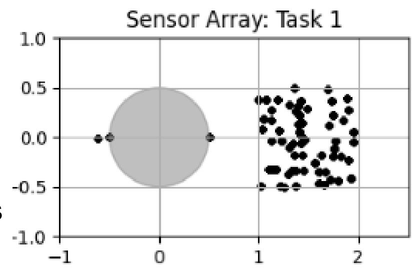
Justin Miller
ENGN2912v Project Writeup: Inverse Heat Transfer Problem

Problem Overview:
The inverse heat transfer problem is a particularly difficult problem to solve using simulation methods because of the lack of known boundary conditions. In this project, a Physics Informed Neural Network (PINN) is used to attempt to infer the thermal boundary conditions of a cylinder undergoing mixed convection. This PINN will also attempt to get accurate measurements of temperature and velocity fields near the cylinder of interest. To develop these models, the PINN will extract data from a spectral/hp CFD simulation, choosing certain nodes to mimic sensors in an experimental setting. While this simulation was run with known cylinder temperature and heat transfer properties, the PINN will only be provided the raw data and the assumption that the cylinder temperature is an unknown constant.

To explore performance in a range of engineering applications, three tasks will be attempted. The first task will be to achieve the best possible PINN performance using a robust patch of sensors. The second task will be to explore performance on a reduced number of sensors, aiming to learn more about the capabilities of PINNs when sensor availability is constrained. The last task will be to explore the applicability of PINNs on noisy sensor data, this time leveraging Bayesian PINNs to get the best result.

Data / Experimental Setup:
The existing CFD dataset was provided in the form of a .dat file. The raw file includes temporal spatial coordinates on the typical simulation ranges of x=[-7.5, 22.5], y=[-10, 10], with 214 uniform timesteps from 0 to 21.3 seconds. Each timestep includes 42,600 nodes, with nonuniform nodes concentrated near the cylinder of radius 0.5 located at [0, 0]. This data came with output data on Temperature, pressure, U velocity, V velocity, and vorticity. The temperature data was on the scale 0 to 1, the pressure and velocities were also reasonably scaled for a neural network while the vorticity had some higher values that would have to be addressed if incorporated. This simulation was governed by the 2D Incompressible Navier Stokes Equations as well as the temperature equation with given flow parameters (Re=100, Pe=71, Ri=1.0). The following tasks were carried out on a Windows machine utilizing a RTX 4070 GPU. All code was written in Python using vscode and the jupyter notebook extension.

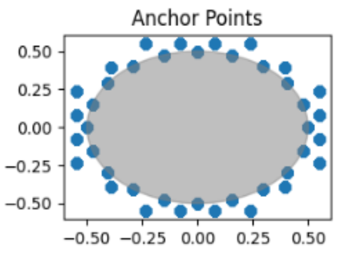
Task 1: Method
To develop a PINN predicting thermal boundary conditions using sensor data, the first step was to extract a patch of sensors from the CFD data to mimic a laboratory experiment. This patch was chosen using a random selection of 64 points from the bounding box of [[1, 2],[-½, ½]]. Additionally, points at the leading and trailing edge of the cylinder, with an additional point at [-0.6, 0] were selected as sensors.The CFD simulation was subset to include these points from all timestamps in a training dataset. The rest of the points from the simulation were reserved for testing and validation.



Once the data was extracted and split, the DeepXDE library was used to craft a model with a Torch backend. The geometry submodule was used to create the rectangular domain with an empty circle with the proper time component. Once the domain was initialized, a trainable torch variable “TEMP_C” was defined with an initial value of 0.1 to represent the constant temperature of the cylinder. Then, the loss conditions were defined. The first loss condition was the custom pde() function, which encoded the Temperature Equation, X-Momentum, Y-Momentum and Continuity in four separate loss values using Auto Differentiation. Additionally, the cylinder surface temperature loss was defined in this PDE function to allow for easy training of the TEMP_C variable. This was done using the torch.isclose() method to create a boundary loss for any point located on the cylinder that wasn’t equal to TEMP_C.

Once the equation loss was defined, the dde.icbc.PointSetBC() method was used to set up losses for the sensor data. Four of these were crafted, with each one representing the loss in Temperature, Pressure, U Velocity, or V velocity for all available sensor points. Note that, for this experiment, vorticity was ignored as attempts to incorporate the 2d vorticity equation ($\omega = \nabla \times \mathbf{u}$) added more computation time than benefit. In addition to the sensor data, the actual boundary conditions were incorporated using the dde.icbc.PointSetBC() method. These included both the wind tunnel boundary conditions (T=0, u=1, v=0 at all edges of the rectangular domain) and the no-slip condition (u=v=0 at the surface of the cylinder). All in all, this resulted in 13 separable loss components that could be monitored and reweighted if needed.

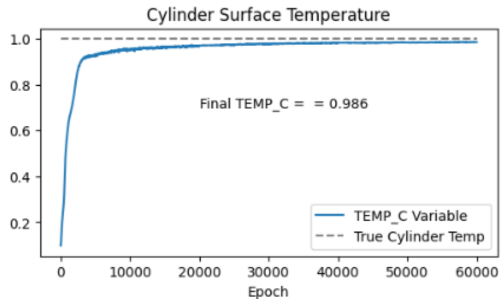
Once all losses were defined, the deepXDE model was initialized and ran. The model used a torch.nn of 6 layers of 50 width with 3D input and 4D output. Domain and boundary sampling was done using a num_domain=10000 and num_boundary=1000, with an additional 8000 anchor points on and around the cylinder to ensure sufficient sampling near the area of interest. The randomized domain and boundary points were resampled every 100 epochs. In addition, the trainable variable TEMP_C was updated every 10 epochs. All of this was trained using 60,000 epochs of ADAM starting with a learning rate of 0.001 and decaying at a multiplier of 0.1 every 20,000 epochs.



Note that these parameters were not chosen arbitrarily, many iterations were run to balance the batch coverage and training efficiency. The guiding paper [1] used 800,000 epochs of a much more complex model (150 width, 10 layers, adaptive learning), which was not possible to replicate using the available hardware and timeframe. As such, decisions were made to focus on results very close to the cylinder and on the project’s most relevant variable (temperature).

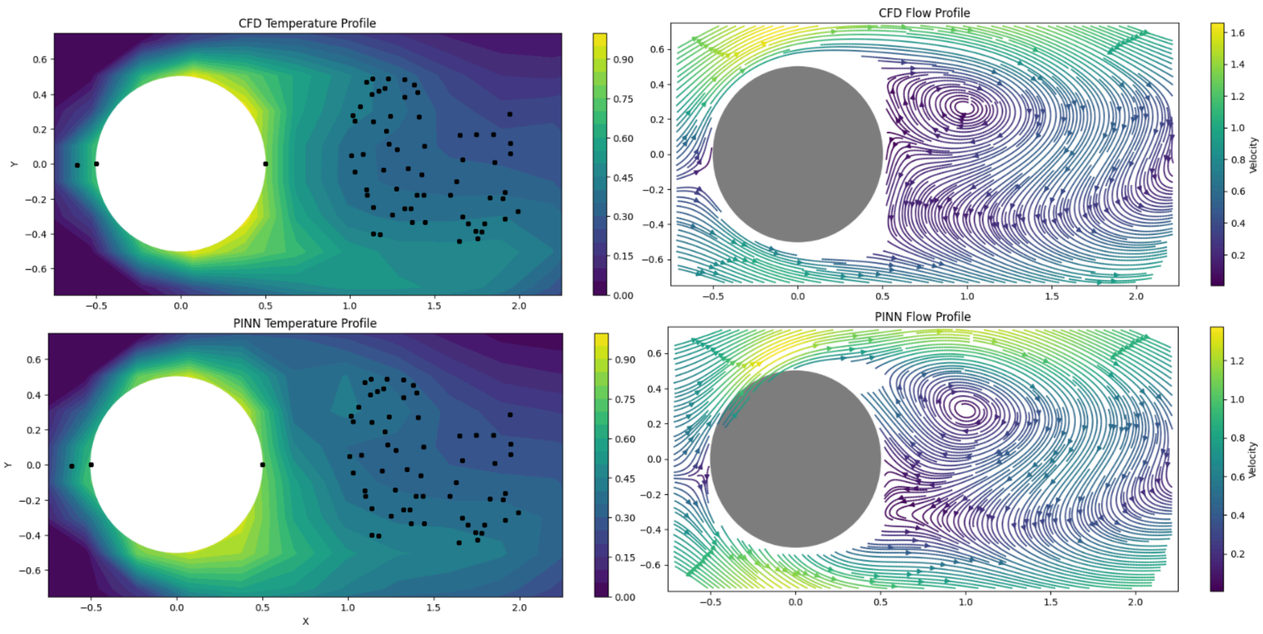
Task 1: Results

The above model took around 6 hours to train for each attempt. The resulting train curve was relatively successful for both the Equation Losses and the Sensor losses without requiring any tuning of category weights. In addition, the model was able to train the TEMP_C external variable very well, converging quickly and estimating the final value at 0.986 for a temperature boundary condition error of just 1.4%. Upon convergence, the total PDE loss for the model is 0.0014. Recall that this



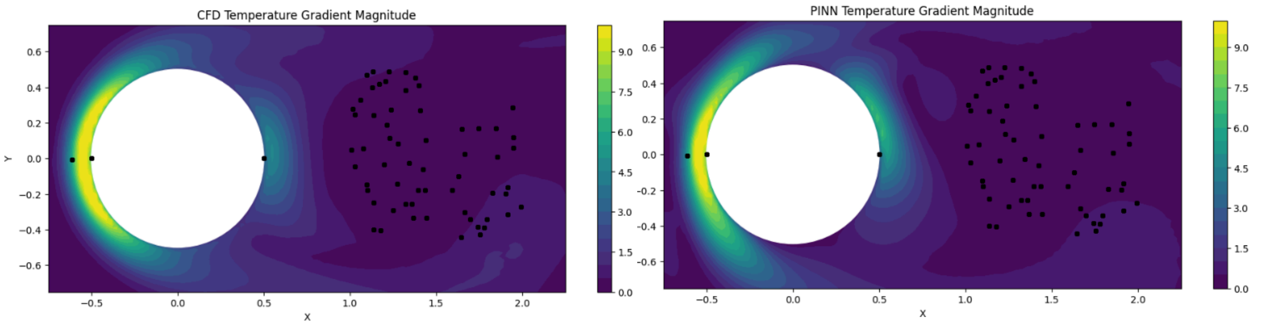
value represents the MSE of 8000 points on/near the cylinder, 1000 points at the sensors and other boundary conditions, and 10000 points randomly chosen throughout the domain so this result represents a relevant error metric. Additionally, the total sensor loss converges to 0.00039; note that this just represents the errors at sensor locations.

In addition to the pure model losses, another way of measuring results is by comparing the local behavior in temperature and velocity. As these two predicted variables are key properties in determining convective heat transfer, appropriate behavior near the cylinder is critical in determining if the PINN can create an accurate model. The images below show the comparisons between the true and predicted values for Temperature and Velocity on the region of most interest. The relative L2 values on the relative interval [[-¾,2¼],[-¾,¾]] is 0.0011 for temperature and (0.0019, 0.0025) for (u, v) velocity. Overall, the PINN generalizes very nicely.



Lastly, the most succinct way of measuring success can be boiled down to two numbers: the familiar TEMP_C and the calculated Nusselt number. The temperature of the surface cylinder has already been confirmed to have a successful training. Since the Nusselt number is not one of the predicted values, we need to calculate that from the network. Since the flow temperature difference is 1 and the cylinder diameter is also 1 the Nu value simplifies to dT/d**n**. When using automatic differentiation on the network, we achieve a mean Nusselt number of 3.96. Note that this value is quite different from the true averaged Nusselt number which can be calculated as 5.90 from the experimental data. This extremely high error highlights the main issue with using this simplified neural network compared to the complex network from the paper, which yielded a Nusselt number with only one percent error. The Nusselt number is very sensitive to errors because it is only calculated at the boundary of the cylinder. Even if the model correctly predicts behavior in the domain any imperfections will be glaring as this is on the edge of the domain.

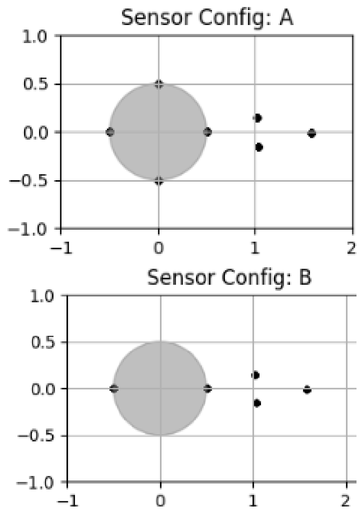
The plots on the next page compare temperature gradients (calculated numerically as we can not use Auto Differentiation on the CFD results) to highlight this source of error. Note that these



plots show the magnitude of the gradient, not in the direction of the cylinder's **n**, but that these quantities are the same at the cylinder's surface.The visualizations show the temperature gradient is very accurate at most of the domain, especially at the leading edge of the cylinder which has the biggest impact on the overall heat transfer. This indicates that the model has trained well within the open domain. However, focusing on the exact cylinder edge, easily seen farther away from the sensors, we see that the model does not train nearly as well given the lack of data on both sides of the boundary of interest. Much of the time spent on this project was attempting to improve this Nusselt accuracy by increasing sampling near the cylinder, yet in the end computational constraints (and perhaps user error) caused this Nusselt accuracy to be a persistent issue through all three tasks.

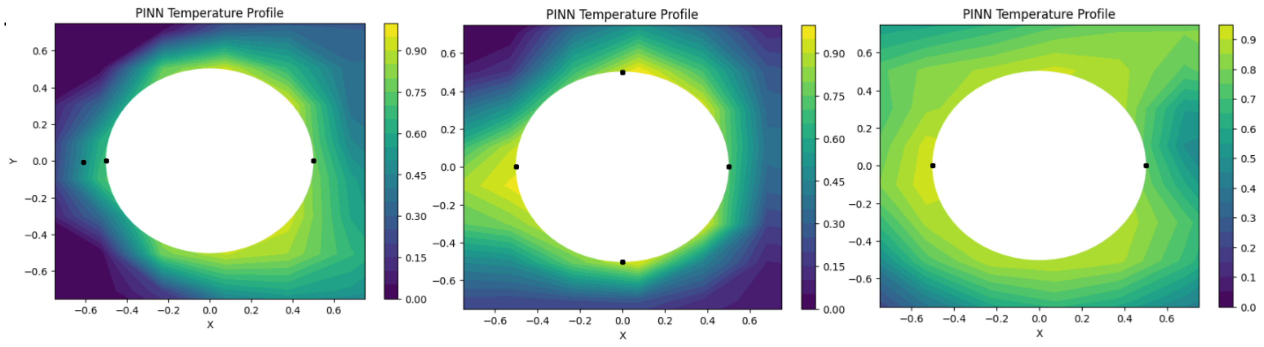
Task 2: Method

To explore the applicability of PINNs with less available sensors, the experiment from task 1 was repeated on 2 reduced configurations to draw conclusions about training behavior and capabilities. Note that, due to time constraints of training models to convergence, I was not able to implement an active sensor scheme to fully achieve the goals mentioned in the project statement for this task. Instead, I will use the results from these two reduced configurations to draw preliminary conclusions about the data required for PINNs to achieve success in a computationally intensive problem such as inverse heat transfer. For this task, the same exact network architecture, losses, and training process was used to provide comparable models



Task 2: Results

The results from sensor configurations A and B give us immediate insights on the number of sensors needed to approximate the surface temperature value. Configuration A, with it's four sensors on the cylinder surface, quickly converges to the true value giving a final estimate of TEMP_C=0.992. Configuration B, on the other hand, converges to an incorrect value of 0.898, giving us a 10% accuracy rating. In addition, we see that the relevant predictions near the cylinder are less and less accurate as sensors are removed. This is true for both velocity, temperature, and temperature gradient, but just temperature is shown for brevity (plots for the Robust sensor array, config A, and config B are shown respectively on the top of the next page). These plots allow us to draw conclusions about the behavior of PINNs when provided with different amounts of data, and with data in different locations.



As seen in the plot, running the PINN without a robust sensor array somewhere in the domain makes training much more difficult. As we go from 67, to 7, to just 5 points we see that the overall quality of the predicted fields goes down until it is clear that the model is not able to truly converge at this number of epochs (note that it is possible that extra training would clean up the nonsensical behavior of configuration B into something useable). Another insight is that sensor position needs to consider the problem's physics. Even though configuration A has more points plastered on the cylinder surface, its temperature field is less accurate near the cylinder because the leading point of the robust sensor task provides crucial information on tip velocity.

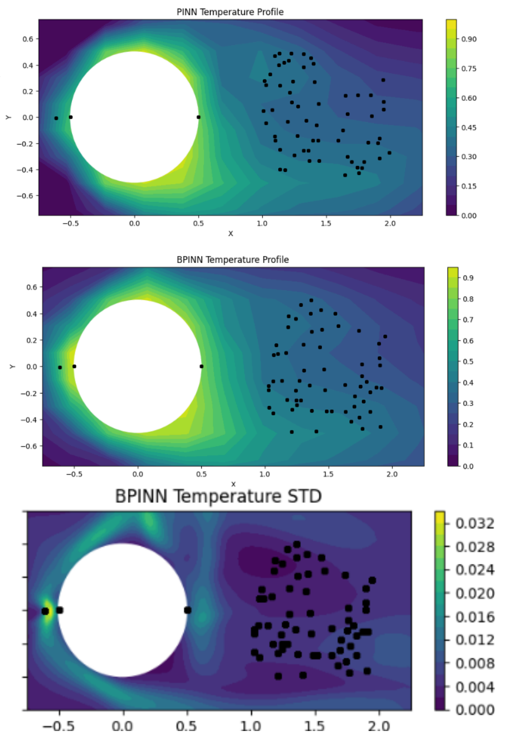
Task 3 Method:

To explore the applicability of PINNs with more realistic sensors, a 5% artificial noise is added to each sensor. Tasks 1 and 2 are then repeated using a Bayesian PINN [3], and by comparing these results we explore the ability of BPINNs to withstand noise and excel in real life situations. This task is carried out using the same processes except the torch.nn() network is replaced by a torchbnn network incorporating 6 BayesLinear layers of width 50 with initial weights of mean 0 and standard deviations that begin at 0.1 and decrease by a multiplier of 0.1 every two layers. To simulate sensor noise, all data is given an 5% gaussian noise on each independent variable.

Task 3 Results:

The results of task three show strong inferences from the BPINNs, almost matching the PINN results regardless of sensor noise. The TEMP_C converges to a relatively accurate value of 0.978. While plots are omitted for brevity, it is worth noting that the BPINNs for sensor configurations A and B both resulted in similar results as the PINNs for those sensor configurations.

Another benefit of the BPINN is highlighted where this temperature profile isn't totally accurate. The third plot on the right shows the standard deviation of temperature as inferred by the BPINN. This peaks at σ=0.032 right around the leading sensor where we also see the greatest error between our PINN and BPINN. The ability to provide uncertainty is extremely important to safely apply models to real life engineering problems. This feature makes BPINNs very useful compared to black box models.



References:

[1] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks for heat transfer problems. Journal of Heat Transfer, 143(6):060801, 2021.

[2] A. F. Psaros, X. Meng, Z. Zou, L. Guo, and G. E. Karniadakis. Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. arXiv preprint arXiv:2201.07766, 2022.

[3] L. Yang, X. Meng, and G. E. Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data. Journal of Computational Physics, 425:109913, 2021.