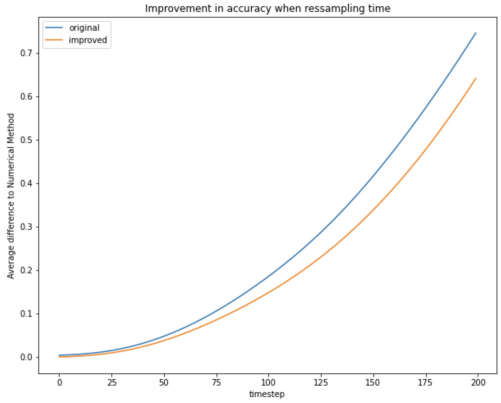


Figure 21: Comparison between original and improved training technique and improvement seen in the average difference between numerical method and PINN solutions over all points for the grid.

6 Conclusion

We empirically investigated how physics-informed neural networks can be used to solve differential equations. We studied three problems and applied several modeling tricks to accelerate convergence including the reparametrization trick, weighed loss function, and curriculum learning.

Solving differential equations with PINNs can bring great benefits. The solution framework is generalizable to any differential equation and it gives analytic closed-form continuous solutions which can be evaluated at arbitrary accuracy without the need for explicit interpolation. Furthermore, the solution is infinitely differentiable and partial derivatives can easily be obtained with automatic differentiation. Finally, if we train a PINN for solution bundles then we can have a single function that will be the solution of a family of differential equations and can thus provide a solution for any parameter at test time as a forward pass.

However, we have also seen some downsides to this method. To achieve a given level of accuracy, PINNs are less computationally efficient than traditional numerical methods. As conventional numerical methods including finite differences, finite elements, and finite volumes have been developed for several decades, error bounds and proofs of convergence are readily available for many of them. PINNs are a comparably recent invention and research on their theoretical properties is still in its infancy.

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

Tim De Ryck and Siddhartha Mishra. Error analysis for physics informed neural networks (pinns) approximating kolmogorov pdes. *arXiv preprint arXiv:2106.14473*, 2021.

Cedric Flamant, Pavlos Protopapas, and David Sondak. Solving differential equations using neural network solution bundles. *arXiv preprint arXiv:2006.14372*, 2020.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Remco van der Meer, Cornelis W Oosterlee, and Anastasia Borovykh. Optimally weighted loss functions for solving pdes with neural networks. *Journal of Computational and Applied Mathematics*, page 113887, 2021.

A Appendix

A.1 Residuals for the Convection-Diffusion Equation

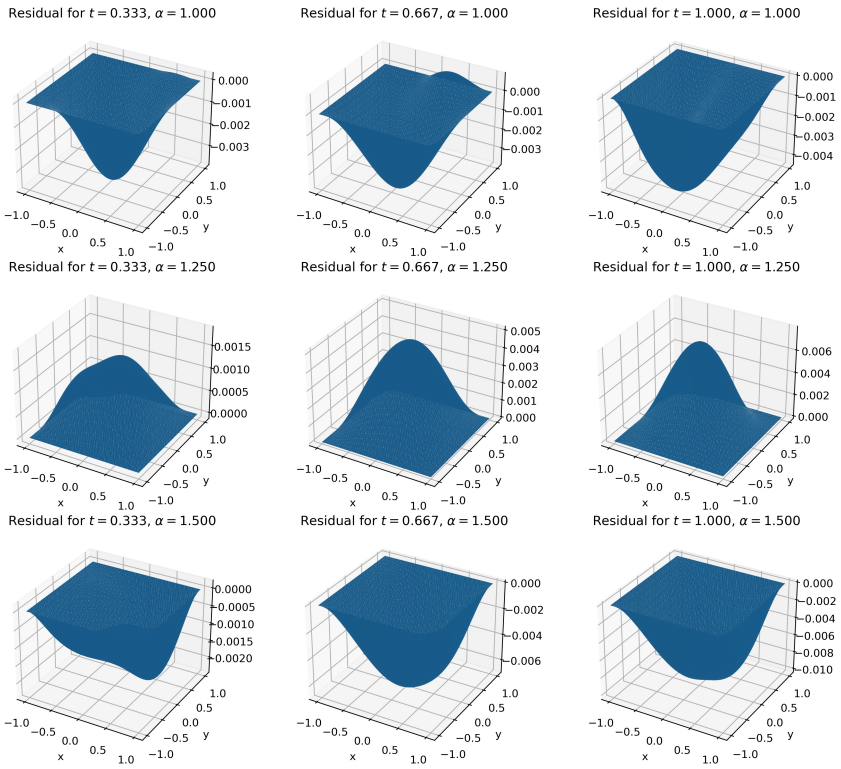


Figure 22: Residual plots in space for $t = 0.333$ (left), $t = 0.667$ (middle) and $t = 1.000$ (right), $\alpha = 1.0$ (top row), $\alpha = 1.25$ (middle), $\alpha = 1.5$ (bottom row).

A.2 Neural Network model architecture for the Pierce Hall problem

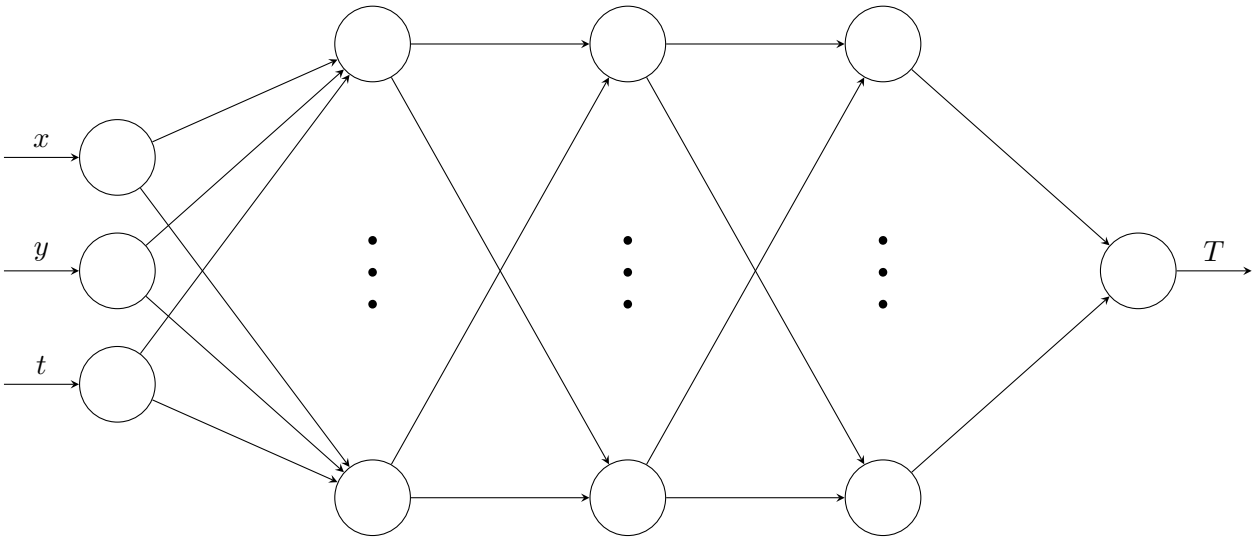


Figure 23: Neural Network model architecture used for the Pierce Hall problem, it contains 4 fully connected layers, with the first one having 3 neurons, the 3 following have 20 hidden neurons and the last layer has a single neuron.