Physics Informed Neural Networks

Finding Fluid Velocity with temperature and concentration data

John Chang Su
josu7836@uni.sydney.edu.au

University of Sydney
10th Feb 2023

Contents

1. Introduction

Deep learning and neural networks has seen explosive growth in recent years, thanks to the significant advancement in computer hardware and the highly parallelisable backpropagation algorithms. Furthermore, Neural networks are known to be universal approximators meaning with enough data and network size, any function can be arbitarily approximated to any accuracy.

Numerical simulation methods such as computational fluid mechanics (CFD), finite element method (FEA) and finite differences on the contrary have a much longer history and success in providing highly accurate predictions and results.

However, numerical methods such as CFD and FEA tend to be very slow and often require significant model simplification and/or runtimes that can last for weeks. However, in recent years, so called Physics informed Neural Networks (PINNs) have been implemented to solve PDEs and ODEs where it is infeasible to run numerical methods such as inverse modelling or fractional PDEs.

This summer research aims to examine the use of PINNs for inverse modelling to determine unknown quantities such as velocity of a flow field when only provided with partial information such as temperature. In particular, the project focuses on recovering the velocity field when only given the temperature field for temperature driven fluid flow.

2. Physics Informed Networks

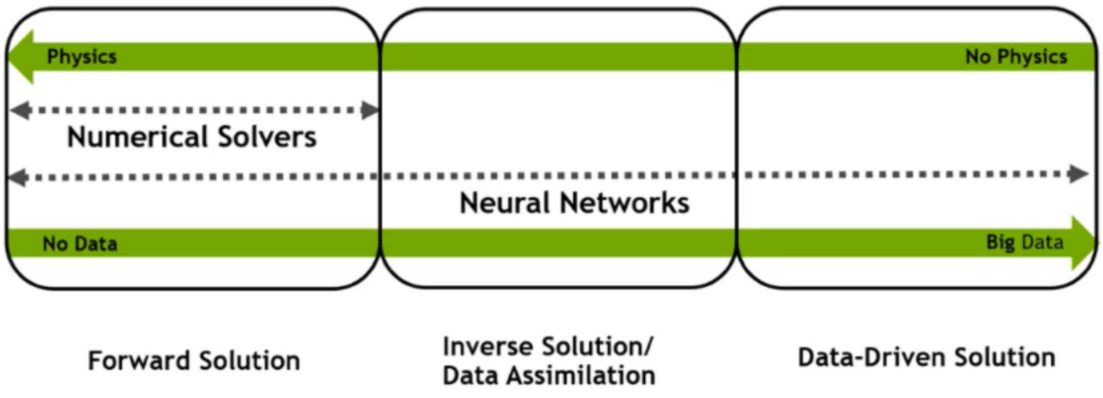


Figure 1. Diagram of different modelling approaches [1]

Computational modelling can roughly be placed in three categories: forward,inverse and data-driven modelling. In forward modeling, one uses the governing equations to progress the simulation forward. Typically one only knows the boundary and initial conditions of the model. CFD and FEA are two well established methods for solving these types of models with many commercial codes such as Ansys and Abaqus being widely adopted in industry. However, these methods tend to struggle with inverse modelling approaches.

Data driven modelling is almost the opposite where one has data but little to no information on the underlying equations. Machine learning and neural networks have seen a significant success in this category providing state of the art results for problems such as image classification and natural language processing.

Inverse modeling, is leveraging both governing equations and data to determine unknown states. This could range from predicting values in areas wheredata is not available for example satellite imagery being blocked by clouds, or determine unknown quantities such as velocity when only given temperature of a fluid. PINNs excel in both the forward and inverse modelling.

2.1. How a PINN works

With the universal approximation theorem, neural networks have the power to approximate any function [2]. As a result, researchers have turned to using networks to solve equations where traditional methods are infeasible such as high dimensional PDEs or even fractional PDEs [3].

However, training on data alone does not guarantee a network will learn the underlying physics and correctly generalise to regions where data is sparse or unknown.To ensure networks are learning the correct physics, the network is 'informed' by including the underlying equations during network training. In general, a PDE can be expressed as [4]:

$$u_t + \mathcal{N}[u] = 0, x \in \Omega, t \in [0, T] \tag{1}$$

where $\mathcal{N}[\cdot]$ is the non linear operator over the output function u. To inform our network of the PDE that governs the data, one treats the equation as a residual:

$$r = u_t + \mathcal{N}[u] \tag{2}$$

which is added to the loss function and minimised at each point (x,t) via mean square error:

$$\mathcal{L}_{residual} = \frac{1}{N}\sum_{i}^{N}(|r(x_i, t_i)|^2 \tag{3}$$

Combined with appropriate boundary, initial or data which are also minimised via mean square error, the total loss is:

$$Loss = \mathcal{L}_{residual} + \mathcal{L}_{data} + \mathcal{L}_{BC} + \mathcal{L}_{IC} \tag{4}$$

The types and formulation of PINNs is varse and can vary considerably. However, this report primarily focus on the above formulation where point cloud data e.g. (x, t) is mapped to it's corresponding output $u(x, t)$.

For forward modeling problems the 'training' data is a mix of boundary/initial conditions or known data points such as from a sensor and 'collocation points'. Collocation points are data points located inside the domain where there is no data but must adhere to the equations. An example of collocation points are time points in the future. Collocation points ensure that the system equations are being followed across the domain.

Inverse modelling is determining unknown quantities or underlying equation from provided data. This has tremendous use in real-life applications where often some quantities are well known such as temperature or concentration but desired quantities such as velocity field or fluid properties such as viscosity are unknown. Raissi [5] showed the power of neural networks for this problem being able to to discover the underlying velocity by only providing the governing equations and concentration data of the flow.

PINNs have also begun to see usage outside of research interests. Nvidia Modulus is a recent python library dedicated to solving PINNs [1]. Modulus has been successfully used by companies such as Siemans in the optimisations of wind turbine placement [6]

2.2. Advantages of PINNs

1. Derivative can be obtained via automatic differentiation
2. PINNs are meshless method allowing querying at any point inside a domain whereas typical CFD or FEA methods uses discretization
3. Highly parralisable and can scale well with GPUs. standard physics solvers are sequential in nature are not as parralisable
4. Requires minimal pre-processing, as typically PINNs take point cloud data as input
5. The method for foward modelling and inverse modelling with PINN is identical.

2.3. Disadvantages of PINNs

1. PINNs have no guarantee of convergence
2. Network architecture and size is problem dependent
3. For large spatial or large time horizons, PINNs can 'forget' previous information.

3. Forward Modeling Example - 1D Spring Equation

Consider the following linear ODE for a mass-spring system with damping:

$$\frac{d^2u}{dt^2} + \frac{du}{dt} + 6.5t = 0, \quad \text{for } t \in (0, 2\pi) \tag{5}$$

$$u(0) = 0, \quad \frac{du}{dt}(0) = 2.5 \tag{6}$$

which has an analytic solution:

$$u(t) = e^{-0.5t} (\sin(2.5t)) \tag{7}$$

However suppose one is unaware of the analytic solution and one only has data points for $0 < t < 1$ and equation 5 and one is interested in the systems behaviour for $1 < t < 2\pi$. Then using a PINN the problem is formulated as an optimisation problem:

$$Loss = \mathcal{L}_{residual} + \mathcal{L}_{Data} \tag{8}$$

$$\mathcal{L}_{residual} = \frac{1}{N+M} \sum_{i}^{N+M} (\frac{d^2u_\theta}{dt^2}(t_i) + \frac{du_\theta}{dt}(t_i) + u_\theta(t_i))^2, \quad \mathcal{L}_{Data} = \frac{1}{N} \sum_{j}^{N} (u_{net}(t_j) - u(t_j))^2 \tag{9}$$

Where $u_net(t)$ is the current network prediction for input t, $u(t)$ is the true data of u, N it the number of data points and M is the number of 'collocation points'. The Collocation points are 20 uniformly space points between $1 < t < \pi$. The residual equation 'informs' the network that these collocation points must behave according to the the equation.

3.1. Network and training

The PINN is a simple single hidden layer with 50 neurons with the $\sin$ function as the activation function. Adam optimizer is used with an learning rate of 1e-3 and run for 10,000 epochs. This simple example can be quickly trained with a CPU.

3.2. Results

Figure 2 shows the comparison between a PINN network prediction vs a regular network trained only on the provided data. At 9800 epochs, the PINN solution matches the analytic solution (dashed green) very closely for $t > 1$ even though only data points were provided for $t < 1$. On the other hand, a standard neural network trained to fit only the data points fails to generalise correctly for time points greater than 1

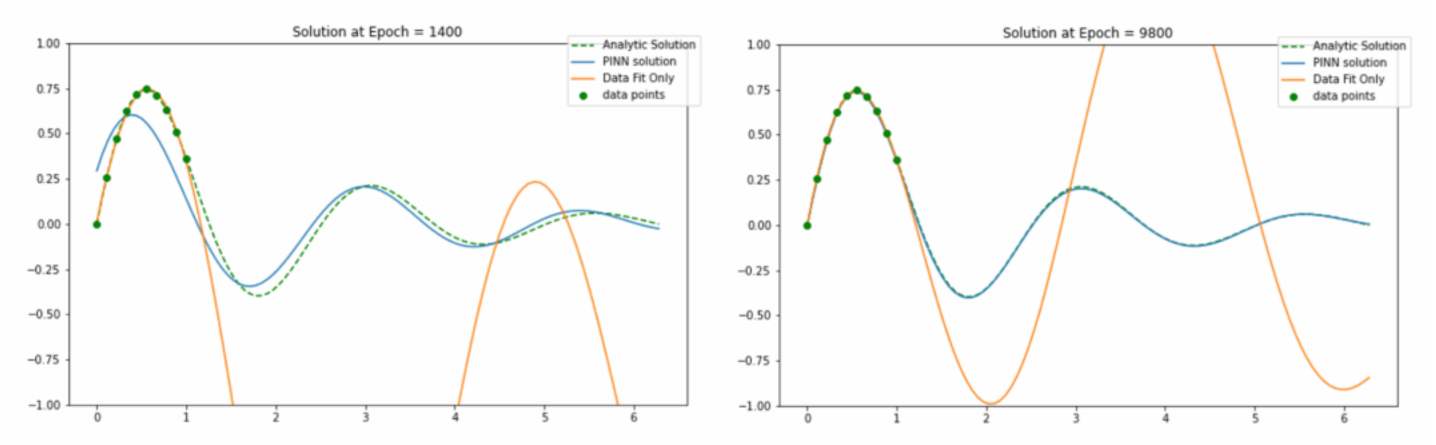


Figure 2. Accuracy of PINN vs model fit to only the green data points

4. Inverse Modelling

4.1. Scalar-Advection Equation Coupled With Navier Stokes

For inverse modelling, the 2D flow around a cylinder is recreated from Raissi [7][4]. Here one is given the **only** concentration data being advected by an incompressible fluid and wishes to uncover the unknown or 'hidden' flow velocity and pressure variables. The domain is a 2D channel of width 10 and height 5 with a cylinder of diameter $D = 1$ located at the origin. In this problem the concentration, which can represent dye or temperature, is assumed to not affect the underlying flow.

The equations that govern this flow are defined as:

$$\frac{\partial u}{\partial t} + \nabla u \cdot \vec{u} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\nabla^2 u \tag{10}$$

$$\frac{\partial v}{\partial t} + \nabla v \cdot \vec{u} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\nabla^2 v \tag{11}$$

$$\nabla \cdot \vec{u} = 0 \tag{12}$$

$$\frac{\partial c}{\partial t} + \nabla c \cdot \vec{u} = +\frac{1}{Pec}\nabla^2 c \tag{13}$$

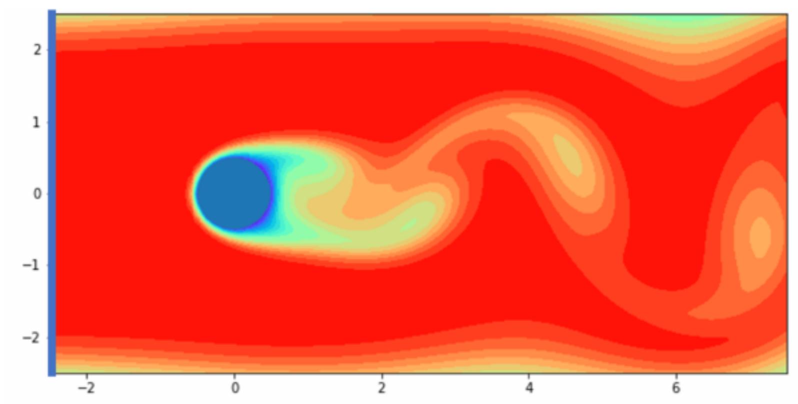


Figure 3. Visualisation of the flow problem. Fluid enters from the left boundary (blue line) and flows past the cylinder. Concentration can represent smoke or dye being advected by the fluid

Where $\vec{u} = [u, v]^T$ is the velocity vector, t is time, p is pressure, c is the concentration (which varies between 0 and 1), Re is the Reynolds number, and Pec is the Peclet number. For this problem, $Re = Pec = 100$. Equations 10 and 11 are the non-dimensionalised Navier-Stokes equations that govern fluid flow, 12 is the continuity equation for incompressible flow (density is constant across the flow) and 13 is the scalar advection equation.

These equations can be reformulated as residuals where the goal of the neural network is to minimise these residuals to zero:

$$\begin{aligned}
e_1 &= \frac{\partial u}{\partial t} + \nabla u \cdot \vec{u} + \frac{\partial p}{\partial x} - \frac{1}{Re}\nabla^2 u \\
e_2 &= \frac{\partial v}{\partial t} + \nabla v \cdot \vec{u} + \frac{\partial p}{\partial y} - \frac{1}{Re}\nabla^2 v \\
e_3 &= \nabla \cdot \vec{u} = 0 \\
e_4 &= \frac{\partial c}{\partial t} + \nabla c \cdot \vec{u} - \frac{1}{Pec}\nabla^2 c
\end{aligned} \tag{14}$$

The loss can then be formulated as:

$$\begin{aligned}
Loss &= \sum_{i}^{4} \mathcal{L}_{residual_i} + \mathcal{L}_{data} \\
Loss &= \sum_{i}^{4} \frac{1}{N} \sum_{j}^{N} e_i(x_j, t_j)^2 + \frac{1}{N} \sum_{j}^{N} c_{net}(x_j, t_j) - c(x_j, t_j))^2
\end{aligned} \tag{15}$$

4.2. Network and Training

For this example, Raissi suggests 50 neurons per output variable per hidden layer ($50 \times 4 = 200$). As such, a fully connected model is used with 10 hidden layers each with 200 neurons per layer. Adam is the optimizer used with a learning rate of 1e-3 and a mini batch size of 10,000 is used. Finally, 2 million random data points are used from the 9 million data points available.

The L2 relative error is used as a way to measure the relative average error of the PINN solution. The L2 error is defined as:

$$L_2\text{Error} = \sqrt{\frac{||\hat{x} - x||^2}{||x||^2}} \tag{16}$$

where $\hat{x}$ is the outputs of the neural network across all inputs expressed as a single vector, x is the corresponding true values and $||\cdot||$ denotes the magnitude of the vector.

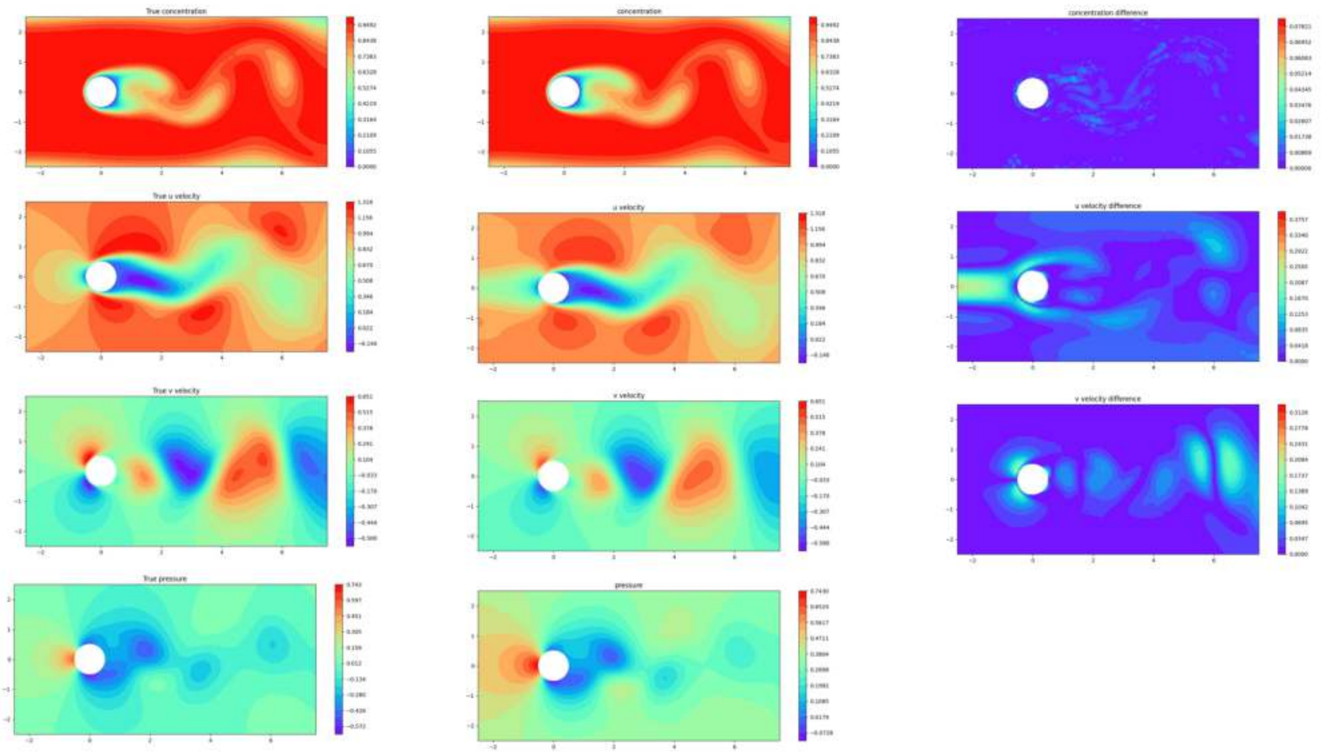


Figure 4. Predicted Results for PINN when given only concentration data and equations for the flow. The first column shows the true values for the output variables, the second column shows the predictions by the PINN and the final column shows the absolute difference between the prediction and actual data. The difference in pressure is not shown as they are offset by a constant

4.2.1. Results

Using only concentration data, the PINN is able to reconstruct the flow variables to a reasonable approximation. However, the PINN fails to reconstruct the the u velocity in front of the cylinder. This is likely due to the lack of concentration gradients in front of the cylinder where the flow is not disturbed by anything. To circumvent one would

Table 1. relative L2 error for the scalar-advection problem

	concentration	u velocity	v velocity
Relative L2 Error	0.50%	22.5%	8.20%

likely need to introduce a boundary condition at the left boundary to improve the PINN prediction.

A interesting point from figure 5 is that when the values of concentration outside of the range (0,1) are excluded from the contours, the network is able to determine the boundary condition of the cylinder even though no information about the boundary condition has been provided.

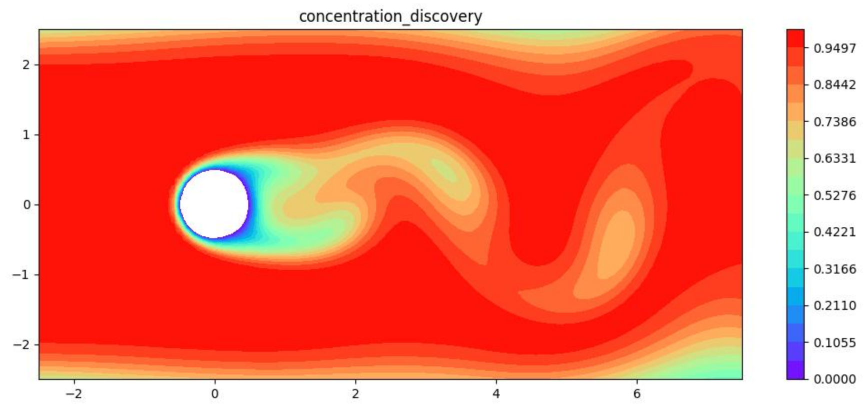


Figure 5. The cylinder can be 'discovered' by the PINN by removing points outside the expected range for concentration data

4.3. Steady State 2D Boussinesq Bouyancy Approximation

The previous example was an example of one-way coupling where the fluid affected the concentration but the concentration does not affect the flow. In this example, natural convection due to thermal gradients are examined. As opposed to the previous problem temperature affects the flow field and vice versa.

4.3.1. Problem Setup

In this example, a steady state differential heated cavity problem is analyzed. From Figure 6, the left wall of a 0.1×0.1 metre box is heated to 296K while the right wall is cooled to 294K. The fluid inside the box is water and is initially 295K. The top and bottom walls are thermally insulated (i.e. no heat can flow out of these walls). Each wall also obeys the no slip condition where fluid flow is zero at the walls.

The fluid dynamics inside the box is governed by the boussinesq approximation:

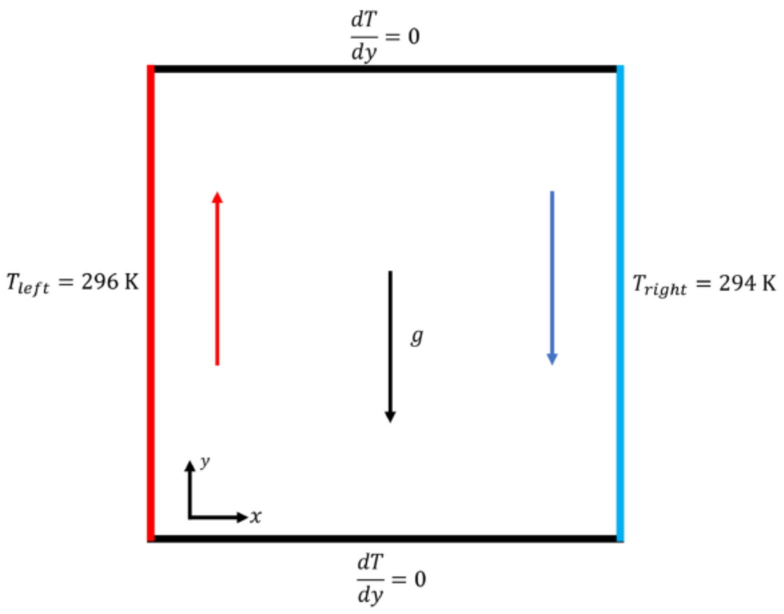


Figure 6. Set up for the heated cavity problem. The origin is located at the lower left corner of the box. The initial fluid temperature is at 295K while gravity is the usual -9.81 m/s^2. Hotter fluid rises to the top while colder fluid sinks to the bottom

Table 2. Material Properties used for the model

Reference Density ρ_0 (kg/m^3)	Dynamic Viscocity μ (Pa·s)	Thermal Diffusivity α m^2/s	Co-eff Thermal Expansion β $1/^\circ K$
998	0.00959	1.452e-07	0.000228

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho_0}\frac{\partial p}{\partial x} + \frac{\mu}{\rho_0}\nabla^2 u \tag{17}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho_0}\frac{\partial p}{\partial y} + \frac{\mu}{\rho_0}\nabla^2 v - \beta g(T - T_0) \tag{18}$$

$$\nabla \cdot \vec{u} = 0 \tag{19}$$

$$\nabla T \cdot \vec{u} = \alpha \nabla^2 T \tag{20}$$

Where T is the temperature of the fluid, ρ_0 is the density of the fluid at T_0, μ is the dynamic viscosity, β is the co-efficient of thermal expansion, g is the acceleration due to gravity, and α is the thermal diffusivity. The Boussinesq equations are very similar to the equations in section 4.1 but with the added buoyancy term due to gravity in equation 18. Here T_0 is the initial temperature of the fluid of 295K.

The CFD data for this problem was generated with Ansys Fluent with a 200 x 200 mesh size.

4.3.2. Non-dimensionalation and scaling

Scaling outputs and inputs closer to unity or centered around zero has been shown to greatly improve convergence and accuracy of neural networks [1] . An example outside of PINNs is in image recognition tasks where pixel values are scaled from 0-255 to 0-1.

To ensure unit consistency, the equations themselves must also be scaled and shifted appropriately with the aim of having outputs and inputs close to unity. To do so one non-dimensionalises the equations. An example of this are the equations seen in section 4.1 which have been non-dimensionalised using a length and velocity scale.

For this problem we choose a length, velocity and temperature scale $L, U, \Delta T$ respectively and non-dimensionalise in the following:

$$t = \frac{t^*}{L/U}, \qquad x = \frac{x^* - x_s}{L}, \qquad y = \frac{y^* - y_s}{L} \tag{21}$$

$$u = \frac{u^*}{U}, \qquad v = \frac{v^*}{U}, \qquad p = \frac{p^*}{\rho_0 U^2} \qquad \theta = \frac{T^* - T_0}{\Delta T} \tag{22}$$

where the superscript $*$ is the dimensionalised parameter and subscript s represents a reference value to shift the centre value to zero. For this problem $L = 0.01$, $U = 1e - 3$, $x_s = y_s = 0.05$ and $\Delta T = 1$. This leads to the following non-dimensional form of the Boussinesq equation:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{\mu}{\rho_0 UL}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{23}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{\mu}{\rho_0 UL}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) - \frac{\beta g l \Delta T \theta}{U^2} \tag{24}$$

$$\nabla \cdot \vec{u} = 0 \tag{25}$$

$$u\frac{\partial \theta}{\partial x} + v\frac{\partial \theta}{\partial y} = \frac{\alpha}{UL}(\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2}) \tag{26}$$

These scaling and shifts change the domain of our problem: the origin is now located in the centre of the box and has dimensions of 10 x 10 (ranging from -5 to 5). Furthermore, the non-dimensional temperature θ now ranges from -1 and 1 as opposed to 294K to 296K. In a similar vein as before, Only the temperature data is provided to the network during training and equations 23 to 26 are treated as residuals e_1 to e_4 respectively in the loss function:

$$\begin{aligned} Loss &= \sum_i^4 \mathcal{L}_{residual_i} + \mathcal{L}_{data} \\ Loss &= \sum_i^4 \frac{1}{N}\sum_j^N e_i(x_j)^2 + \frac{1}{N}\sum_j^N \theta_{net}(x_j) - \theta(x_j))^2 \end{aligned} \tag{27}$$

4.3.3. Training and Network

of the 40,000 training points available, only 8000 random points are used in training the network. The batch size used is 4000. Again, the Adam optimiser scheme is used with an initial learning rate of 1e-3. The network used is the same fully connected network as before with sin activation functions. Finally, $\mathcal{L}_{data}$ term is given a weighting of 20 over the residual terms.

4.3.4. Results

Table 3. relative L2 error for the steady state boussinesq problem

	Temperature	u velocity	v velocity
Relative L2 Error	2.57%	45.6%	38.2%

From Figure 7 one can see that the PINN primarily struggles in the regions near the outer walls of the cavity, in particular the corners of the cavity. This is likely due to the high gradients that occur near the walls, making convergence of the residuals difficult. Furthermore, the no slip condition is not met on the top and bottom walls for the u velocity. This is likley due to the lack of temperature gradients normal to the wall (note this is actually the boundary condition of the wall when set in the CFD model) which makes it difficult to infer the correct solution [4].

Futhermore, training of the network was difficult likely due to the loss being multi objective. As can see in the training loss in figure 8, the loss became unstable after about 5000 epochs. To stabilise the training, the learning rate was manually decreased to 1e-4 after 15000 epochs. This required manual intervention which is not ideal. From Figure 9 the noisiest loss is clearly the residual related to equation 24. This is likely due to the buoyancy term which contains the a temperature term.

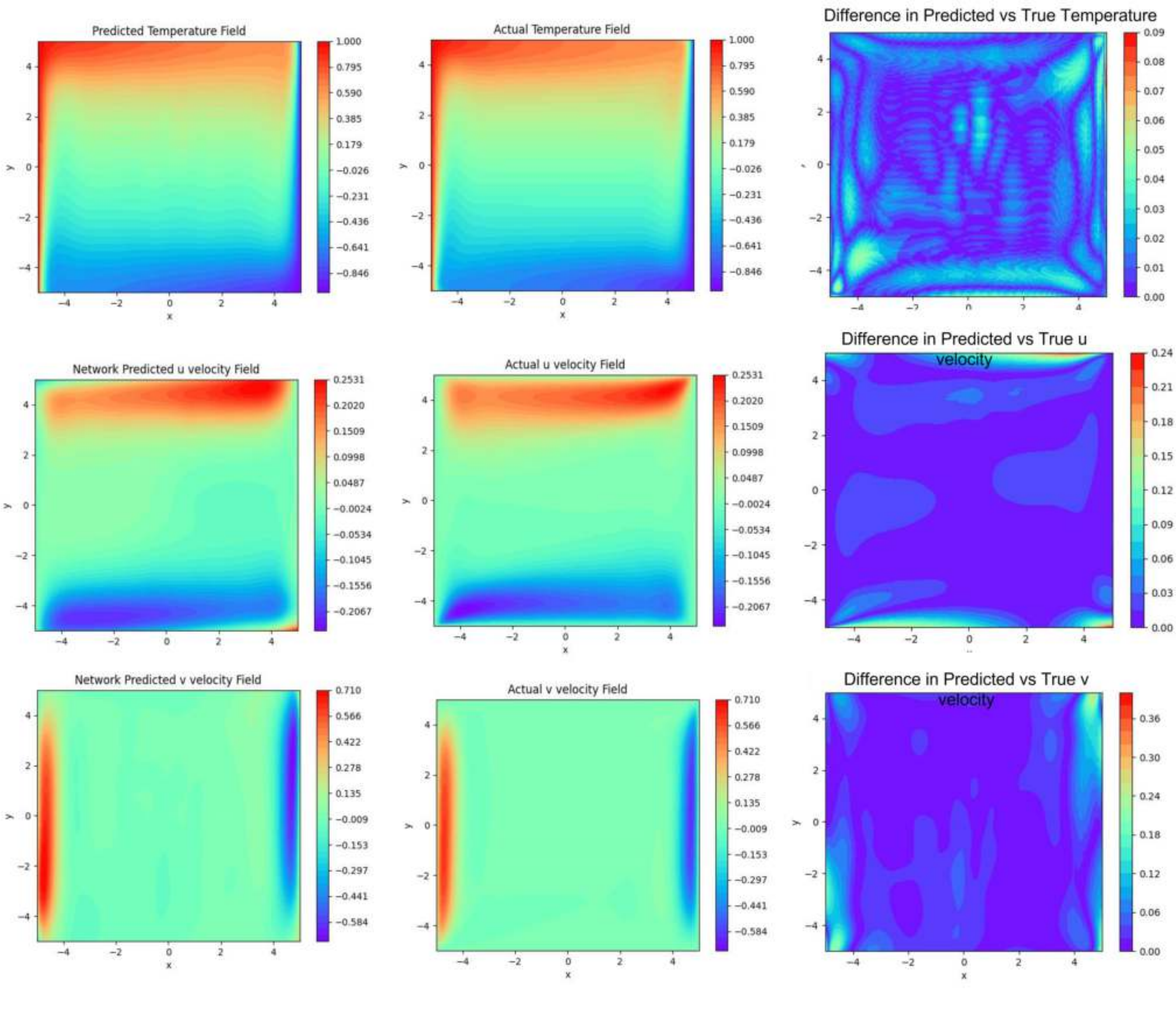


Figure 7. Predicted velocity and temperature profile of PINN when given only on temperature data. The PINN struggles to capture the zero velocity condition especially on the top and bottom walls

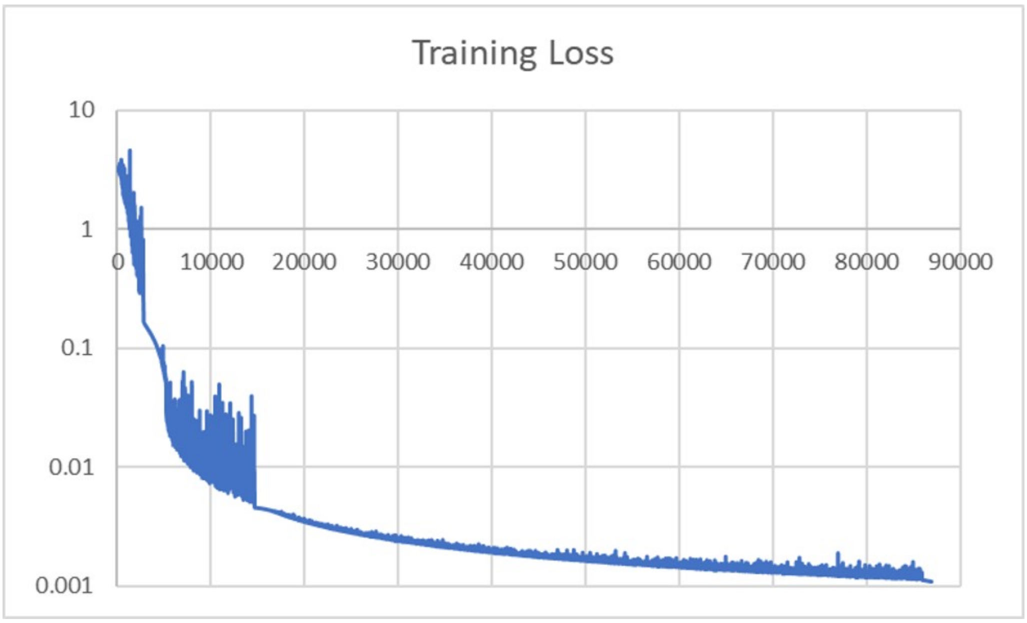


Figure 8. Training Loss over 90,000 Epochs

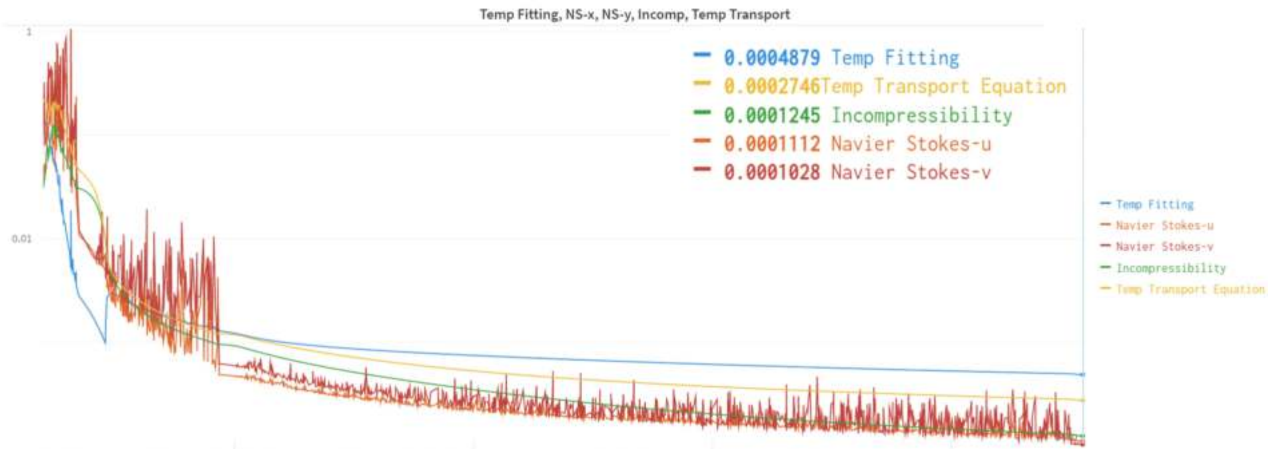


Figure 9. Individual training losses for each loss term. note incompressibily loss curve represents the continuity equation

4.4. Steady State Heated Cavity with Additional Boundary Conditions

To improve the PINN the no slip condition is introduced into the loss function:

$$Loss = \sum_{i}^{4} \mathcal{L}_{residual_i} + \mathcal{L}_{data} + +\mathcal{L}_{BC}$$
$$Loss = \sum_{i}^{4} \frac{1}{N} \sum_{j}^{N} e_i(x_j)^2 + \frac{1}{N} \sum_{j}^{N} \theta_{net}(x_j) - \theta(x_j))^2 + \frac{1}{M} \sum_{k}^{M} u_{net}(x_k)^2 + v_{net}(x_k)^2 \tag{28}$$

Where x_k are points located on the walls of the domain. Each wall has 250 points that are randomly sampled every epoch. The training of the previous cavity PINN is then finetuned to help with faster convergence. The network was then trained on an additional 3300 epochs.

4.4.1. Results

Table 4. Comparison of relative L2 error for the steady state Boussinesq problem with and without no slip boundary conditions

	Temperature	u velocity	v velocity
No BC	2.57%	45.6%	38.2%
With BC	3.77%	34,29%	21.2%

From Figure 10 introducing the boundary helps improve the error located near the walls and allows the PINN to capture the no slip boundary at the top and bottom walls. From table 4, the average relative error is significantly less when BC are introduced. However the average error is less for the u velocity, one can see that the u velocity is underestimating the true u velocity suggesting more training is needed.

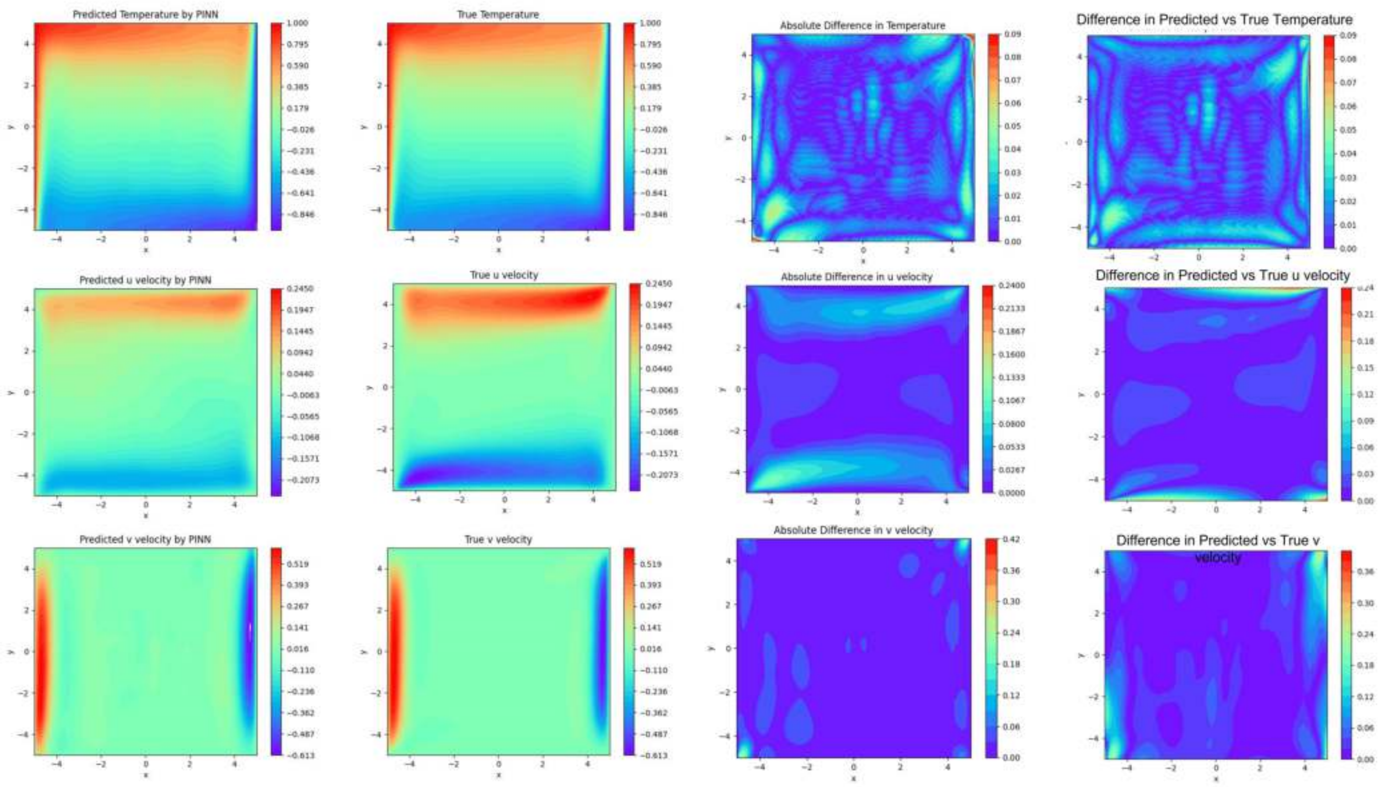


Figure 10. Results after the no slip condition is included in the loss function. the images on the right most column are the absolute errors from the previous PINN trained without the boundary conditions.

4.5. Transient 2D Boussinesq Flow

Finally a transient 2D Boussinesq Flow is simulated to see the feasibility. In this case the 2D model contains a heated cylinder at 315K and a cooling cylinder at a constant 285K with the surrounding air at an initial temperature of 300K.

The governing equations are now the transient Boussinesq approximations:

$$t = \frac{t^*}{L/U}, \qquad x = \frac{x^*}{L}, \qquad y = \frac{y^*}{L} \tag{29}$$

$$u = \frac{u^*}{U}, \qquad v = \frac{v^*}{U}, \qquad p = \frac{p^*}{\rho_0 U^2}, \quad \theta = \frac{T^* - T_0}{\Delta T} \tag{30}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{\mu}{\rho_0 UL}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{31}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{\mu}{\rho_0 UL}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) - \frac{\beta g l \Delta T \theta}{U^2} \tag{32}$$

$$\nabla \cdot \vec{u} = 0 \tag{33}$$

$$\frac{\partial \theta}{\partial t} + u\frac{\partial \theta}{\partial x} + v\frac{\partial \theta}{\partial y} = \frac{\alpha}{UL}(\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2}) \tag{34}$$

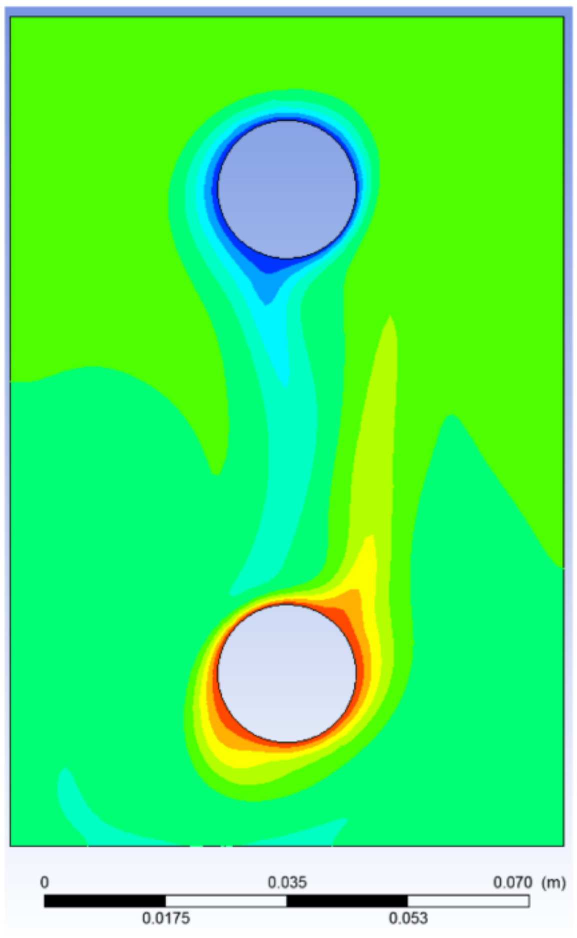


Figure 11. Geometry of model. The domain is a 0.08m x 0.12m rectangle with 2 cylinders of radius 0.02m. The bottom cylinder walls are at a constant 315K while the top cylinder acts as a heat sink at a constant 285K. The reference air temperature T_0 is 300K. The outer walls are thermally insulated i.e. $\frac{\partial T}{\partial n} = 0$

.

With $L = 0.1$, $U = 1$,$T_0 = 300$, $\Delta T = 30$. This sets the non-dimensional temperature range between -0.5 and 0.5. The PINN is trained over a 5 second interval in 0.1 increments with approximately 10,000 points of data per increment. All data points in this interval are used. The network is identical to that of section 4.3.3 except with an additional input in time. Adam is the choice of optimizer with an initial learning rate of 1e-3. The data loss is given and additional weighting of 20 and the batch size for training is set to 8000.

Table 5. Material Properties of Air used for the model

Reference Density ρ_0 (kg/m^3)	Dynamic Viscocity μ (Pa·s)	Thermal Diffusivity α m^2/s	Co-eff Thermal Expansion β $1/^{\circ}K$
1.225	1.7894e-05	1.963e-05	0.00338983

Table 6. relative L2 error for the transient boussinesq problem

	Temperature	u velocity	v velocity
Relative L2 Error	7.87%	60.58%	38.42%

4.6. Results

From the training curve in Figure 13 show sporadic spikes at epochs 1300 and 2900. The first spike at epoch 1300 was a change in weighting and accidental overwrite and overwrite of the best running loss. The second spike was due to a change in weighing. The weighing for the temperature transport residual was set to 10 while the temperature fitting loss remained at a weighting of 20. Due to time constraints only 10,000 epochs were run. Due to time constraints, the run was prematurely ended. More training time would be needed to improve accuracy.

In a similar vein to section 4.1, the PINN is able to start to infer the boundary of the 2 cylinders as seen in Figure 15. More training would likely be needed

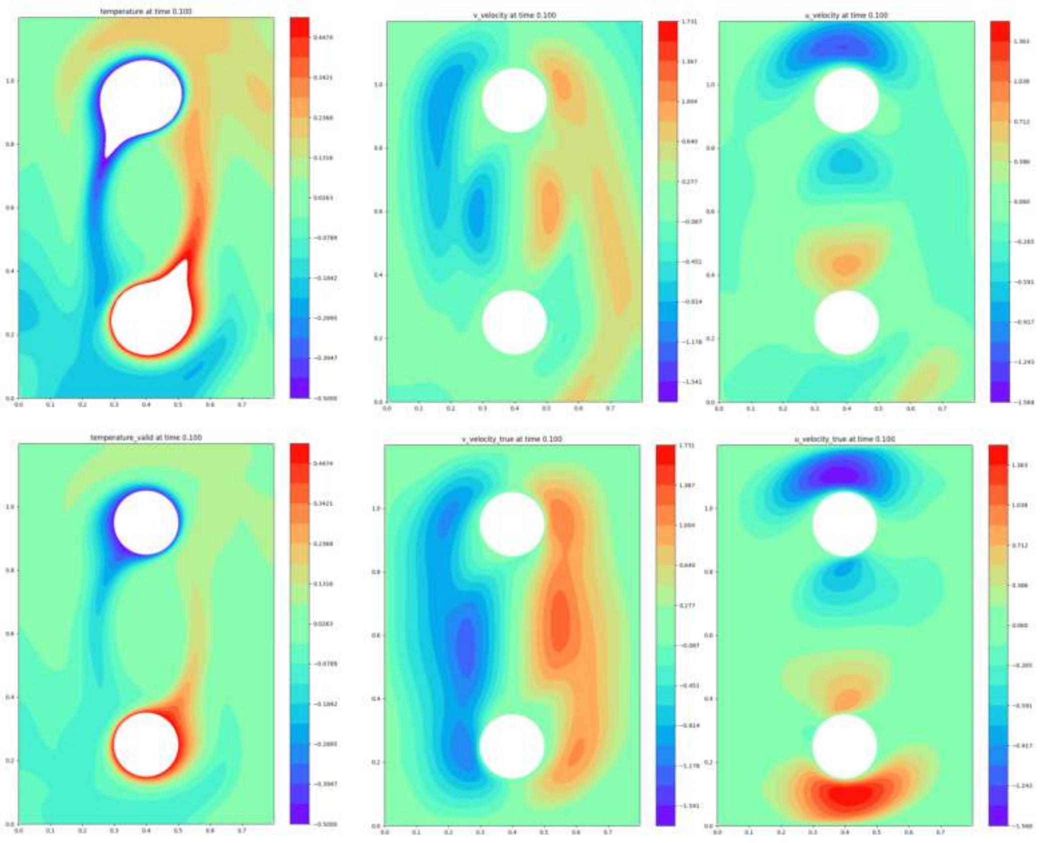


Figure 12. Results for the transient case. More training is likely needed to achieve better accuracy

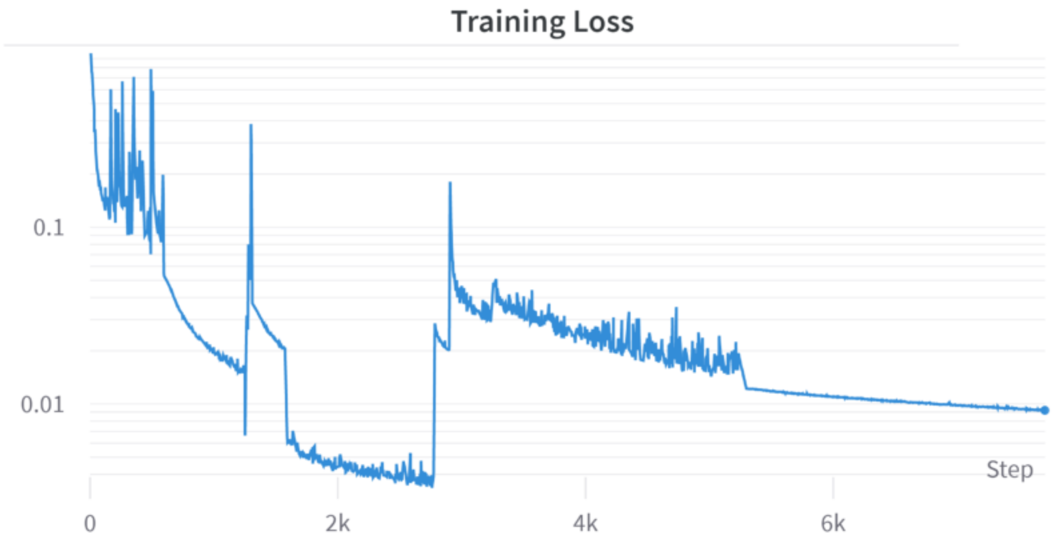


Figure 13. Training Loss over epochs. The spikes at step 1300 and 2900 are due to experimental changes in weighting and learning rate changes

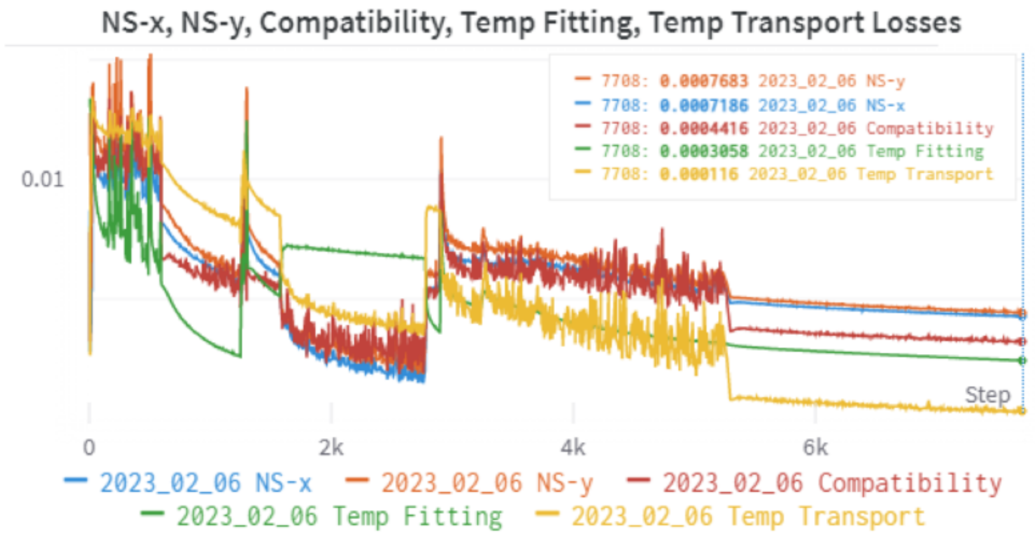


Figure 14. Individual Training Losses over epochs. The spikes at step 1300 and 2900 are due to experimental changes in weighting and learning rate changes

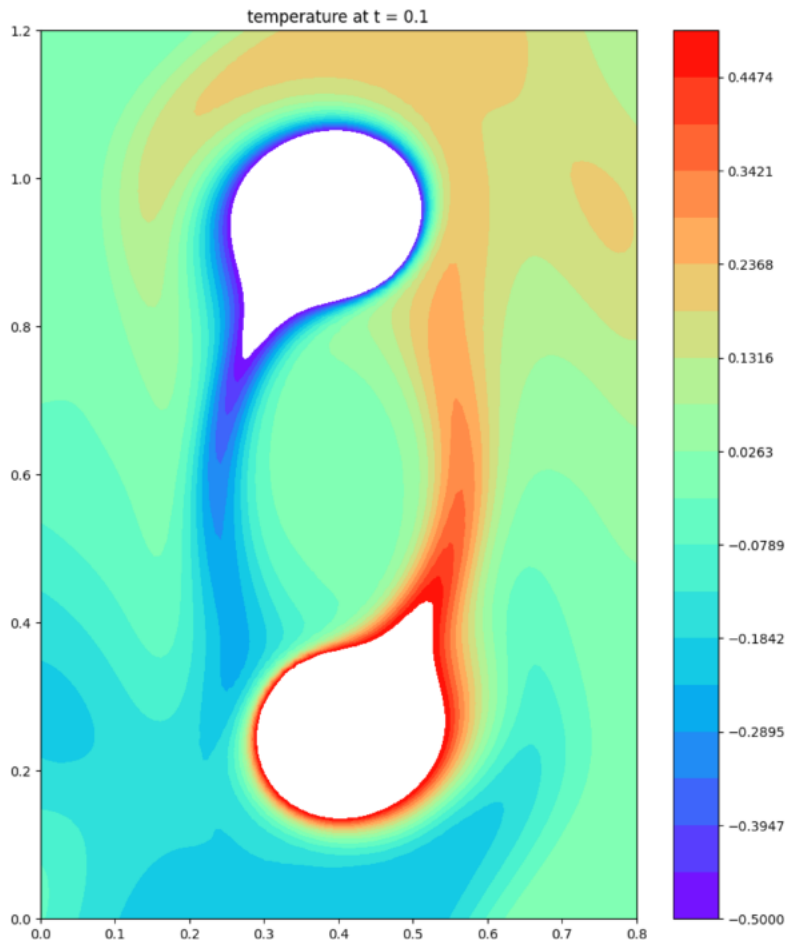


Figure 15. When values outside the temperature range of -0.5 and 0.5 are excluded, the PINN can begin to infer the location of the boundary conditions of the 2 cylinders

5. Challenges Faced

5.1. Network Selection and Activation Function

Network architecture remains an important choice. During the research project, numerous networks were tested aside from the standard fully connected network. Most of the networks tested are outlined and used by Nvidia Modulus [8]:

1. Fully Connected Network - a MLP network used by Raissi [4] who suggests each hidden layer should have 50 neruons per output
2. Fourier Network - model is first mapped to a frequency-esque domain and then followed by a standard output
3. Modified Fourier Network - Similar to Fourier Network but is better able to capture the multiplicative relationships
4. Residual Network - This was a custom network that had skip connections similar to Resnet for image processing

Due to limited time, a true comparison between different networks was not performed. The difference between the networks was not significant enough to notice a dominant network although from external results suggest the Fourier and Modified Fourier can perform better than the fully connected network [8]. The networks used in this research were either the resiudal like network or the fully connected network.

For activation functions, it was observed that $\phi = \sin(x)$ performed significantly better than standard activation functions such as $\tanh$ or the sigmoid function.

5.2. Weighing of Loss functions

For PINNs the resiudals create a multi-objective loss function. This can make is difficult to achieve convergence as the network may prefer to decrease the loss of one objective than another. Generally speaking, data fitting terms (including boundary conditions and initial conditions) should be weighted higher than residual loss terms [9][10]. A significant amount of research into PINNs attempts to tackle this problem to automatically weight these objectives to have approximately the same gradient magnitude. The two weighting methods attempted were the weighting via Neural Tangent Kernels and learning rate annealing [10][9] strategies. However, implementing both weighting systems failed. Correctly weighting the different objectives has been shown to significantly improve convergence [10][9]. From this project, weighting the data term higher has been beneficial in improving convergence.

Secondly one can also have spatial weighting of the residuals terms in the loss functions. Areas of high gradients such as near a wall or discontinuities, can often be source of convergence issues. Decreasing the weight of residuals near walls might have helped improved convergence.

6. Future areas to investigate

6.1. Exact Continuity for Navier-Stokes

In 2D, one can automatically satisfy the continuity equation $\nabla \cdot \vec{u} = 0$ for incompressible fluid flow by defining the velocity as the derivatives of a scalar potential ϕ known as the stream function [9]:

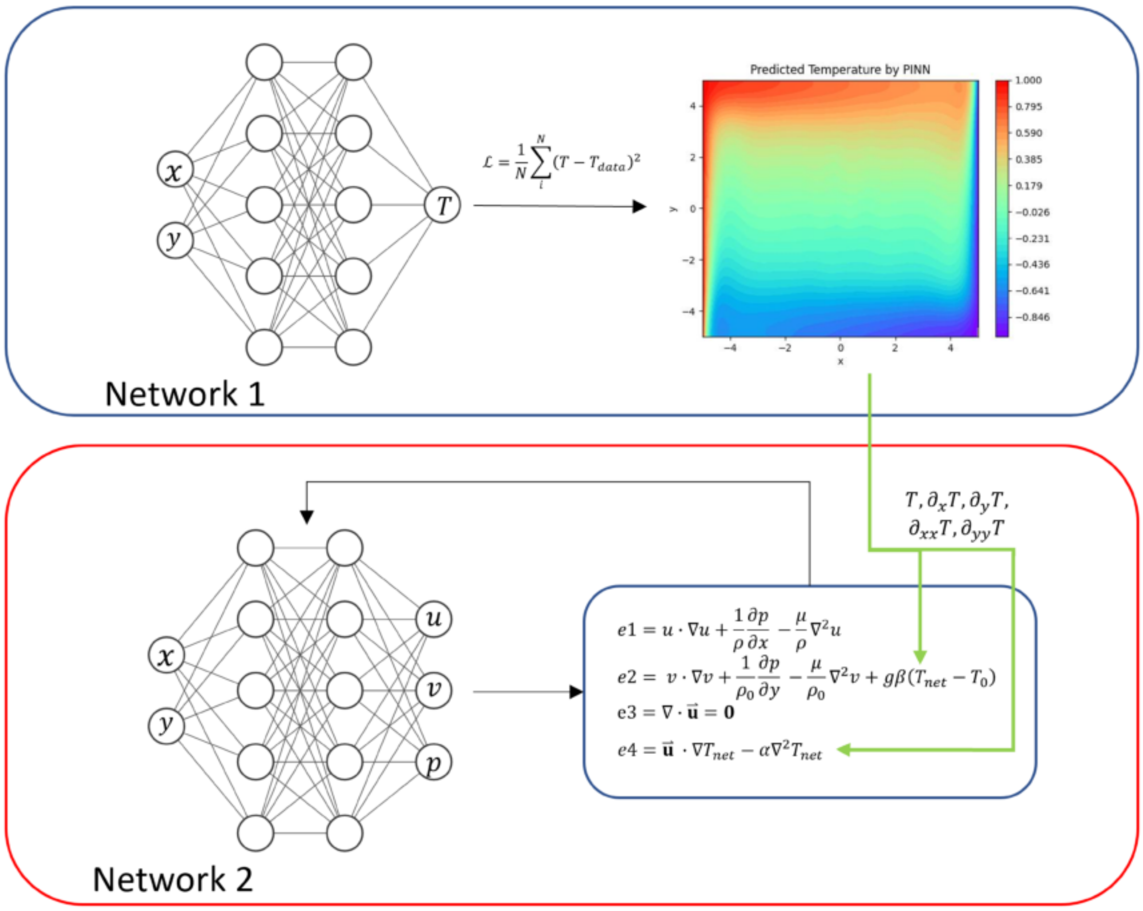
$$u = \frac{\partial \phi}{\partial y}, u = -\frac{\partial \phi}{\partial x} \tag{35}$$

In this case, the network would output the stream function ϕ and pressure. Velocity would then be recovered by taking the appropriate gradient of ϕ. This formulation ensures the continuity equation is satisfied and removes simplifies the multi-objective loss function. Reducing the objective function should help improve convergence [9]. However, the trade-off is that the Navier stokes equations become third order PDEs increasing computation complexity.

6.2. Using 2 networks

For inverse modelling if one has data that covers the domain, such as the examples this project examined, one could split up the training into to 2 networks. The first network learns to predict the scalar (e.g. temperature) field and its derivatives while the second network uses the derivatives of the first network to determine the other hidden variables.

Taking the steady state natural convection problem in section 4.3 as an example, one first trains a network to learn the temperature field $\theta(x, y)$ through standard mean square error fitting. At high convergence, this network gives us the temperature and the respective gradients $\frac{\partial\theta}{\partial x}, \frac{\partial\theta}{\partial y}, \frac{\partial^2\theta}{\partial x^2}$ and $\frac{\partial^2\theta}{\partial y^2}$. Then the second network is trained to find the unknown variables $u(x,y), v(x,y), p(x,y)$ with the loss function being only the residuals. Information of the temperature field is passed to the network through the temperature gradients. During optimisation of the second network, the temperature gradients are not part of the computational graph and treated like constants.



x
y
T
$\mathcal{L} = \frac{1}{N}\sum_{i}^{N}(T - T_{data})^2$
Predicted Temperature by PINN
1.000
0.795
0.590
0.385
0.179
−0.026
−0.231
−0.436
−0.641
−0.846
4
2
0
−2
−4
y
x
Network 1
$T, \partial_x T, \partial_y T,$
$\partial_{xx} T, \partial_{yy} T$
x
y
u
v
p
$e1 = u \cdot \nabla u + \frac{1}{\rho}\frac{\partial p}{\partial x} - \frac{\mu}{\rho}\nabla^2 u$
$e2 = v \cdot \nabla v + \frac{1}{\rho_0}\frac{\partial p}{\partial y} - \frac{\mu}{\rho_0}\nabla^2 v + g\beta(T_{net} - T_0)$
$e3 = \nabla \cdot \vec{\mathbf{u}} = \mathbf{0}$
$e4 = \vec{\mathbf{u}} \cdot \nabla T_{net} - \alpha\nabla^2 T_{net}$
Network 2

Figure 16. Set up for inverse modelling with 2 networks. Network 1 learns the temperature field via MSE which implicitly also learns the spatial derivatives for the temperature field. Network 2 recovers the unknown velocities and pressure by passing in the temperature derivatives. the temperature values for Network 2 are treated as constants

By splitting up the task, one should be able to obtain higher quality temperature derivatives, as one does not need to compete with other objectives in Network 1, and simplify the objective function for network 2 helping with convergence. The downside of this method is that this method would only work in regions that contains temperature data. On the contrary, the methods outlined by Raissi [5] and in this report can be used when the data is very sparse and or does not cover the entire domain.

7. Conclusion

This research project examined how physics informed neural networks, or PINNs, can be used to recover unknown quantities when only given partial data of a single scalar field such as temperature or concentration. PINNs are able leverage both the governing equations and data where traditional numerical simulations or a purely data-driven approach fails. Using only thermal data, one could reasonably recover the velocity variables from incompressible flow drive by thermal gradients in both steady state and transient settings. From a numerical simulation point of view, one would still consider the error rate in these PINNs to be too high to be considered converged. The methods implemented during the project were a somewhat brute force approach with many instances of manual tuning and intervention. Convergence of the loss functions remains a difficult problem that would need to be explored further.

References

[1] *Modulus user guide*. URL: https://docs.nvidia.com/deeplearning/modulus/index.html.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[3] Guofei Pang, Lu Lu, and George Em Karniadakis. “fPINNs: Fractional Physics-Informed Neural Networks”. In: *SIAM Journal on Scientific Computing* 41.4 (Jan. 2019), A2603–A2626. DOI: 10.1137/18m1229845. URL: https://doi.org/10.1137%5C%2F18m1229845.

[4] M. Raissi, P. Perdikaris, and G.E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[5] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. “Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations”. In: *Science* 367.6481 (2020), pp. 1026–1030. DOI: 10.1126/science.aaw4741. eprint: https://www.science.org/doi/pdf/10.1126/science.aaw4741. URL: https://www.science.org/doi/abs/10.1126/science.aaw4741.

[6] *Using Nvidia modulus and Omniverse Wind Farm Digital Twin for simulate gamesa*. URL: https://resources.nvidia.com/en-us-energy-utilities/siemens-gamesa-wind.

[7] Maziar Raissi, Alireza Yazdani, and George E. Karniadakis. “Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data”. In: *CoRR* abs/1808.04327 (2018). arXiv: 1808.04327. URL: http://arxiv.org/abs/1808.04327.

[8] *Architectures in modulus*. URL: https://docs.nvidia.com/deeplearning/modulus/user_guide/theory/architectures.html.

[9] Sifan Wang, Yujun Teng, and Paris Perdikaris. “Understanding and mitigating gradient pathologies in physics-informed neural networks”. In: *CoRR* abs/2001.04536 (2020). arXiv: 2001.04536. URL: https://arxiv.org/abs/2001.04536.

[10] Sifan Wang, Xinling Yu, and Paris Perdikaris. “When and why PINNs fail to train: A neural tangent kernel perspective”. In: *CoRR* abs/2007.14527 (2020). arXiv: 2007.14527. URL: https://arxiv.org/abs/2007.14527.