ISyE 6740 - Spring 2022
Final Project Report

Team Member Names: Andrew Bartels

Project Title: Lid Driven Cavity Flow: A Physics Informed Neural Network Approach

Contents

List of Figures

1 Problem Statement

Computational fluid dynamics (CFD) is the process of mathematically modeling a physical phenomenon involving fluid flow and solving it numerically using computers[**SimScale**]. From winning race cars to commercial aircraft that people fly in every day, CFD is a fundamental process in the pipeline from idea to design, while saving millions in costs. Rather than building multiple versions and prototyping (which is exceedingly expensive), engineers today will simply run a series of models to find the best design and then iterate this into a final product (see figure 4). The main issue with this is depending on the detail and length of the simulation, even the most basic simulations will either cost many thousands of dollars on a high performance computing (HPC) cluster, or take a very long time to run on a standard desktop or laptop (usually in terms of weeks to months).

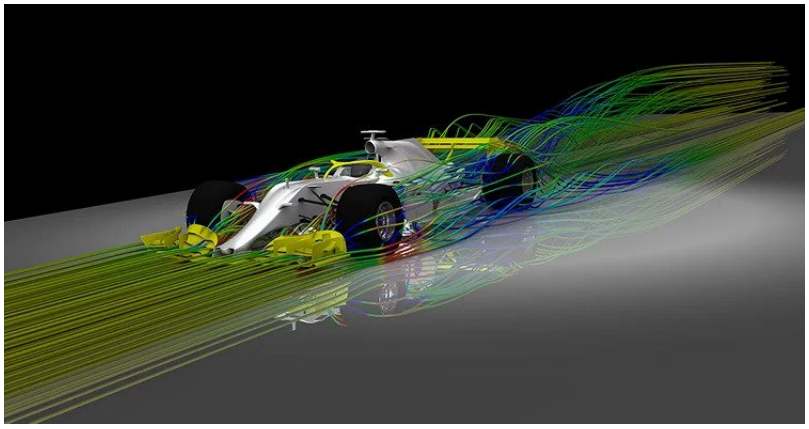

Figure 1: A picture of CFD model [**ANSYS.com**]

In the separate field of machine learning, neural networks and other non-explicitly programmed algorithms are estimated to be an industry in the billions [**nn˙history**]. This has led to an explosion in interest and growth into the field, with researchers and scientists applying neural networks to more applications than just image identification. While these statisical models can capture nonlinear effects very well, they often require hundreds or millions of examples to fit a model, where generalization of new data is not always guaranteed. Conversely, in CFD, data is often measured in terms of gigabytes to petabytes, so sifting through this data for meaning or trying to feed more than a few examples to a neural network can be extremely prohibitive (time and cost) [**DBLP:journals/corr/abs-1711-10566**] This leads to the introduction of physics informed neural networks (PINNs) [**PINN˙History**] . This is where the loss function of the optimization is substituted for a part of physics (in this study, the continuity equation), in order for the model to adhere to physics better. This has led to models that adhere to physics applications much better with less data, converge to a solution faster, and put "guard rails" on models that attempt to predict vastly different phenomena that might be practical [**DBLP:journals/corr/abs-1711-10561**]

1.1 Problem Setup

In this study a classic fluid mechanics problem will be observed with an application of PINNs, specifically the Lid-driven cavity flow problem (or sometime referenced as the driven cavity flow problem). This is a 2 dimensional application of the Navier-Stokes equation with the simplifications of being in 2 dimensions, steady-state, incompressible fluid flow, at lower Reynolds number (See Theory[4]) ($\leq$ 2000). This is a well studied test case with many permutations such as nonlinear gridding, spectral methods and different optimization algorithms for faster convergence, however this is all outside the scope of this study. The algorithm used as baseline is the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) algorithm. For the baseline examples to reproduce Reynolds numbers between roughly 300-1,900. A PINN will then be trained with the same grid size and continuity equation as the optimization function. A comparison between the baseline and machine learned approach will then be quantitatively compared[4]. This will then give a further insight into the usefulness of PINNs, and how generalizable something like this could become in the future development in this field of study.

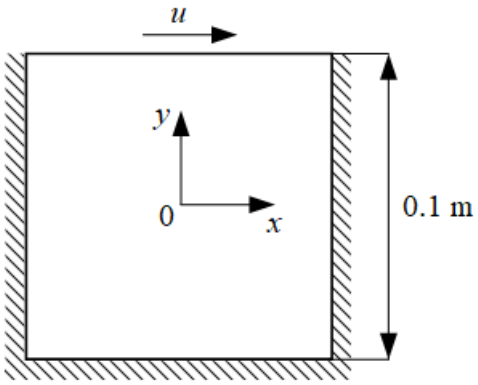

Figure 2: Open Lidded Cavity Flow Free Body Diagram [**NVIDIA**]

1.2 Computational Fluids Overview

Computational fluid dynamics (CFD) is the science of predicting fluid flow, heat transfer, mass transfer, chemical reactions, and related phenomena by solving the mathematical equations which govern these processes using a numerical process.[**Bakker**] CFD is one of the most impactful uses of physics and computers in current century. Since the creation of the Navier-Stokes (N-S) equations in the 1820s, scientists, physicists, and engineers have tried to create faster and more efficient ways to model fluid flows to make more informed decisions and designs [**Bakker**]. Depending on the different assumptions one might make when creating a fluid simulation in a computer, the time it takes to calculate the final simulation or time series of a fluid flow can vary greatly from a matter of seconds for a RANs (Reynolds Averaged N-S) simulation, all the way to Direct Numerical Simulation (DNS) where turbulent features are fully resolved with an extremely high resolution model even for simple cases. For better reference see 3 to visually show how different fluid models resolve energy dissipation (viscosity) throughout the simulation.

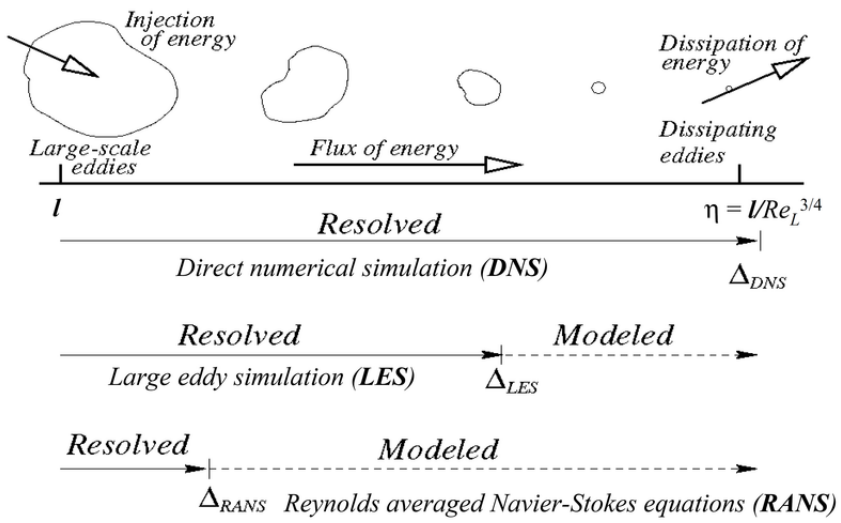

Figure 3: A picture of CFD model complexity versus reality

1.3 Physics Informed Neural Networks (PINNs) Overview

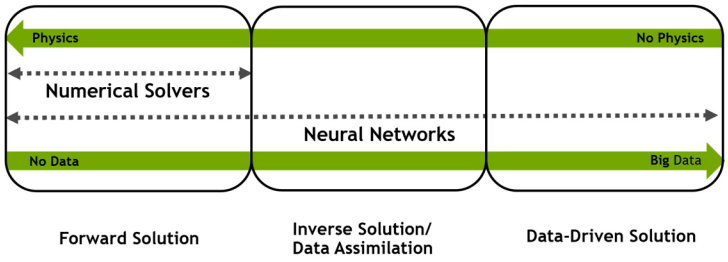

Figure 4: A picture of PINN spectrum from Nvidia *Modulus*© [**NVIDIA**]

Physics-informed machine learning debuted in the 1990s, with a few scattered publications throughout the decade, but did not gain traction until around 2010 when there was a resurgence in machine learning. The year 2011 introduced Google Brain, a cutting-edge platform capable of discovering and categorizing relatively simple objects (with a highlighted capability to identify cats.)[10] Another machine learning milestone was achieved in 2014 with Facebook's DeepFace Algorithm being able to detect whether two faces in unfamiliar photos were of the same person with 97.25% accuracy, a metric that effectively rivaled human perception.[10] By the late 2010s, increases in computational power had become more easily accessible to the average researcher, and thus, applications of machine learning to other scientific fields became increasingly prevalent.

In the past few years, physics-informed neural networks (also known as PINNs) have allowed scientists to tackle problems that were previously out of reach. It is important to recognize that PINNs differ from traditional neural networks in that PINNs use constraints bound by physical laws or equations. Instead of feeding random data points to the neural network, the input data is purposefully constrained according to the physical model that the data should satisfy [10]. For example, in 2020 within the field of drug discovery, a PINN-based model was built that embedded prior knowledge about molecular structure in a neural network to facilitate the identification of molecular water configurations with preferrable properties. [10] In particular, however, PINNs have had a significant impact within the field of computational fluid dynamics for their ability to tackle traditionally expensive problems. [**PINN˙History**]

Previous papers have by Raissi's group have demonstrated how PINNs can seamlessly integrate multifidelity/multi-modality experimental data with the various Navier–Stokes formulations for incompressible flows [**DBLP:journals/corr/abs**

DBLP:journals/corr/abs-1711-10566] as well as compressible flows [**physics-informed-machine-learning**] and biomedical fluid flows. Additionally, Raissi's group have introduced hidden physics models, which are essentially data-efficient learning machines capable of leveraging the underlying laws of physics, expressed by time dependent and nonlinear partial differential equations, to extract patterns from high-dimensional data generated from experiments. [10] This study builds upon Raissi's framework, and in order to reproduce results, we have substituted programming packages when necessary (i.e. using pytorch in lieu of tensorflow.)

2 Theory Overview

The general idea of this section is to give a general overview of the Navier-Stokes equations in a more applied sense, so a reader could have more context while not requiring an intimate knowledge of fluid mechanics to effectively analyze the results of the study. These equations are named after Claude-Louis Navier (1785-1836) and George Gabriel Stokes (1819-1903). The Navier-Stokes equations consist of a time-dependent continuity equation for conservation of mass, three time-dependent conservation of momentum equations, and time-dependent conservation of energy equation. There are four independent variables in the problem, the x, y, and z spatial coordinates of some domain and the time t.[**NASA**] They are second-order nonlinear partial differential equations, and in multiple simplified forms have known analytical solutions. ***In other words, the N-S equations describe (fully) how the velocity, pressure, temperature, and density of a moving fluid are related.*** An analytical solution of the full form would win someone a million dollars from the Millenium prize series.

To get started the "general" form of the N-S equations (in the convective form) will be presented:

Continuity Equation:

$$\nabla \cdot \vec{V} = 0 \tag{1}$$

Momentum Equation(s):

$$\rho \frac{D\vec{V}}{Dt} = -\nabla p + \rho \vec{g} + \mu \nabla^2 \vec{V} \tag{2}$$

This is the most concise full form (no assumptions) of the N-S equation, and the differential form with assumptions will be seen in the Methodology [3] section. In general terms, the continuity equation is a time-dependent way to account for the conservation of mass (back to Newton's laws of physics) such that no mass is created or destroyed in the area of observation these equations are being applied. The Momentum equations are time-dependent conservation of energy (ensuring no energy is created nor destroyed). Below is an illustration for the explanation of each term in the equation.

It is worth noting the "Convective Term" is simply a physical process that occurs in a flow of gas in which some property is transported by the ordered motion of the flow.[**NASA**] Additionally, the "velocity diffusion" term is a physical process that occurs in a flow of gas in which some property is transported by the random motion of the molecules of the gas. Diffusion is related to the stress tensor and to the viscosity of the gas. Turbulence, and the generation of boundary layers, are the result of diffusion in the flow.

Moreover, to solve a flow problem, you have to solve all five equations simultaneously; that is why we call this a ***coupled*** system of equations. There are actually some other equation that are required to solve this system. We only show five equations for six unknowns. An equation of state relates the pressure, temperature, and density of the gas. And we need to specify all of the terms of the stress tensor. In CFD the stress tensor terms are often approximated by a turbulence model. [**NASA**]

3 Methodology[Barba2019]

This study will implement the Cavity Flow with Navier–Stokes problem using a system of three differential equations. The first two equations are for the velocity components u, v. The third equation is for pressure p. Then using neural networks, the optimization function of a network will be replaced with a loss function form of the continuity equation in attempt to create a computationally efficient and faster model, while

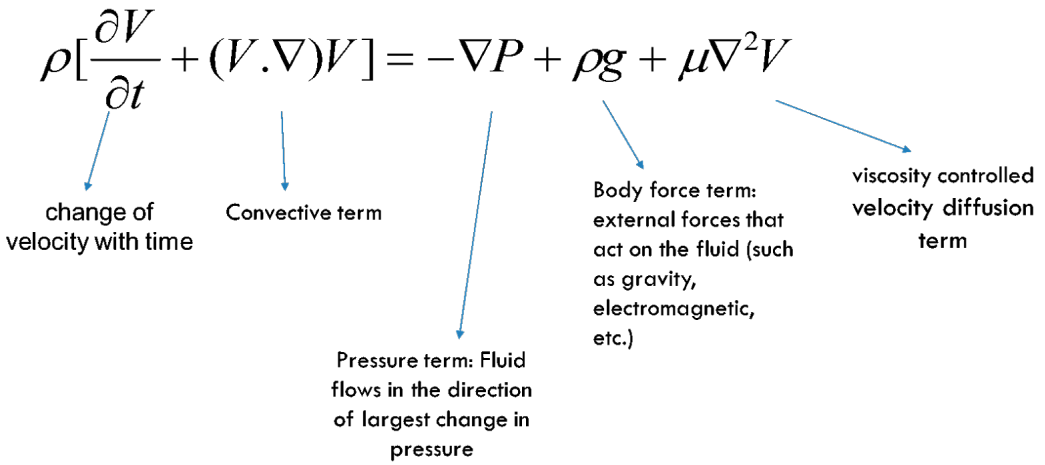

Figure 5: Navier-Stokes Explained[**sim˙world˙2014**]

fitting a model that adheres to physics. The derivation below is for an open lidded cavity problem where the assumptions are:

1. Two Dimensional (x and y)
2. Incompressible (ρ is a constant number)
3. Laminar, turbulence is minimal, the mean Reynolds number ≤ 2000

Now starting with the fundamental equations for this problem:

$$\begin{aligned}
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} &= -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \\
\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} &= -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \\
\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} &= -\rho\left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y}\right)
\end{aligned}$$

Each of these equations will then be Discretized as follows:

Discretization of the u-momentum equation:

$$\begin{aligned}
\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} + u_{i,j}^n\frac{u_{i,j}^n - u_{i-1,j}^n}{\Delta x} + v_{i,j}^n\frac{u_{i,j}^n - u_{i,j-1}^n}{\Delta y} = \\
-\frac{1}{\rho}\frac{p_{i+1,j}^n - p_{i-1,j}^n}{2\Delta x} + \nu\left(\frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{\Delta x^2} + \frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{\Delta y^2}\right) \quad (3)
\end{aligned}$$

Discretization of the v-momentum equation:

$$\begin{aligned}
\frac{v_{i,j}^{n+1} - v_{i,j}^n}{\Delta t} + u_{i,j}^n\frac{v_{i,j}^n - v_{i-1,j}^n}{\Delta x} + v_{i,j}^n\frac{v_{i,j}^n - v_{i,j-1}^n}{\Delta y} = \\
-\frac{1}{\rho}\frac{p_{i,j+1}^n - p_{i,j-1}^n}{2\Delta y} + \nu\left(\frac{v_{i+1,j}^n - 2v_{i,j}^n + v_{i-1,j}^n}{\Delta x^2} + \frac{v_{i,j+1}^n - 2v_{i,j}^n + v_{i,j-1}^n}{\Delta y^2}\right) \quad (4)
\end{aligned}$$

Discretization of the pressure-Poisson equation:

$$\begin{aligned}\frac{p^n_{i+1,j}-2p^n_{i,j}+p^n_{i-1,j}}{\Delta x^2}+\frac{p^n_{i,j+1}-2p^n_{i,j}+p^n_{i,j-1}}{\Delta y^2} = \\ \rho\Bigg[\frac{1}{\Delta t}\left(\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}+\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y}\right) \\ -\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x} \\ -2\frac{u_{i,j+1}-u_{i,j-1}}{2\Delta y}\frac{v_{i+1,j}-v_{i-1,j}}{2\Delta x} \\ -\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y}\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y}\Bigg]\end{aligned} \tag{5}$$

The discretized equations can once again be rearranged to convey how iterations will proceed in the code.

The momentum equation in the u direction:

$$\begin{aligned}u^{n+1}_{i,j} = \\ u^n_{i,j}-u^n_{i,j}\frac{\Delta t}{\Delta x}\left(u^n_{i,j}-u^n_{i-1,j}\right)-v^n_{i,j}\frac{\Delta t}{\Delta y}\left(u^n_{i,j}-u^n_{i,j-1}\right) \\ -\frac{\Delta t}{\rho 2\Delta x}\left(p^n_{i+1,j}-p^n_{i-1,j}\right) \\ +\nu\left(\frac{\Delta t}{\Delta x^2}\left(u^n_{i+1,j}-2u^n_{i,j}+u^n_{i-1,j}\right)+\frac{\Delta t}{\Delta y^2}\left(u^n_{i,j+1}-2u^n_{i,j}+u^n_{i,j-1}\right)\right)\end{aligned} \tag{6}$$

The momentum equation in the v direction:

$$\begin{aligned}v^{n+1}_{i,j} = \\ v^n_{i,j}-u^n_{i,j}\frac{\Delta t}{\Delta x}\left(v^n_{i,j}-v^n_{i-1,j}\right)-v^n_{i,j}\frac{\Delta t}{\Delta y}\left(v^n_{i,j}-v^n_{i,j-1}\right)) \\ -\frac{\Delta t}{\rho 2\Delta y}\left(p^n_{i,j+1}-p^n_{i,j-1}\right) \\ +\nu\left(\frac{\Delta t}{\Delta x^2}\left(v^n_{i+1,j}-2v^n_{i,j}+v^n_{i-1,j}\right)+\frac{\Delta t}{\Delta y^2}\left(v^n_{i,j+1}-2v^n_{i,j}+v^n_{i,j-1}\right)\right)\end{aligned} \tag{7}$$

Rearrangement of the pressure-Poisson equation:

$$\begin{aligned}p^n_{i,j} = \\ \frac{\left(p^n_{i+1,j}+p^n_{i-1,j}\right)\Delta y^2+\left(p^n_{i,j+1}+p^n_{i,j-1}\right)\Delta x^2}{2\left(\Delta x^2+\Delta y^2\right)} \\ -\frac{\rho\Delta x^2\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)} \\ \times\Bigg[\frac{1}{\Delta t}\left(\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}+\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y}\right)-\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x} \\ -2\frac{u_{i,j+1}-u_{i,j-1}}{2\Delta y}\frac{v_{i+1,j}-v_{i-1,j}}{2\Delta x}-\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y}\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y}\Bigg]\end{aligned} \tag{8}$$

This study will have the initial conditions $u, v, p = 0$ everywhere.

The boundary conditions will be:

- $u = 1$ **at** $y = 2$ **(the "lid");**
- $u, v = 0$ **on the other boundaries;**
- $\frac{\partial p}{\partial y} = 0$ **at** $y = 0$**;**
- $p = 0$ **at** $y = 2$
- $\frac{\partial p}{\partial x} = 0$ **at** $x = 0, 2$

3.1 PINN Optimization & Neural Network Layout

Lastly, this section will enumerate the optimization and loss function theory that makes a neural network a "PINN" (see PINN Overview Section 1.3). The following methodology is emulated from **DBLP:journals/corr/abs-1711-10** small alterations). To begin, this form of the Navier-Stokes takes a 2D equation with while studying percentage of noise, however, this noise input was not implemented in this study. The λ term below was still used to optimize the model to it's final solution. Additionally, the subscript denotes the derivative w.r.t. the variable, and the double subscript the second derivative.

$$\begin{aligned} u_t + \lambda_1(uu_x + vu_x) &= -p_x + \lambda_2(u_{xx} + u_{yy}) \\ v_t + \lambda_1(uv_x + vv_y) &= -p_y + \lambda_2(u_{xx} + u_{yy}) \end{aligned} \tag{9}$$

This is the Navier stokes simplified in two dimensions where $u(t, x, y)$ denotes the x-component of the velocity field, $v(t, x, y)$ the y-component, and finally $p(t, x, y)$ being the pressure. Additionally $\lambda = (\lambda_1, \lambda_2)$ are the unknowns of the function. Invoking inviscid fluid theory a typical assumptions are the following:

$$u = \psi_y \qquad\qquad v = -\psi_x \tag{10}$$

To invoke the continuity the following was used, where this study added in an additional latent term of the Reynolds number (Re, in attempt to embed additional characteristics of the fluid flow in the cavity):

$$\{t^i, x^i, y^i, u^i, v^i, Re^i\}_{i=1}^N \tag{11}$$

furthermore, **DBLP:journals/corr/abs-1711-10566** used the following functions to define the approximation $[\psi(t, x, y), p(t, x, y)]$

$$\begin{aligned} f(t, x, y) &\coloneqq u_t + \lambda_1(uu_x + vu_y) + p_x - \lambda_2(u_{xx} + u_{yy}) \\ g(t, x, y) &\coloneqq v_t + \lambda_1(uv_x + vv_y) + p_x - \lambda_2(v_{xx} + v_{yy}) \end{aligned} \tag{12}$$

Therefore the functions above are now the critical optimization parameter for the model to tune the network weights. While the mean squared error loss is now defined as:

$$MSE \coloneqq \frac{1}{N}\sum_{i=1}^{N}(|u(t^i, x^i, y^i) - u^i|^2 + |v(t^i, x^i, y^i) - v^i|^2) \tag{13}$$

$$+ \frac{1}{N}\sum_{i=1}^{N}(|f(t^i, x^i, y^i)|^2 + |g(t^i, x^i, y^i)|^2) \tag{14}$$

Lastly, the network used was just a simple array of fully connected layers with the input being [batch size x 4 x grid size^2] where for this study a grid of 151x151 in the x and y direction was used for the truthly data

of the *FlowPy* python model. This python code for computing the traditional lid driven cavity solution was fundamental.[**Barba2019**, **FlowPy**]

As seen in the table below, the network layout of the model is not complex, and there are only 3,103 total parameters to attempt to fit a robust model to.

BoxFlowNet

Feature batch shape: torch.Size([2, 22801, 4])		
Layer (type)	Output Shape	Param #
Linear-1	[-1, 2, 22801, 20]	100
Tanh-2	[-1, 2, 22801, 20]	0
Linear-3	[-1, 2, 22801, 20]	420
Tanh-4	[-1, 2, 22801, 20]	0
Linear-5	[-1, 2, 22801, 20]	420
Tanh-6	[-1, 2, 22801, 20]	0
Linear-7	[-1, 2, 22801, 20]	420
Tanh-8	[-1, 2, 22801, 20]	0
Linear-9	[-1, 2, 22801, 20]	420
Tanh-10	[-1, 2, 22801, 20]	0
Linear-11	[-1, 2, 22801, 20]	420
Tanh-12	[-1, 2, 22801, 20]	0
Linear-13	[-1, 2, 22801, 20]	420
Tanh-14	[-1, 2, 22801, 20]	0
Linear-15	[-1, 2, 22801, 20]	420
Tanh-16	[-1, 2, 22801, 20]	0
Linear-17	[-1, 2, 22801, 3]	63
Tanh-18	[-1, 2, 22801, 3]	0
Total params: 3,103 Trainable params: 3,103 Non-trainable params: 0		

Input size (MB): 0.70
Forward/backward pass size (MB): 113.42
Params size (MB): 0.01
Estimated Total Size (MB): 114.13

4 Evaluation and Final Results

Overall this was a really great learning experience, not just with PINNs, but with PyTorch, the basic building blocks of fluid mechanics, and creating robust pipelines that could be quickly reproduced to generate and train machine learned models. From the goal of learning valuable knowledge from a class project, this was a huge success regardless of the results. That being said, this section is going to go over the actual training results and visual outputs of the models along with a discussion of what could be improved for when this PINN is further investigated.

Before beginning, the first note is that the model didn't end up training to completion or convergence due to the speed of custom optimization and loss computations (loss plot and discussion below). Therefore these results should be seen as the "worst" a model of this size could produce, and again, is partially trained.

The total validation MSE is 975,651,365.54 for a more converged model, and 1,084,119,800.14 with a lesser converged model for 944 samples the model did not see before. This seems ridiculously high, and anyone would conclude from a metric standpoint the model isn't converged, or didn't learn anything at all! However since this model is generated to produce a truth-like replication of physics, the visual interpretation will

provide much more value as to whether there is just a bug in the code somewhere, the model needs some hyperparameter tuning/more fitting, or the model indeed, didn't learn anything in this case (likely due to user error rather than the method not being sound.) Nonetheless, moving on to testing the prediction values of a lower Reynolds number and a higher Reynolds number, and then take the difference so see where the model is predicting incorrectly. First let us observe contour and streamline plots of the truth data produced from *FlowPy* and note some key features that the model should also have.

FlowPy Output

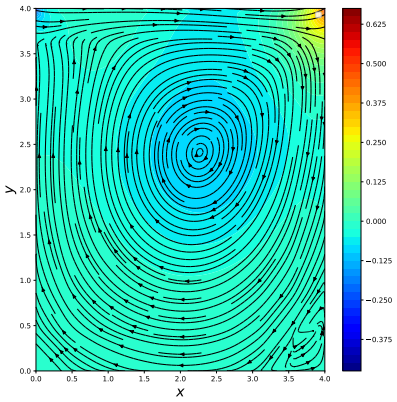

Figure 6: Streamlines (ψ)

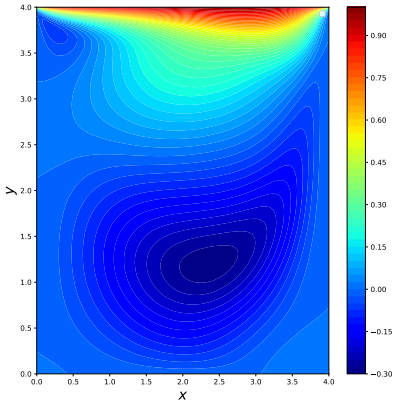

Figure 7: U Velocity $Re = 400$

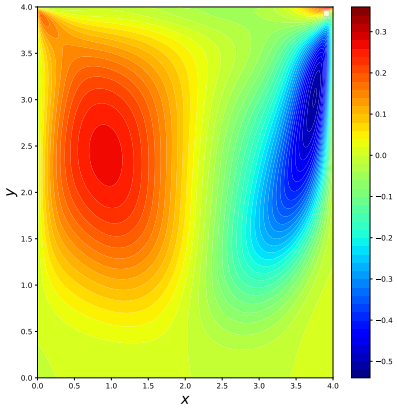

Figure 8: V Velocity $Re = 400$

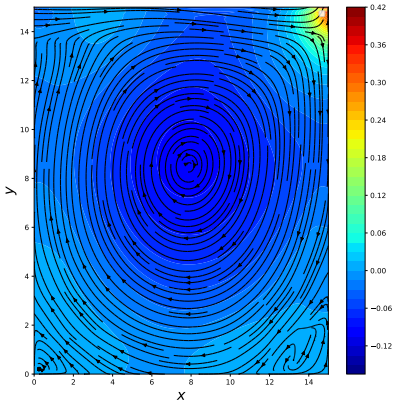

Figure 9: Streamlines (ψ)

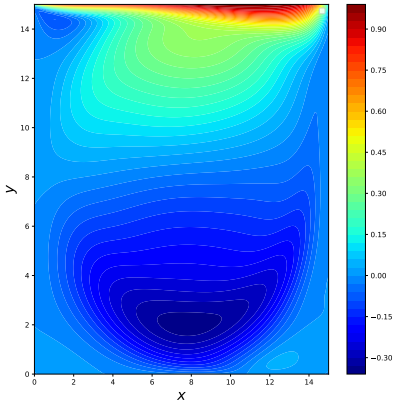

Figure 10: U Velocity $Re = 1500$

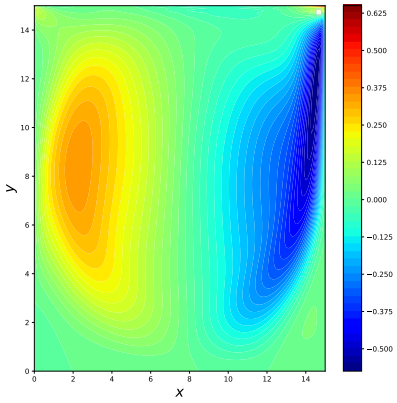

Figure 11: V Velocity $Re = 1500$

Overall the FlowPy results converged nicely, and usually take 5-10 minutes to converge depending on how long in time the model is run. There are many observations of note that could help determine the validity of BoxFlowNet and the prediction results. For the predictions to keep the observations concise only the difference of the U velocity will be observed. The list below are the features most noteworthy:

- Bottom left and right corners of Figure 6 and Figure 9 have smaller vortices, where the higher Reynolds number's bottom left corner is slightly more developed.
- Lower speed zones (darkest blue regions) in Figure 7 and Figure 10 move lower with respect to y the higher the speed of the fluid. This is due to a more energetic flow.
- V velocity high and low speed regions on the left and right of Figure 8 and Figure 11 have different shapes and sizes.

And now for the interesting part... or in this case, not so interesting! This was a great, really fascinating study, however it's very apparent that the model is either severely under-fit, or the model needs convolution

etc, to predict the spatial element. However it is worth nothing that the model isn't completely wrong, which leads me to conclude with the right layer tweaking this might be on to something. The reasoning behind this is the higher (or higher u velocity) is on the top, with a lower pressure on the bottom! A few things that come to mind with respect to what could be going on, is the resolution is simply too high, since the original code base for **DBLP:journals/corr/abs-1711-10566** used multiple optimizers, one being a specifically difficult to implement L-BFGS optimizer for the lambdas, and a regular Adam, while this model only used 2 parameter sets one with the weights and another with the lambdas. Another issue that might have caused this are the autograd calls to differentiation as seen in equation 12. Overall a very fun and interesting project, even if the model didn't fit very well and tweaking would be required.

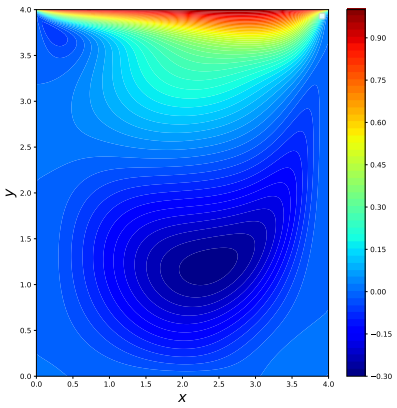

Figure 12: Original U (FlowPy)

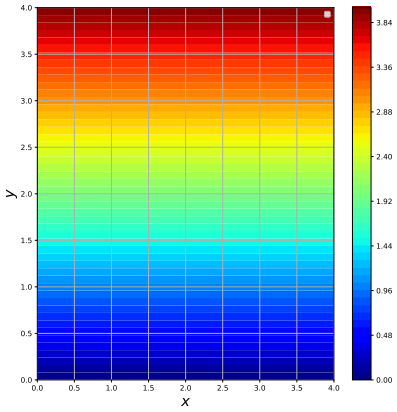

Figure 13: predicted U at $Re = 400$

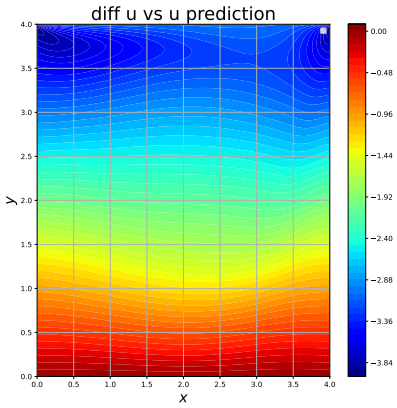

Figure 14: U minus U predicted

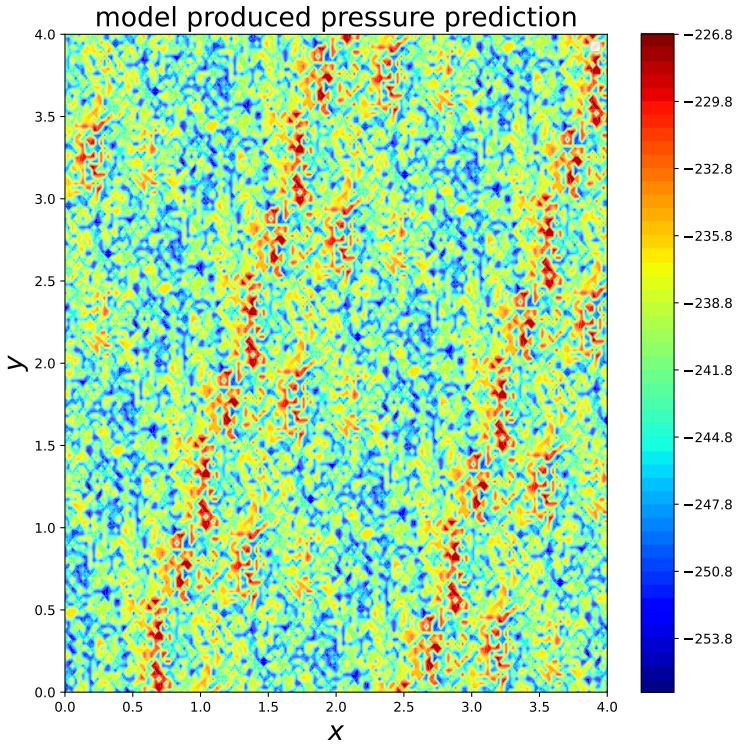

Figure 15: "pressure" (i.e. model weights)

$$
\begin{aligned}
a_t &= Ux_t + Wh_{t-1} + b \\
h_t &= \tanh(a_t) \\
o_t &= Vh_t + c \\
\hat{y}_t &= \text{softmax}(o_t) \\
L_t &= \text{CE}(\hat{y}_t, y_t)
\end{aligned} \tag{15}
$$

$$
\begin{aligned}
\frac{\partial L_t}{\partial U} &= \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial o_t}\frac{\partial o_t}{\partial h_t}\frac{\partial h_t}{\partial a_t}\frac{\partial a_t}{\partial U} \\
&= (\hat{y}_t - y_t)\frac{\partial \hat{y}_t}{\partial o_t}V\frac{\partial h_t}{\partial a_t}\frac{\partial a_t}{\partial U} \\
&= (\hat{y}_t - y_t)\frac{\partial \hat{y}_t}{\partial o_t}V\frac{\partial \tanh(a_t)}{\partial a_t}\frac{\partial a_t}{\partial U} \\
&= (\hat{y}_t - y_t)\frac{\partial \hat{y}_t}{\partial o_t}V\text{sech}^2(a_t)\frac{\partial a_t}{\partial U} \\
&= (\hat{y}_t - y_t)\frac{\partial \hat{y}_t}{\partial o_t}V\text{sech}^2(a_t)(x_t + h_{t-1})
\end{aligned} \tag{16}
$$

Code can be found here: https://github.gatech.edu/abartels3/Lid-Driven-Cavity-Flow-PINN

We have:

at = Uxt + Wht-1+b

ht = tanh(at) Of = Vht + c yt = Softmax(of) Lt = CE(yt, yt)

We want to compute:

dU = dL/dU dW = dL/dW db = dL/db dc = dL/dc dV = dL/dV

Using the chain rule, we have:

dL/dU = dL/dOf * dOf/dht * dht/dat * dat/dU dL/dW = dL/dOf * dOf/dht * dht/dat * dat/dW dL/db = dL/dat * dat/db dL/dc = dL/dOf * dOf/dc dL/dV = dL/dOf * dOf/dht * dht/dat * dat/dV

We can compute each of these derivatives as follows:

dL/dOf = olt

dOf/dht = V

dht/dat = f'(at)

dat/dU = xt

dat/dW = ht-1

dat/db = 1

dat/dV = ht

dat/dc = 1

Putting it all together, we have:

dU = olt * V * f'(at) * xt

dW = olt * V * f'(at) * ht-1

db = olt * V * f'(at) * 1

dc = olt * V * 1

dV = olt * ht

This is how we compute the gradients for a 4 length RNN.

Assuming a 4 length sequence RNN, we can compute the derivative of the Loss function with respect to the output of the RNN as follows:

dL/dOf = dL/d Lt * dLt/dOf

= (yt - yt) * Vht

= yt*Vht - yt*Vht

= yt*(Vht - Vht)

= 0

Therefore, the derivative of the Loss function with respect to the output of the RNN is 0.

Similarly, we can compute the derivative of the Loss function with respect to the hidden state of the RNN as follows:

dL/dht = dL/d Lt * dLt/dht
= (yt - yt) * Vht * dOf/dht
= (yt - yt) * Vht * Vht-1 * f'(ht-1)
= (yt - yt) * Vht * Vht-1 * (1 - f(ht-1)$\hat{2}$)
= (yt - yt) * Vht * Vht-1 * (1 - ht-1$\hat{2}$)
= yt*Vht*Vht-1*(1-ht-1$\hat{2}$) - yt*Vht*Vht-1*(1-ht-1$\hat{2}$)
= yt*Vht*Vht-1 - yt*Vht*Vht-1
= 0
Therefore, the derivative of the Loss function with respect to the hidden state of the RNN is also 0.