Cranfield UNIVERSITY

Applications in Practical High End Computing

Modelling of New RANS Viscous Model Equations with the Use of Model Discovery Libraries and Machine Learning

Group Project

Author(s):

Mr. Daniel James Rogers (s413363)

Mr. Syed Muhammad Gillani (s424669)

Mr. Adam Sohail (s425676)

Mr. Gabriel Maire (s432086)

Mr. Yasser El Karkouri (s430915)

Miss. Hyejoon Lee (s431846)

Mr. Venceslas Bignon (s432930)

Date:

Friday May 3^{rd}, 2024

Course:

Computational & Software Techniques in Engineering

Supervisor(s):

Dr. Irene Moulitsas

Dr. Jun Li

Dr. Tamás Jozsa

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

API Application Programming Interface

CFD Computational Fluid Dynamics

DNS Direct Numerical Simulation

FDM Finite Difference Method

FEM Finite Element Method

FVM Finite Volume Method

GUI Graphical User Interface

LES Large Eddy Simulation

RMSE Root Mean Squared Error

PDE Partial Differential Equation

PINN Physics Informed Neural Network

RANS Reynolds Averaged Navier Stokes

RNG Re-Normalisation Group

RST Reynolds Stress Tensors

RSM Reynolds Stress Model

SINDy Sparse Identification of Non-Linear Dynamics

SST Shear Stress Transport

URANS Unsteady Reynolds Average Navier Stokes

Nomenclature

μ	Dynamic Viscosity
ν	Kinematic Viscosity
ρ	Density
τ	Wall Shear Stress
Re_{tau}	Friction Reynolds Number
$u'u'$	Covariance of Normal Reynolds Stress in the streamwise direction
$u'v'$	Covariance of the Reynolds Stresses in the streamwise and wall-normal direction
$u'w'$	Covariance of the Reynolds Stresses in the streamwise and spanwise direction
u_{tau}	Friction velocity
$v'v'$	Covariance Normal Reynolds Stress in the wall-normal direction
$v'w'$	Covariance of the Reynolds Stresses in the wall-normal and spanwise direction
$w'w'$	Covariance Normal Reynolds Stress in the streamwise direction
y^+	Dimensionless Wall Distance
delta	Channel half width
dU/dy	stream wise velocity gradient in wall-normal direction
k	Turbulent kinetic energy
Lx	Streamwise domain size
Ly	Wall Normal Direction Size
Lz	Spanwise domain size
P	Mean profile of pressure
U_mean	Avg streamwise velocity
U	Mean profile of streamwise velocity
V	Mean Profile of wall-normal velocity

W Mean profile of spanwise velocity

y/delta Distance in the wall-normal direction, normalised by half the channel width

Abstract

This study introduces a novel framework that applies advanced data-driven techniques, including Sparse Identification of Non-Linear Systems (SINDy) and Physics-Informed Neural Networks (PINN), to predict Reynolds stresses in Poiseuille (Channel) and Couette unidirectional turbulent flows. Our approach involves constraining the PINN model with equations of motion derived using SINDy from a publicly available Direct Numerical Simulation (DNS) dataset. The models are trained using DNS data for Channel Flows across a range of Reynolds Numbers, including 182, 500, 2000, and 5200, and subsequently tested to gauge the performance on unseen data given a Reynolds Number of 1001. Our results demonstrate promising outcomes, with the model achieving a Root Mean Square Error of 0.0096 for Channel Flow and 0.016 for Couette Flow in the prediction of the normal stresses compared to the DNS results. Furthermore, the final model was incorporated into an automated Reynolds-Averaged Navier-Stokes (RANS) script, enhancing its capability to predict Reynolds stresses for unseen Reynolds cases. This modification resulted in a significant reduction in RMSE, achieving an improvement of 63.9% for the Reynolds 1001 Channel Flow and 54.6% for the 220 Couette Flow, both unseen test cases. This research contributes to advancing the predictive capabilities of turbulence modelling by including the increasing power of data-driven techniques. Our findings not only underscore the potential of SINDy and PINN in capturing complex turbulent phenomena but also highlight the steps for further exploration and application in practical engineering contexts with turbulence flow modelling.

1 Introduction

The fundamental equations governing the movement of fluids have been well-defined for quite a while, however, modelling turbulence is still a major engineering and challenge due to the physical nature of turbulent flow. Even though it is possible to resolve the smallest scales of turbulent flows by using Direct Numerical Simulations (DNS), it is not practical for real-world engineering applications. Ever since the efforts of Osborne Reynolds to develop a new approach to modelling turbulence by decomposing the flow field into a fluctuating term and a time-averaged term, different models to close this turbulence have been developed to accurately predict the flow field by maintaining a low computational cost. Most of the models relied on the Boussinesq Hypothesis, which related the new terms produced in the Reynolds Decomposition, known as Reynolds Stresses, with the velocities gradients. Such models were the k-epsilon model $(k-\epsilon)$ or $(k-\omega)$, however, after the publication of `https://arc.aiaa.org/doi/10.2514/3.12149` a new RANS turbulent model consisting of two equations named k omega Shear Stress Transport $(k-\omega(SST)$ became the industry standard.

Historically, turbulence closure models have relied on mathematical derivations, leading to dependencies on assumptions and simplifications that may not fully capture turbulent flow complexity. However, the emergence of machine learning and data-driven approaches has garnered interest due to their pattern recognition capabilities. These methods stand out in capturing intricate flow patterns and relationships without solely relying on traditional mathematical frameworks and assumptions. The incorporation, testing and validation of this new approach, required the implementation of different numerical techniques as well as data analysis tools such as interpolation, norm calculation and optimisation algorithms developed by the team to assess the requirements needed by each approach.

The pipeline, showcased in figure 1, followed throughout this project consisted of three different approaches. Firstly, to assess the difference between Reynolds-Averaged Navier-Stokes (RANS) and DNS predictions, an initial study was performed to compare current RANS models and the DNS results. Secondly, with the use of the DNS dataset, the Sparse Identification of Non-Linear Systems (SINDy) libraries were used to describe new governing momentum equations for both test cases. Thirdly, the PINN was constrained with both RANS and SINDy momentum equations; trained and validated with the DNS dataset, and compared with the RANS and DNS dataset.

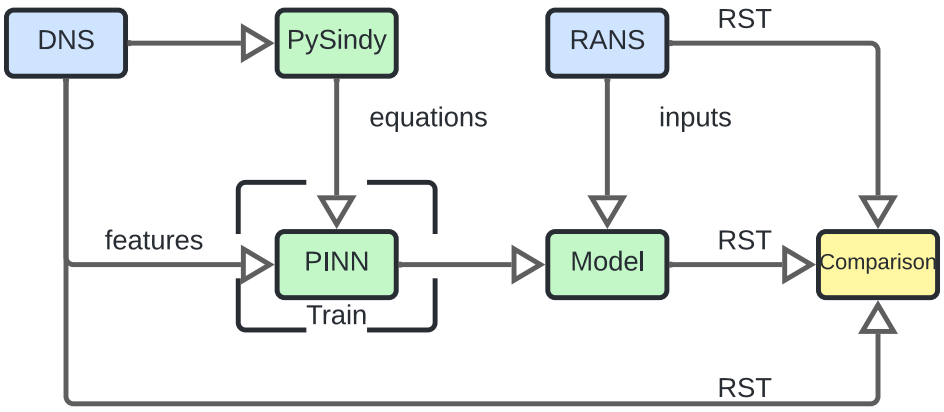

Figure 1: The Global Pipeline

Once the PINN model was finalised, it was combined with an automated Computational Fluid Dynamics (CFD) tool capable of computing both Channel and Couette flows with different Reynolds Numbers and exporting the inputs needed by PINN, to improve existing RANS results from simulations.

1.1 Literature Review

This project heavily relies on a DNS dataset, proportioned by Lee and Moser, from the University of Texas which can be found at: `https://turbulence.oden.utexas.edu/`. In our project, only the DNS data for the Channel Data (2015) and Couette Data (2018) will be used, which are accompanied by their respective papers used in the understanding of the data. Both Channel and Couette flow datasets were sourced by Myoungkyu Lee and Robert D. Moser [9] [8], whose papers serve as valuable references for the respective datasets utilised in our project.

The CFD part of the report was divided into three main subjects. Firstly, a great part of the literature was focused on understanding the Navier Stokes and Reynolds Average Navier Stokes (RANS) Equations, which was done with the use of Professor Alexander Smits's Viscous Flows and Turbulence Lectures Notes in Fluid Mechanics for the University of Princeton 20 [1]. Finally, to understand the background of RANS viscous models, walls functions and discretisation methods as well as their respective implementation in ANSYS Fluent, the online ANSYS FLUENT 12.0 Theory Guide [3], was heavily relied upon throughout this project.

The concept of Sparse Identification of Nonlinear Dynamical Systems (SINDy) is pivotal for understanding complex systems directly from data. Brunton et al. (2016) [10] introduced SINDy as a revolutionary data-driven framework that uses sparse regression to discover governing equations from high-dimensional data sets. The fundamental premise is that many physical systems' dynamics can be expressed by sparse combinations of basis functions, which lowers the complexity of the model while maintaining key dynamics.

Building on the foundational work of SINDy, the Python package PySINDy, initially developed by de Silva et al. (2020) [11], extends this methodology by incorporating advanced features that facilitate the application of SINDy to more complex systems. This includes the ability to handle control inputs, enforce physical constraints, and model partial differential equations (PDEs). PySINDy adheres to scikit-learn standards, making it accessible to a broad audience, including those with limited programming experience. This ease of use is coupled with robustness, as demonstrated in various examples that show PySINDy's effectiveness in identifying accurate dynamical models from noisy and incomplete data.

The recent updates to PySINDy by Kaptanoglu et al.(2022) [12] further enhance its capabilities by introducing methods to improve performance on real-world data. These updates include new optimization algorithms and methods for promoting model stability and handling inequality constraints. The software's expansion to accommodate a wider variety of system dynamics, such as actuated systems and implicit differential equations, alongside robust formulations like integral SINDy and ensemble techniques, marks significant progress in the field. PySINDy's comprehensive approach provides a powerful platform for researchers to perform automated model discovery, contributing to faster scientific insights and advancements

Researchers have employed SINDy to derive reduced-order models for turbulent systems, allowing for a simplified representation that retains key characteristics while reducing computational demands, such as R. Rubini's PhD Thesis [13].

Many researchers have demonstrated the method's capability to discover lower-dimensional governing

equations in various high-dimensional fluid dynamic systems, such as lock exchange and flow past one and two cylinders. This approach has facilitated the discovery of governing equations that describe turbulent behaviours, contributing to advancements in fluid dynamics and aerodynamics. Furthermore, SINDy's integration with machine learning has enhanced its capability to adapt to complex and noisy data, leading to more robust and accurate predictions. However, some areas still haven't been explored. One area for improvement is handling high-dimensionality and multi-scale turbulence, where SINDy may struggle to capture the complete range of dynamics. Increasing robustness to noise and uncertainty is another crucial focus, ensuring the derived models remain reliable.

Sparse system identification has been also used to determine the dynamics of 70 polynomial chaotic systems, effectively reproducing strange attractors and their fundamental dynamics [14]. This approach relies on extensive sub-sampling to generate large ensembles of models and uses the Akaike Information Criterion (AIC) to find optimal hyperparameter values. The computational speed of sparse linear regression allows for the computation of over one million dynamical models in just two days of CPU time, mainly spent during the MIOSR algorithm. This methodology has been used to benchmark various SINDy optimisers, revealing that STLSQ and MIOSR yield the best performance, while Lasso is more robust to noise but has generally lower performance. Large-scale fitting with SR3 requires additional hyperparameter scanning to achieve high performance. As noise increases, STLSQ's speed comes with some computational mistakes, whereas MIOSR slows down but maintains robustness. The study also indicates that the weak formulation can significantly improve performance, even without added noise, and can be used with any PySINDy optimiser, promoting its use in most cases. Additionally, sub-sampling the data to create an ensemble of SINDy models is recommended, especially in the presence of noise.

Turbulent flow, characterised by chaotic changes in velocity and pressure, poses significant challenges for modelling and simulation. Recent advancements in machine learning (ML) techniques have opened up new avenues for tackling these complex problems related to turbulent flow modelling.

One promising approach is the use of a Physics-Informed Neural Network (PINN), which incorporates underlying physical laws, such as the Navier-Stokes equations, directly into the neural network architecture. This allows for more accurate and physically consistent predictions of turbulent flow behaviour. Raissi et al. demonstrated the effectiveness of PINNs in modelling various turbulent flow phenomena, including Reynolds stress tensor prediction [15] and flow control [16]. Similarly, Mao et al. used PINNs to predict the evolution of turbulent channel flows [17].

Another area of active research is the application of deep learning architectures, such as Residual Networks (ResNets) and Deep Operator Networks (DeepONets), to turbulent flow simulations. Guastoni et al. utilized ResNets to perform temporal predictions of turbulent flow dynamics [18], while Zhu et al. employed DeepONets for spatial reconstructions of turbulent fields around high-Reynolds number aerofoils [19].

The use of unsupervised, semi-supervised, and supervised machine learning techniques has also been explored for turbulent flow analysis. Pandey et al. applied these methods to data from direct numerical simulations (DNS) of homogeneous isotropic turbulence, Rayleigh-Bénard convection, and minimal flow units in turbulent channel flows [20]. Iyer et al. studied the fractal properties of high-Reynolds number scalar turbulence using machine learning tools [21].

The integration of machine learning with traditional turbulence modelling approaches has also been investigated. Majchrzak et al. reviewed the use of random forests to model the Reynolds stress tensor [22],

while Fang et al. developed neural network models for the anisotropic Reynolds stress tensor in turbulent channel flow [23].

Beyond data-driven modelling, machine learning has been applied to flow control. Vignon et al. used deep reinforcement learning to reduce the Nusselt number in two-dimensional Rayleigh-Bénard convection [24], and Guastoni et al. demonstrated a complete framework based on multi-agent reinforcement learning for active flow control [25].

The literature reviewed highlights the potential of machine learning for accelerating and improving the simulation of turbulent flows. Obiols-Sales et al. developed a deep learning-based accelerator for fluid simulations [26], and Maulik et al. used machine learning to classify and blend sub-grid scale models [17]. Despite significant progress, several challenges remain in applying machine learning to turbulent flow modelling. These include the need for scalable algorithms that can handle the large datasets generated by high-fidelity simulations, as well as the development of interpretable models that can provide insights into the underlying physical mechanisms [28].

Overall, the integration of machine learning with turbulent flow modelling holds great promise for advancing the understanding and predictive capabilities in this field. As the field continues to evolve, further advancements can be expected in areas such as flow control, optimization, and the discovery of new physical phenomena through data-driven approaches [29].

1.2 Team Management

As this project aimed to integrate CFD, data-driven techniques and software development, one of the main challenges raised throughout its course was the combination and integration of several models carried out by different people with different approaches. Therefore, teamwork management became one of the cornerstones for successful completion of projects and tasks. Effective teamwork management has a heavy impact on every stage of the project, even before the project officially begins, as creating a well-balanced and technical team can have a great influence on the work dynamics and outputs. Furthermore, teams, by their nature, are expected to handle more complex tasks compared to individuals, requiring a high degree of collaboration. This section aims to explore the key elements that have defined this team as well as different methodologies and tools used to encourage collaboration and provide a quality output.

1.2.1 Team Formulation

When forming teams, one of the main focuses relies on properly assessing the strengths and weaknesses of each team member both in their personality and performance, as having a group with poor technical skills can have complications related to performance and outputs, but on the other hand groups with highly technical but individualistic members can lead to failure as the tasks presented usually cannot been handled alone. Moreover, diversity is also a quality looked at in teams as it plays a crucial role in obtaining a global perspective regarding specific issues raised during the course of the project. Additionally, people from different backgrounds and cultures provide different technical approaches as education and experience may differ from different parts of the globe.

When presented with the task of forming groups, people from three different options (Computational Intelligence for Data Analytics [CIDA], Computational Engineering Design [CED] and Software Engineering

for Technical Computing [SETC]) had to be integrated. Each group had to be formed with 4-5 people of the CIDA option, 1 member of the CED option and 1 member of the SETC option. Most of the CIDA team already knew eachother and in our case, the CIDA sub-team was formed by technical people who already knew their strengths and weaknesses beforehand. To complete the group, a search for technical and proactive members was done.

To assess the overall chemistry of the team several personality tests such as the *truity Personality Test*, as well as a tailored skills matrix and strengths and weaknesses forms were performed by all members. The results, showcased in figure 2, displayed a balanced team with tendencies to individualism comprised of highly technical individuals. Our team presented profiles with leadership personalities such as Daniel James Rogers, Yasser El Karkouri and Adam Sohail, as well as highly organised and planners individuals such as Hyejoon Lee and Venceslass Bignon and hardworking and effective members such as Gabriel Maire and Syed Muhammad Gillani.

	Strength	Weakness	Team Roles
Daniel James Rogers	Leadership & Direction	Conflict Resolution	**Chair** Self-confident, commands respect, good speaker, thinks positively, good at guiding the team. (Can be domineering, bossy.)
Syed Muhammad Gillani	Completing Documentation & Reports	Presenting ideas to customer, team, module leaders	**Finisher** Painstaking, conscientious, follows through and works hard to finish things properly. Meets deadlines and pays attention to detail. (Can be over-anxious and perfectionist.)
Adam Sohail	Suggesting ideas to the team	Planning &	**Innovator** Produces ideas, imaginative, unorthodox, radical, clever, uninhibited. (Can be over-sensitive, prickly. May need careful handling.)
Gabriel Maire	Presenting ideas to customer, team, module leaders	Organisation	**Evaluator** Careful, makes intelligent judgements, tests out ideas, evaluates proposals, helps the team avoid mistakes. (Can become isolated and aloof, pessimistic or over-critical.)
Yasser El Karkouri	Conflict Resolution	Leadership & Direction	**Teamworker** Sympathetic, understanding, sensitive, shows a strong concern for social interaction, leads from behind. Places the team above personal concerns. (May be indecisive.)
Hyejoon Lee	Helping the team to function (take notes, schedule meetings, book rooms)	Completing Tasks Efficiently	**Investigator** Finds things out, always knows someone who …, brings information back to the team, enthusiastic, gregarious. (Can be lazy and complacent.)
Venceslas Bignon	Programming, Planning Organisation	Suggesting ideas to the team	**Organiser** Methodical, hard-working, reliable, orthodox, turns ideas into plans which are feasible and gets down to tasks which need doing. (Can be inflexible and uninspiring.)

Figure 2: Team strength / weakness

1.2.2 Task Division, Allocation and Tracking and Project Outcomes

One of the particularities of the project was that a clear objective had not been defined initially, therefore it was left to each group to set up their own goal and pipeline to follow. In the first week the topic regarding turbulence modelling was introduced and even though the CED member had previous knowledge regarding CFD, he was not familiar with this topic therefore a collaborative effort between all members was made to gain more insight. One of the main tools used was a Visual Collaboration Online tool, called Miro Map 3, which allowed each member to contribute on a blank white canvas and make notations regarding all aspects of the project.

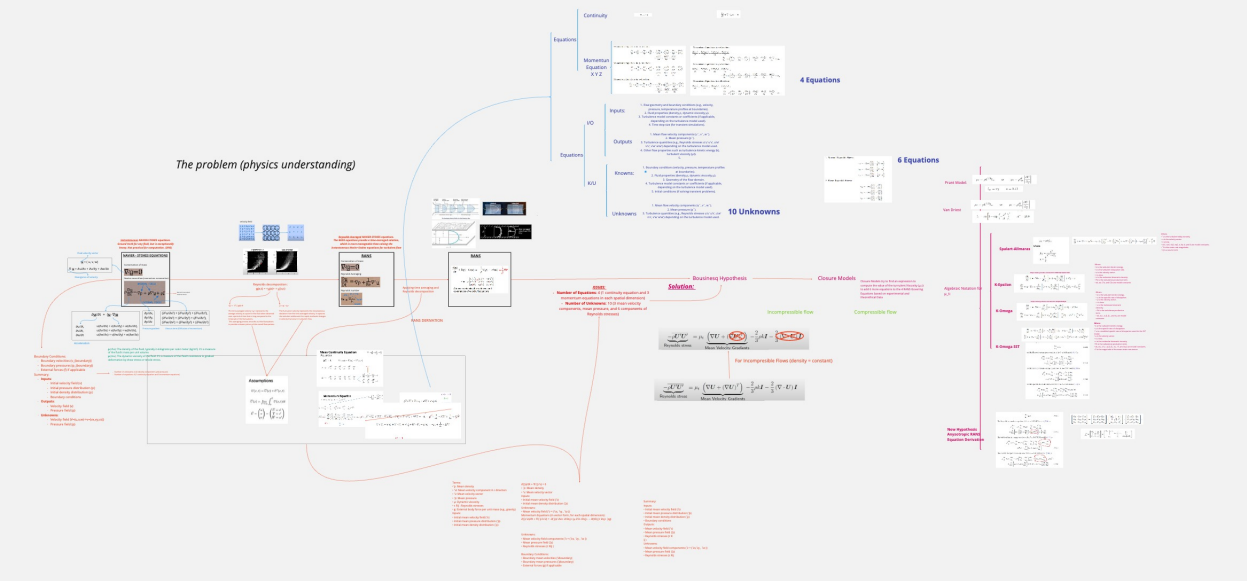

Figure 3: Miro Map

The first weeks were dedicated to conducting an extensive literature review to identify the latest trends in turbulence modelling as well as to assess the current state of the art on data-driven techniques, which helped us choose the best approach for prediction. After one week, the first stage of the literature review was concluded, the two approaches were selected, and a first pipeline was drafted. This led to a division of the team into four specialized groups to tackle each of the approaches defined. The first subgroup consisted of the CED members who focused on gaining experience in Turbulence Modelling stat of the art and CFD set-up. The second subgroup involved the SETC member which focused on retrieving the data and preprocessing it. Finally, the 5 CIDA members were divided into two subgroups one dedicated to physics-informed neural networks (PINNs), and the other to PySindy, a Python-based sparse identification library. The PySindy group focused on developing the PySindy model and pre-processing the data, while the PINN group worked on building, testing, and validating the PINN model. Both groups collaborated to derive new governing equations and refine the models for better outcomes. Initially, this led to individual progress derived from inexperience in group projects and underestimation of the results obtained through personality tests. However, as the group advanced people started relying more on each other and spontaneous meetings, which were advised via messages in the group chat, involving particular members began to occur more frequently as well as more updates and proactive attitudes were developed across the span of the project.

To set and track the Tasks, biweekly meetings were arranged, on Mondays after the supervisor's meeting and on Thursdays. Monday's meeting has the aim to review the supervisor's feedback, reevaluate the state of the project and define the tasks for the week, whereas Thursday's meetings to prepare the submission documents and track the tasks set on Monday. Meetings typically were driven by outspoken members with strong leadership skills, creating discussions and proposing new ideas. Meanwhile, members with a more analytical approach tend to challenge assumptions and offer a healthy dose of scepticism. A moderator kept things on track, consolidating various inputs and coordinating the next steps for the team. The responsibility for setting the meeting agenda and taking notes rotated among team members each week, with the most detail-oriented individual responsible for finalizing and submitting the minutes, however, Hyejoon was in charge of collecting and submitting all documents. The project was characterized by shifts

and modifications of the pipeline and goals after the meetings with the supervisors, where more insight and paths were provided. From the second to last week, the final pipeline (Figure 1) was derived, and efforts were put into integrating all work inside the same software. In the final weeks, each section was integrated under the guidance of a team member with experience as a graphic designer who compiled all the information into a clear and visually appealing presentation and poster highlighting the project's outcomes. Furthermore, a final model capable of enhancing results based on an automated RANS set-up tool was generated as well as a scientific paper and this report. Although the requirement of obtaining a fully functional code was achieved, some improvements such as full integration of both models, the implementation of a Graphic User Interface (GUI) and refined constraints were aimed but unable to be completed due to time constraints.

1.2.3 Project Management Tools

Given the complexity of project management, various tools and techniques were employed to track progress and maintain data integrity. Gantt charts were utilised to visualise project timelines and manage deadlines. This helps ensure that delays in one task do not cascade into other tasks, providing a clear overview of available spare time for contingencies. The tasks were divided based on individual roles, with deadlines set for each task. In each meeting, progress was reviewed to determine whether tasks were completed on time or delayed. (4). GitHub serves as a primary version control system, allowing us to back up data and maintain code continuity. This is particularly useful in case of local machine breakdowns, enabling us to revert to previous versions if needed. To document progress and discussions during weekly meetings, meeting minutes are recorded and stored on OneDrive (Appendix E1 and Appendix E2). Finally, communication channels such as WhatsApp and Teams were used to transfer information and hold meetings, which allowed people to work remotely and continue being productive.

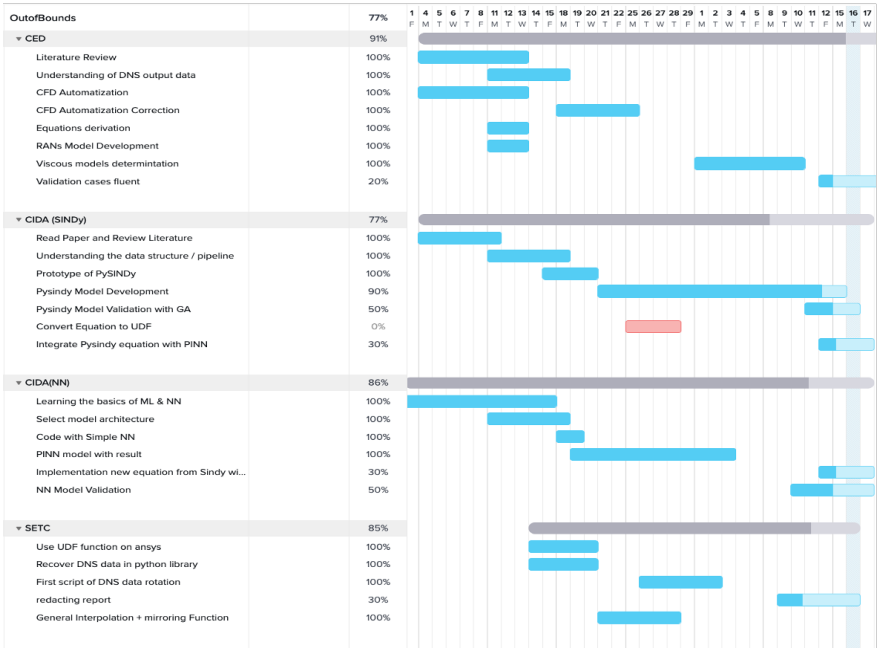

Figure 4: Team Management - Gantt Chart

2 State of The Art

2.1 Fundamental of CFD Equations and Turbulence Modelling

The foundation of CFD lies in the Navier-Stokes equations, formulated by Claude-Louis Navier and George Gabriel Stokes in the 19th century. These equations serve as the fundamental governing principles for fluid flow, encompassing the conservation of mass, momentum, and energy. In CFD, these equations are discretised and solved numerically within a computational domain divided into small cells or elements. Through iterative calculations, CFD simulates the behaviour of fluids, allowing engineers and scientists to predict complex flow patterns, turbulence, and heat transfer in various engineering applications. Despite their simplicity in form, the Navier-Stokes equations pose significant computational challenges due to their non-linearity and complexity. Therefore, the development of efficient numerical algorithms and computational techniques has been essential for the advancement of CFD, enabling its widespread application across the automotive, aerospace, and defence industries to just name a few. Through the integration of computational power and mathematical models, CFD continues to revolutionise the design and optimisation of engineering systems, driving innovation and efficiency in fluid dynamics analysis.

2.1.1 Navier-Stokes Equations

The Navier-Stokes equations are a set of equations that govern the motion and properties of viscous fluids. These set of equations with given initial conditions, boundary conditions and fluid properties, gives as an output the velocity, pressure and density fields of the fluid. These equations used the principles from the conservation of momentum of Newtonian fluids as well as Bernoulli's equation enabling the application to compressible and viscous flows. The Navier-Stokes Equations can be divided as follows:

2.1.1.1 Conservation of Mass Equation

The conservation of mass equations states that inside a control volume, the total mass of the fluid remains constant with time. This implies that for an incompressible fluid ($\rho = constant$) the divergence of the velocity across the domain must be 0, ensuring that there is no net variation (accumulation or reduction) of the mass across time. For incompressible fluids, equation (1) expresses conservation of mass or continuity:

$$\nabla \cdot v = 0 \tag{1}$$

2.1.1.2 Momentum's Equation

Before, explaining the momentum equation for Navier-Stokes, it is important to first mention Euler's Equation, or the momentum equation for inviscid fluids, fluids where the viscosity remains constant through time and space. Equation (2) shows Euler's Equation for incompressible fluids as:

$$\frac{D\mathbf{V}}{Dt} = -\frac{1}{\rho}\nabla p + \mathbf{g} \tag{2}$$

Taking into account the variance in the density, equation (3) is obtained:

$$\frac{\partial \mathbf{V}}{\partial t} + (\mathbf{V} \cdot \nabla)\mathbf{V} = -\frac{1}{\rho}\nabla p + \mathbf{g} \tag{3}$$

Finally, further taking into account the effects of viscosity, the final Navier-Stokes momentum equation can be obtained for both incompressible and compressible fluids, equation (4):

$$\frac{D\mathbf{V}}{Dt} = -\frac{1}{\rho}\nabla p + \mathbf{g} + \mu\nabla^2\mathbf{V} \tag{4}$$

2.1.2 Discretisation Methods

Once the Navier-Stokes equations are explained and derived (Appendix A), the next step is to understand the practical methods that CFD packages used to solve numerically these PDEs. Although the Navier-Stokes equations have a fundamental importance in the description of the physical properties that govern fluid motions, these equations have yet to be solved analytically, therefore to overcome this barrier, part of the CFD process involves an intermediate step where the domain is discretised or divided into smaller portions, often called cells. This step, which corresponds to the meshing step is essential for solving these equations numerically through an iterative process. In this subsection, three different discretisation schemes are analysed and their advantages and limitations are discussed.

2.1.2.1 Finite Difference Method

The Finite Difference Method (FDM) was the first discretisation method ever used and was proposed by Richardson in 1910 and subsequently implemented in CFD in 1933 for the first time ever in the study of flow over a cylinder. Richardson's method involves discretising the domain into grid points and the FDM equations are derived by approximating the derivatives of the PDEs using differences in function values at the points by using different discretisation schemes such as backward, forward, and central schemes. These derivatives are then approximated by truncated approximations of Taylor's Series expansions.

Some of the main advantages that this method provides are the simplicity in the derivation and implementation of the discrete equations as well as the possibility of obtaining accurate representations of high-order spatial approximations. However, some of the shortcomings lie in its inability to implement a model in complex or unstructured grids. Furthermore, there are no physical principles such as the conservation of momentum of mass applied in the generation of the new equations - this can lead to inaccurate representations of complex flow patterns.

2.1.2.2 Finite Volume Method

The Finite Volume Method (FVM) incorporates two main differences from the FDE which allows for solving two of the main disadvantages of the model. Firstly, FDE discretises the domain into grid points, which are used to obtain two the values of the approximated derivatives in each point, however, the FVM divides the domain into control volumes which are used to integrate the governing equations at each control volume, allowing the implementation of this method into more complex and unstructured meshes. Secondly, to

integrate the equations, some principles such as the conservation of mass, momentum and energy are used to take into account the variation in the fluxes around each control volume.

2.1.2.3 Finite Element Method

Finally, the Finite Element Method (FEM) differs from the FVE by taking into account other physical principles to derive the equations as well as the properties and structure of the control volumes that form the domain. Firstly, the equations in FVM are integrated across the control volumes, however, in FEM the equations are derived from shape functions which are defined by the number of nodal points across the control volume. These shape functions contain weighted parameters that are used to define alternative formulations of the governing equations (weak forms). Currently, this approach has been applied mostly in the field of Finite Element Analysis rather than CFD, however, this method is being implemented in complex simulations regarding the combination of structural analysis and fluid dynamics.

2.1.3 Turbulence Modelling

One of the main problems that mathematicians, physicists and engineers have been facing is the accurate prediction of turbulent behaviour of flows. Before diving into all the different turbulent models that have been developed and are currently being used, the concept of turbulence must be defined. Several physicists such as Leonardo Da Vinci and Osborne Reynolds conducted numerous experiments to determine the properties of laminar and turbulent flows. Laminar flows refer to fluids which flow in parallel layers with any disruption between them, enabling the generation of complex and chaotic fluid motions. On the contrary, turbulent flows refer to a chaotic flow where the fluid does not flow in parallel planes and complex structures called eddies, sometimes called vortices or even swirls, are generated. Turbulence is not a property that is only determined by the fluid's properties, but rather it is also determined by the conditions in which the fluid is propagated. On the one hand, laminar flows tend to form when the fluid moves at low velocities and encounters smooth surfaces. On the other hand, turbulence emerges once the velocity reaches a certain value, which depends on several factors, and the fluid enters a transitory phase where small vortices are generated until they fully develop into a turbulent flow. Once the flow becomes turbulent, the size of the eddies or vortices becomes chaotic, ranging from occupying the entire domain to being microscopic.

These phenomena were observed and studied by Reynolds in 1833 when he generated a small jet of water which flowed through a transparent pipe. During this experiment, Reynolds varied the velocity and geometry of the pipe and additionally dyed the colour of the jet stream to easily observe the different phenomena. At low speeds, the fluid had laminar properties and flowed without any disruption, however as the velocity of the stream increased, the fluid entered the transitory phase. Increasing the velocity of the stream further still revealed a chaotic phase in which the whole fluid became turbulent in its behaviour. A depiction of the full transition from laminar to turbulent is apparent in figure 5.

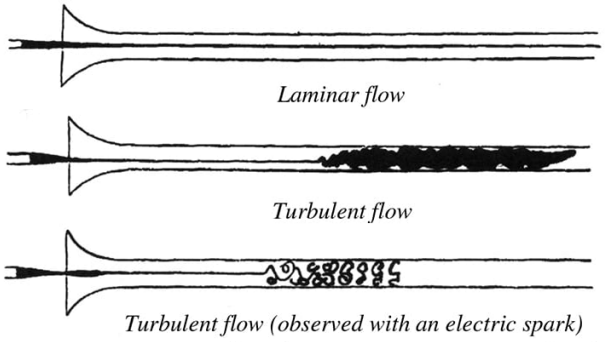

Figure 5: Reynolds's Experiment for Determining Laminar and Turbulent Flow [9]

This experiment enabled the definition of a new dimensionless number, called the Reynolds Number which allowed the estimation of whether a fluid was laminar or turbulent based on the geometry and the fluid's density, viscosity and velocity. Equation (5) captures this relationship.

$$Re = \frac{\rho v D}{\mu} \tag{5}$$

Depending on the value of the Reynolds number, flow types can be divided as follows:

- Laminar Flow: Below Reynolds numbers of 2300
- Transitory Flows: Between Reynolds Numbers of 2300 to 4000
- Turbulent Flows: Reynolds Numbers greater than 4000

The prediction and estimation of turbulent phenomena is one of the main challenges in the field of fluid mechanics in which physicists have been developing new models to predict turbulence as accurately as possible without a large computational demand. Nowadays, in order to model turbulent flow, there is an understandable trade-off between the complexity and accuracy of a model and the computational power and storage necessary for a simulation. The most popular models are illustrated in figure 7.

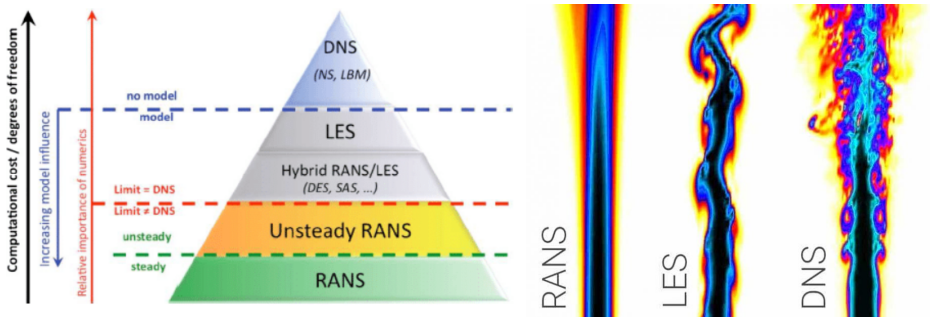

Figure 6: Different Turbulent Models [10]

2.1.3.1 Direct Numerical Simulation

Direct Numerical Simulation is the most accurate model used to predict turbulent properties in fluids and implements the full set of Navier Stokes equations without any approximation for modelling turbulence. This implementation is characterised by being able to compute vortices of all sizes across the domain, however, it comes with the cost of being heavily expensive in terms of computation.

2.1.3.2 Large Eddy Simulation

Large Eddy Simulation (LES) is the next step down from DNS, as it incorporates the full extent of the Navier Stokes equation to compute the vortices of larger sizes but implements additional turbulent models to predict the behaviour of smaller vortices. This in turn achieves more accurate results than if turbulent models were used across the whole domain, Despite this, LES modelling is still expensive to run, albeit to a lesser extent than DNS.

2.1.3.3 Steady and Unsteady Reynolds Average Navier-Stokes

The Steady and Unsteady Reynolds Average Navier-Stokes models (RANS and URANS) are the most widely adopted set of Navier-Stokes equations in both commercial and academic research due to producing accurate models, to a certain degree, with vastly more computational efficiency than DNS. The main difference between the RANS and URANS model is the temporal resolution. In the RANS model the field variables are averaged over time, which reduces the computational power required but this comes at the disadvantage of not capturing transient flow phenomena accurately. On the other hand, URANS models attempt to capture temporal variation in the flow by adding additional terms to the equations to represent the unsteady fluctuations. For the purpose of this project, only the RANS equations is used.

The RANS equations are obtained by averaging out all the field variables over time. This averaging comes from the idea of the Reynolds Decomposition, where the instantaneous variables are obtained by the sum of the averaged quantities and the fluctuation term. This decomposition applied to the velocity term is given by equation (6); however, it can also be applied to other flown variables, such as the pressure.

$$u_i = \bar{u}_i + u_i' \tag{6}$$

where:

u_i : instantaneous velocity

$\bar{u}_i = \dfrac{1}{\Delta t}\displaystyle\int_t^{t+\Delta t} u_i dt$: average term of velocity

u_i' : fluctuation term of velocity

Thee RANS set of equations are composed of 4 expressions: the continuity equation, equation (7) and the momentum equations in all three directions (X, Y and Z), equations (8), (9) and (10) respectively.

$$\nabla \cdot \mathbf{v} = 0 \tag{7}$$

X Momentum Equation

$$U\frac{\partial U}{\partial x} + V\frac{\partial U}{\partial y} + W\frac{\partial U}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial x} - \frac{\partial \overline{u'u'}}{\partial x} + \frac{\partial(\overline{u'v'})}{\partial y} + \frac{\partial(\overline{u'w'})}{\partial z} + \nu\nabla^2 U \tag{8}$$

Y Momentum Equation

$$U\frac{\partial V}{\partial x} + V\frac{\partial V}{\partial y} + W\frac{\partial V}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial y} - \frac{\partial(\overline{u'v'})}{\partial x} + \frac{\partial \overline{v'v'}}{\partial y} + \frac{\partial(\overline{v'w'})}{\partial z} + \nu\nabla^2 V \tag{9}$$

Z Momentum Equation

$$U\frac{\partial W}{\partial x} + V\frac{\partial W}{\partial y} + W\frac{\partial W}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial z} - \frac{\partial(\overline{u'w'})}{\partial x} + \frac{\partial(\overline{v'w'})}{\partial y} + \frac{\partial \overline{w'w'}}{\partial z} + \nu\nabla^2 W \tag{10}$$

Once the momentum equations are derived, in the left-hand side of the equations, some new terms emerge such as $\overline{u'u'}$, $\overline{u'v'}$, $\overline{u'w'}$, $\overline{v'v'}$, $\overline{v'w'}$, and $\overline{w'w'}$, which represent Reynolds Stresses. For example, in the X Momentum equation the term $\partial\overline{u'u'}/\partial x$ represents the contribution of the Reynolds Stress $\overline{u'u'}$ to the momentum flux in the x-direction. Similarly, the terms involving $\overline{u'v'}$ and $\overline{u'w'}$ represent the contributions of Reynolds Stresses in the y and z directions, respectively.

Physically, these Reynolds Stresses indicate the transfer in the momentum due to turbulent fluctuations. For example, $\overline{u'v'}$ represents the correlation between the fluctuations of velocity in the x and y directions. When $\overline{u'v'}$ is positive, it implies that high velocities in the x-direction are associated with high velocities in the y-direction. Conversely, negative values imply an anti-correlation, suggesting that high velocities in one direction are associated with low velocities in the other.

However, when trying to complete the RANS set of equations by correctly modelling the Reynolds Stresses, high-order terms appear making it impossible to correctly close the model, therefore not allowing the implementation of the current RANS equations in CFD. To be able to predict the Reynolds Stresses, closure models are necessary to be implemented to approximate the effects of turbulence on the mean flow field. These models involve adding additional equations for variables such as turbulent kinetic energy and its dissipation rate. The closure problem lies in the accurate formulation of these models, as they often rely on correlations or assumptions about the behaviour of turbulence.

The first and one of the most common closure models is the Boussinesq hypothesis, developed by Joseph Valentin Boussinesq in 1877, where he assumed a linear relation between the velocity gradients and the covariances of the Reynolds Stresses. This relation is defined by the eddy viscosity or turbulent viscosity. This model relies on local turbulent isotropy, which assumes that turbulent flow behaves the same way in all directions. The Boussinesq hypothesis is expressed in equation (11).

$$\tau_{ij}^R = -2\mu_T S_{ij} + \frac{2}{3}k\delta_{ij} \tag{11}$$

The term τ_{ij} refers to the Reynolds stresses as they are often expressed and they can be referred to as the Reynolds Stress Tensors (RST), equation(12)

$$\tau^R = \begin{bmatrix} u'u' & v'u' & w'u' \\ u'v' & v'v' & w'v' \\ u'w' & v'w' & w'w' \end{bmatrix} \tag{12}$$

Some of the most common RANS turbulent models use expressions to predict turbulent kinetic energy and or turbulent viscosity, to compute the Reynolds stresses. Some of the main closure models are the following:

Prandt Mixing Length

The Prandt Mixing Length model is one of the oldest and simplest turbulence models used to predict the Reynolds Stresses. This model assumes that turbulent viscosity is determined by the wall distance by incorporating a new parameter called the mixing length (l_m). Equation (13) explains this model:

$$\tau_{ij}^R = \rho l_m^2 \frac{\partial u_i}{\partial x_j} \tag{13}$$

This model is also called a one-equation model as it only uses one additional equation to close the RANS equations.

Spalart-Almaras Model

The Spalart-Almaras relies on only one additional transport equation to compute the turbulent viscosity (μ_T). This model was developed to be applied in aerospace applications, achieving good results in the computation phenomena around the boundary layer. Contrary to Prantd's model, this one-equation model does not require the calculation of the length of the scale. When applying the linear model for the computation of the Reynolds Stress Tensor, given by the Business Hypothesis (11), the term related to the normal stresses $2/3k\delta_{ij}$, is not taken into account as the model does not provide an estimation for k. There have been efforts in trying to create non-linear equations capable of predicting normal stresses.

K-Epsilon

The k-epsilon model is a two-equation turbulence closure that incorporates an equation that solves for the turbulent kinetic energy (k) and the dissipation rate (ϵ). This model is widely used across all the industry as it produces accurate results with computational efficiency. However, this model fails to predict accurately near-wall turbulent phenomena as the dissipation rate equation can't be computed at the wall, requiring the implementation of wall functions. Equation (14) describes this two-stage model.

$$\mu_T = f\left(\frac{\rho k^2}{\epsilon}\right) \tag{14}$$

The first proposition of the k-epsilon model by Launder and Spalding in 1974 was named the standard k-epsilon model. Future refinements of this model led to the creation of two additional k-epsilon models. Firstly, the RGN (Re-Normalisation Group) (RGN) k-epsilon model differed from the standard model by introducing a new extra term in the equation which allowed more accurate prediction in fluids with highly

sheared flows thus obtaining better results for low Reynolds flows. Secondly, the Realisable k-epsilon has a further improved formulation for the turbulent viscosity as well as a new transport equation. It earns its name by satisfying specific mathematical constraints on Reynolds stresses, aligning with turbulent flow physics; as the other two models do not satisfy this condition in the estimation of the Reynolds Stress Tensor. Some of those conditions are the:

- Positive values for the normal stresses $\overline{u_i^2} > 0$
- Schwarz' inequality for Reynolds shear stresses, where the scalar product of two stresses should be less or equal than the product of the individual magnitudes $\overline{u_i \cdot u_j}^2 \leq \overline{u_i^2 \cdot u_j^2}$

Standard K-Omega Model

The standard k-omega model is another two-equation closure which also solves for the turbulent kinetic energy and instead of solving for epsilon, it solves for another variable called the specific turbulent dissipation rate (ω). This model is characterised for having accurate prediction near the wall but fails to predict accurately turbulent phenomena far from the wall region. Equation (15) highlights it:

$$\mu_T = f\left(\frac{\rho k^2}{\omega}\right) \tag{15}$$

As a way to solve the problem regarding the inaccuracy at specific regions the k-epsilon and k-omega models had individually, the k-omega SST (Shear Stress Transport) combines the formulation of both equations by adding a blending function related to the normal distance to the wall, making it well suited for studying and predicting a great range of flows. The simplicity of this model compared to more complex models, such as the Reynolds Stress Transport, as well as its great accuracy near and far from the wall has made this the most used model across all the main commercial industries.

Reynolds Stress Model

The Reynolds Stress Model (RSM) or Reynolds stress equation model is the most complex RANS viscous model commercially used. Besides solving RANS continuity and momentum equations, it adds an equation for the dissipation rate and an extra equation for each term of the Reynolds stresses, adding up to a total of 7 equations for 3-dimensional flows.

This model often gives better results on flows with a high rotational component as the covariance of the stresses is solved individually, however, the predictions are highly constrained by the assumptions governing the RANS equations. Additionally, although it is more complex and more computationally demanding, the results obtained are not always more accurate than the ones obtained with one-equation or two-equation models, thriving on flows with high anisotropic components.

2.1.3.4 Wall and Near Wall Treatments

One of the main challenges that physicists and engineers face when developing models is accurately predicting near-wall phenomena, as a fully-developed turbulent flow profile highly differs from a laminar flow.

Before, understanding the law of the wall and wall treatments, the concept of the boundary layer

must be defined. The boundary layer accounts for the area formed around a surface for which a fluid is flowing around, due to the effects of a no-slip condition in which the fluid velocity is 0 at the point of contact with the surface. In laminar flows, due to the viscosity effects of the fluid, the layer of the fluid directly in contact with the wall comes to a stop, which affects subsequent layers in contact with each layer before them creating a velocity gradient until layers far from the wall achieve the streamline velocity. In turbulent flows however, this boundary layer is heavily affected by the generation of eddies, which heavily impacts the fluid propagation as well as increases its size and the overall skin drag. Figure 7 depicts the boundary layer and velocity gradient.

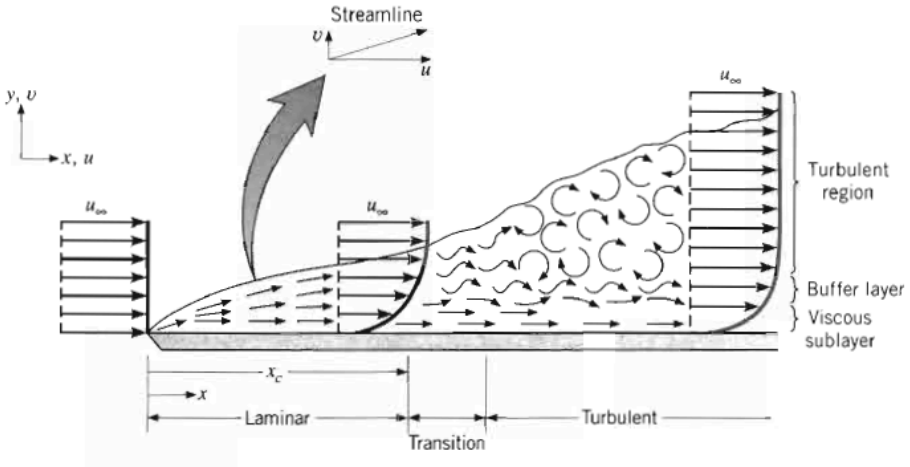

Figure 7: Velocity boundary Layer Development Over a Flat Plate [11]

For turbulent flow, one of the main tools used to predict the boundary layer is the law of the wall, which reduces the case by only taking into account the fluid conditions in the near-the-wall region, excluding the conditions further away. Some important parameters that are often used to study turbulent near-the-wall phenomena, and which are used to understand the DNS dataset are:

- The Wall Distance y
- Domain Length L
- The Mean Velocity U_{mean} or U_m.
- The Wall Shear Stress, τ_W
- The Fluid Density ρ.
- The Fluid Kinematic Viscosity ν or Dynamic Density μ - which are related through the following expression: $(\nu = \mu \cdot \rho)$

The wall shear stress quantifies the interaction between the fluid and the surface at the point of contact and represents the stress-generated tangential force exerted by the fluid in a unit area. Moreover, it is important to understand the magnitude of the friction velocity (u_T) which serves to quantify the magnitude of the stress as a velocity term which produces the same momentum transfer as the shear stress and it is given by

equation (16).

$$u_T = \sqrt{\frac{\tau_W}{\rho}} \tag{16}$$

This enables the definition of several dimensionless parameters useful in depicting turbulent wall phenomena such as:

- Dimensionless Wall Distance (y plus) y^*, given by equation (17).

$$y* = \frac{u_T \cdot y}{\nu} \tag{17}$$

- Dimensionless Velocity u^*, given by the equation (18)

$$u^* = \frac{U}{u_T} \tag{18}$$

By analysing the relationship between u^* and y^* and correlating with the experimental data, the turbulent boundary layer can be further divided into several layers as illustrated in figure 8

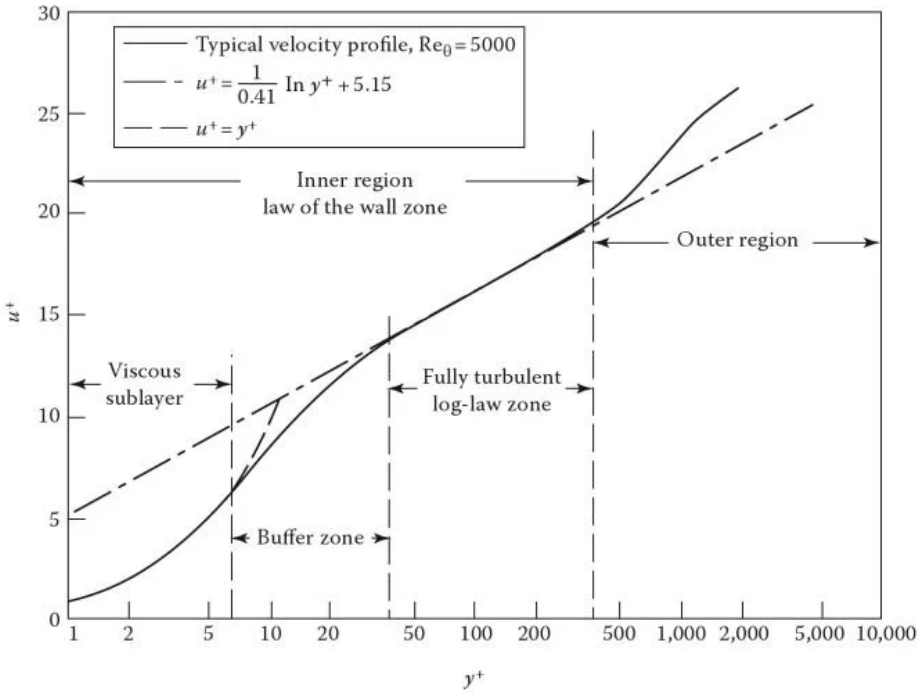

Figure 8: Turbulent Boundary Layer Velocity Profile Wall Zones [12]

Ideally, the first cell should lie in the viscous layer which lies between a $y*$ between 0 and 5 and allows the capturing of the velocity gradients in the closest wall region. To capture the turbulent phenomena it is crucial to ensure that the cells are also present in the following layers, including the transition or buffer

layer ($5 < y* < 30$) and turbulent or log-layer ($y* > 30$).

To resolve some of the turbulent phenomena in the boundary layer or near-the-wall zone, some models including the RSM and $k-\epsilon$ model require additional functions to correctly depict the field variables with physical sense.

The implementation of the wall function inside the Ansys Fluent framework is explained in the ANSYS FLUENT 12.0/12.1 Documentation, where it states that there are currently 3 types of wall functions, which are the Standard Wall Functions, the Non-Equilibrium Wall Functions and the Enhanced Wall Treatment.

Standard Wall Function

The standard wall function is the default option when working with $k-\epsilon$ models and uses an expression for computing the dimensionless velocity field in near the wall region and is given by equation (19), where k_a and E are contents modified to match the experimental data.

$$u^* = \frac{1}{k_a} * ln(E^*) \tag{19}$$

Depending on the layer in which each cell is located (which is given by the value of y^*) different expressions of the wall functions are applied to increase the accuracy of the results. This wall function, predicts correctly the dimensionless velocity profile in the region between y^* of 30 and 300.

Non-Equilibrium Wall Functions

The Non-equilibrium wall function has been used to replace the Standard Wall Function in more complex flows or non-uniform wall boundaries. It uses a new parameter called the turbulent velocity scale, which is obtained with a particular transport equation, rather than using the friction velocity.

Enhanced Wall Treatment

This new modelling is mainly applied to the Reynolds Stress Transport Equations to resolve the near-the-wall region with approximately y^* of 1 (viscous sub-layer), therefore one of the main requirements when using this model is to have a sufficient refined mesh so cells within this layer can be correctly modelled.

2.1.3.5 Computational Cost and CPU Time usage

Referring to Ansys Fluent Guide 5.1, chapter 10 (Modelling Turbulence), section 10.2.11 (Computational Effort: CPU Time and Solution Behaviour), some comparison regarding the computational cost of different viscous models was performed. Firstly, as expected the one-equation models (Spalart-Almaras) required less computational time than the standard 2 equation models and although efforts have been put into the optimisation of the RSM model, it was seen that it was between 50% and 60% more computational demanding than the latter ones and between 15% and 20% more expensive in terms of memory allocation. It was also seen that the standard $k-\epsilon$ and $k-\omega$ (both standard and SST) models required more or less the same time per iteration. However, the realisable and RNG models required more computational time, the latter one being between 10% and 15% more demanding. Finally, it was emphasised that the convergence

rate, which heavily impacts the total computation time varies between different models and problems, as well as to arrive at a converged solution, it will be necessary to start the solution with previously converged models.

2.1.3.6 Turbulent Flow in a Unidirectional Direction

The problem at hand entails the turbulent flow within a channel, as illustrated in Figure 1, focusing on the stream-wise (x-direction) and wall-normal (y-direction) axes. The channel comprises of two parallel plates separated by a distance of $2h$ where the flow of the fluid is orientated stream-wise. In figure 9, the flow velocity depends only on the y-coordinate, rendering the problem as one-dimensional. The mean velocity can therefore be expressed in equation (20):

$$U = (U, V, W) = (U_x, 0, 0) \tag{20}$$

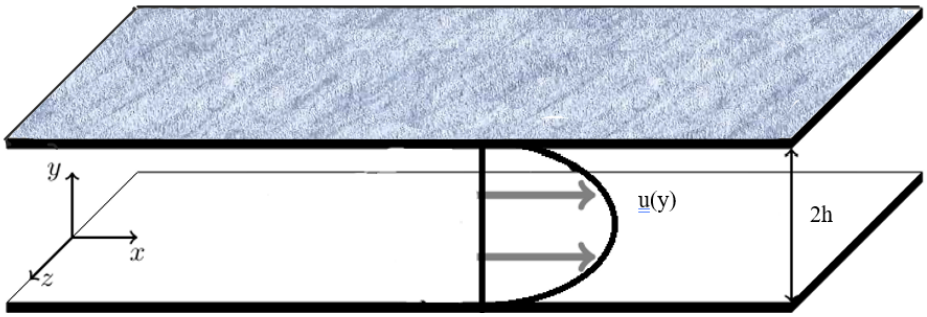

Figure 9: Turbulent flow in a Channel.

To address this problem, the Reynolds-Averaged Navier-Stokes (RANS) equations are used and these represent the temporal mean of the Navier-Stokes equations. The momentum conservation equation for a turbulent fluid flow in a channel, expressed in a vector is found by using equation (21):

$$U_j \frac{\partial U_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial P}{\partial x_j} + \nu_t \frac{\partial^2 U_i}{\partial x_j^2} - \frac{\partial \langle u_i' u_j' \rangle}{\partial x_j} \tag{21}$$

Focusing equation (21) on the x-direction, equation (22) is obtained:

$$U_x \frac{\partial U_x}{\partial x} + U_y \frac{\partial U_x}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \nu_t \left[\frac{\partial^2 U_x}{\partial x^2} + \frac{\partial^2 U_x}{\partial y^2}\right] - \frac{\partial u_x'^2}{\partial x} - \frac{\partial \langle u_x' u_y' \rangle}{\partial y} \tag{22}$$

With the assumption that there exists no variation in the x-direction apart from pressure, and the absence of U_y due to the exclusive presence of mean flow in the x-direction ($U = (U_x, 0, 0)$), equation (23) is arrived at:

$$-\frac{1}{\rho}\frac{\partial P}{\partial x} + \nu_t \frac{\partial^2 U_x}{\partial y^2} - \frac{\partial \langle u_x' u_y' \rangle}{\partial y} = 0 \tag{23}$$

Similarly, following analogous reasoning for the y-direction momentum equation, equation (24) is deduced:

$$-\frac{1}{\rho}\frac{\partial P}{\partial y} - \frac{\partial \langle u_y'^2 \rangle}{\partial y} = 0 \tag{24}$$

2.2 Machine Learning and Deep Learning

Machine learning is a field of artificial intelligence focused on developing algorithms and statistical models that enable systems to learn from data and make predictions or decisions without being explicitly programmed [30]. This involves training models on datasets to identify patterns and relationships, allowing models to make accurate predictions or decisions on new, unseen data.

Machine learning can be broadly categorised into three main types:

1. **Supervised Learning:**
    - This involves training models on labelled data, where the input data and corresponding output are provided [30].
    - The goal is to learn a mapping function that can accurately predict the output for new, unseen input data.
    - Examples include image classification, speech recognition, and regression tasks.
2. **Unsupervised Learning:**
    - This deals with finding patterns and relationships in unlabelled data [30].
    - The goal is to discover inherent structures, clusters, or associations within the data without relying on labeled outputs.
    - Examples include clustering algorithms, dimensionality reduction techniques, and anomaly detection.
3. **Reinforcement Learning:**
    - This involves an agent learning to make decisions by interacting with an environment and receiving rewards or penalties based on its actions [30].
    - The agent learns to maximise its cumulative reward by exploring different actions and observing their consequences.
    - Examples of this include game-playing agents, robotics control systems, and recommendation systems.

Deep learning, on the other hand, is a subset of machine learning that is inspired by the structure and function of the human brain, specifically the neural networks and the way that they interact with one another [31]. Deep learning models, such as artificial neural networks, comprise of multiple layers of interconnected nodes called neurons that can learn hierarchical representations of data, enabling them to capture more intricate patterns and relationships.

The practical applications of deep learning span across numerous domains, ranging from computer vision and natural language processing to healthcare and finance. For example, in computer vision, deep learning techniques like convolutional neural networks (CNNs) have achieved remarkable success in tasks such as image classification, object detection, and image segmentation. Similarly, in natural language processing, models like recurrent neural networks (RNNs) and transformers have revolutionised tasks like machine translation, sentiment analysis, and text generation [31].

2.2.1 Simple Neural Network

A neural network is a computational model that consists of interconnected nodes, called neurons, arranged into layers: an input layer, one or more hidden layers, and an output layer [32]. The architecture of a simple neural network is seen in figure 10.

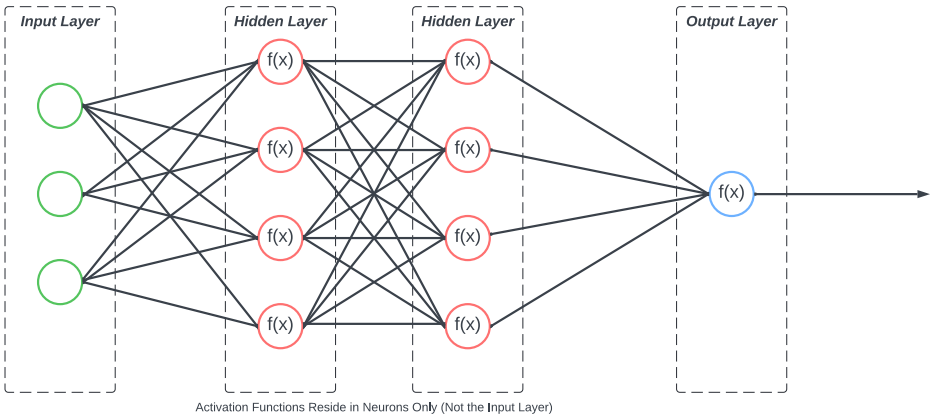

Figure 10: Simple Neural Network Architecture

The input layer receives the input data where each neuron in a hidden layer performs a weighted sum of the inputs from the previous layer and applies an activation function before passing the result to the next layer [32]. This process, known as forward propagation, continues until the output layer, where the final predictions or decisions are made. A detailed diagram of a neuron is illustrated in figure 11.

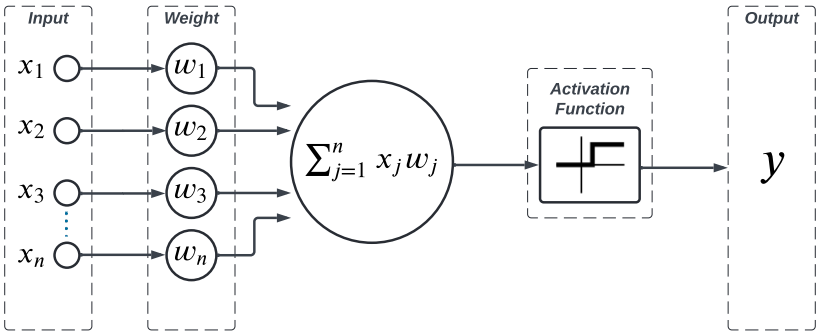

Figure 11: Neuron Logic

During training, the neural network adjusts its weights and biases through a process called back propagation [33]. Backpropagation involves calculating the gradients of the loss function with respect to the weights

and biases, and updating them using an optimisation algorithm, such as gradient descent, to minimise the loss. This iterative process allows the neural network to learn from the training data and improve its performance over time.

2.2.1.1 Keras Model

Keras is a high-level neural networks API `https://keras.io/`, written in Python and developed with a focus on enabling fast experimentation with deep neural networks. Keras allows a user to define and train neural network models given just a few lines of code. A Keras model is a way to define the architecture or topology of a neural network. It specifies the layers that make up the network and how they are connected. There are two main ways to define models in Keras, figure 12:

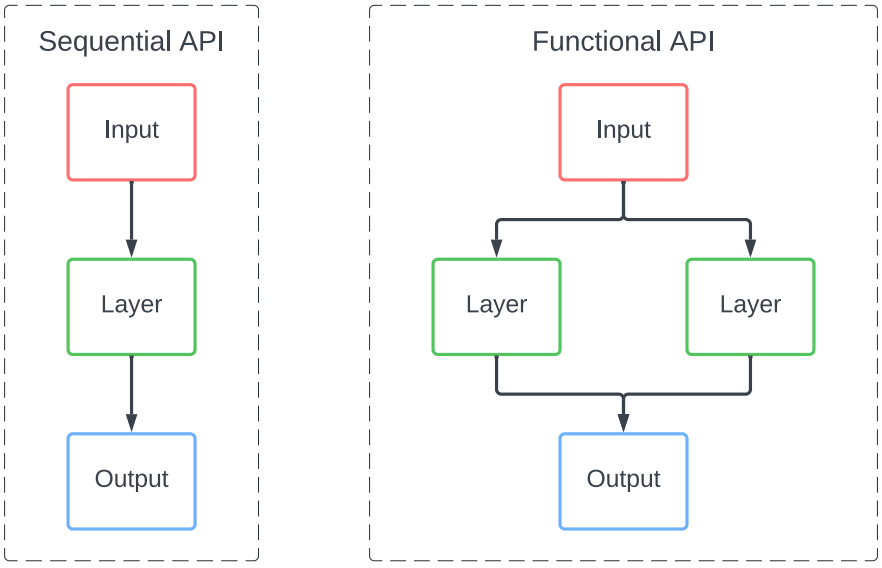

Figure 12: Keras Model Types

1. **The Sequential API Model:**
   - This is the simplest way to build a model in Keras.
   - It allows the creation of a linear stack of layers, where each layer has exactly one input tensor and one output tensor.
2. **The Functional API Model:**
   - This provides a more flexible way to define complex architectures, including models with shared layers or multiple inputs and or outputs.
   - The input tensors are defined and subsequently used to specify the connectivity of the output tensors.

2.2.2 Physics Informed Neural Network

The integration of Machine Learning techniques into domains such as CFD has proven challenging. This is primarily due to the complexity of CFD problems which demand significant time and computational resources for simulations aimed at understanding intricate flow patterns. Moreover, the traditional approach of relying solely on data for machine learning is not enough for such challenges, especially given the scarcity of available data in CFD applications and more importantly, the fundamental limitation of conventional machine learning lies in its lack of comprehension of underlying physical laws governing fluid dynamics phenomena. Consequently, achieving accurate and reliable results becomes a difficult task.

To address these challenges, new algorithms and models have emerged, offering promising solutions at the intersection of machine learning and CFD. One such groundbreaking approach is known as Physics-Informed Neural Networks (PINNs) [36]. This innovative solution enables the fusion of data-driven learning with the incorporation of fundamental physical principles.

This report delves into the application of the Physics-Informed Neural Network algorithm through two distinct methodologies. The initial approach involves leveraging simplified Reynolds-Averaged Navier-Stokes equations, while the second approach entails using the PySindy equations. Each of these approaches offers unique insights into the behaviour and performance of the PINN algorithm which will afterwards be compared.

As explained in this paper Raisis M. paper [37], the PINN algorithm requires a specific form, seen in equation (25):

$$u_t + N(u, \Lambda) = 0 \tag{25}$$

Where u corresponds to the unknown provided by the neural network and $N(u, \Lambda)$ represents a nonlinear function which incorporates the physical laws governing the problem being studied. Indeed, it encompasses both the (simplified) Navier-Stockes equations and the equations derived from SINDy.

However, the scenario of this study does not allow to have a temporal component due to the equations (RANS, SINDy) being time averaged as explained already. Consequently, the final general equation form which is used in the study is represented in equation (26):

$$N(u, \Lambda) = 0 \tag{26}$$

Given the two governing equations identified in Raisis M. paper [37], two continuous functions, f and g, are established in equations (27) and (28) respectively:

$$f := N_1(u'_x u'_y, \Lambda) \tag{27}$$

$$g := N_2(u'^2_y, \Lambda) \tag{28}$$

It is important to note that both of the outlined functions for f and g differ as the equations are different for RANS and SINDy based PINNs.

2.3 Sparse Identification of Non-Linear Dynamics

Sparse identification of nonlinear dynamics (SINDy), [10] is a data-driven approach that utilises available data, such as DNS results, to identify the underlying dynamics of a system. This approach allows for the discovery of patterns and relationships that may not be obvious through traditional theoretical methods. One of the primary advantages of SINDy is that it produces interpretable models. It provides equations that directly relate the system's variables to understand and analyse the system.

The PySINDy package is a Python library that provides basic functionalities for SINDy. It can be used to develop a RANS closure model and compare it with traditional RANS models to evaluate the potential of data-driven model discovery in capturing the complex dynamics of turbulence. Moreover, passing the SINDy-discovered model through a neural network can improve the model's predictive accuracy and robustness. The sparse regression technique requires available measured data that show a temporal history of the state variable vector $x(t)$ at multiple time instants. A time-differentiated data matrix is created by collecting the time series data of the time-differentiated value. A library of candidate functions is created that includes nonlinear terms of X, which can describe the system's dynamics. To solve the equation, sparse regression techniques are applied, with SINDy.differentiation determining the numerical differentiation scheme to approximate the derivative of the state vector, SINDy.feature_library selecting the candidate functions that compose the feature library, and SINDy.optimiser choosing the sparse regression algorithm to find the coefficient matrix.

3 Methodologies

3.1 DNS Data Used

As discussed in Section 2, the most accurate results achievable through numerical methods come from Direct Numerical Simulation (DNS), which involves the direct application of the Navier-Stokes equations; it resolves the turbulent phenomena at all spatial and temporal domain within the domain. For that reason, this data is usually taken as ground truth and it is the data that will be used to train all the data-driven models explained in the previous section.

The project primarily focuses on utilising databases from direct numerical simulations of turbulent flows obtained from `https://turbulence.oden.utexas.edu/`. Initially, it simulates a Couette flow and a channel flow in one dimension. Additionally, some simulations using the RANS set of equations were modelled and cases by replicating the same cases to provide a direct comparison with the new data obtained from the models to assess the improvement in the accuracy of the results.

The complexity in these datasets lies in the fact that they may not necessarily have the same variables from one simulation to another. The challenge is therefore to succeed in making correlations between the available data. For that reason, post-processing scripts were designed to transform the field variables that can be outputted from the CFD software to the same properties displayed in the dataset.

3.1.0.1 Test Cases

As has been stated, two different cases were used to obtain the data that will be used to train the models as well as directly compare to. The test cases that were launched correspond to the Poiseuille (Channel) Flow and the Couette flow through an open channel with two infinite parallel plates located at the top and the bottom. The main differences between the two cases rely on the initial or boundary conditions, as well as some specific changes in the set up between both DNS and RANS simulations.

Channel Flow

The Channel flow consists of a pressure-driven incompressible flow in a steady state. Originally, the Poiseuille flow was conceived to be a laminar flow travelling through a pipe where $R << L$, however in the current dataset that we are working with, the flow is described as flowing in a rectangular domain with periodic boundary conditions in the inlet, outlet, right and left sides and a no-slip wall condition at the top and bottom.

The flow is being driven by a pressure drop at the inlet and outlet achieving a steady state with a constant velocity. In order to set up the RANS set-up, the X-direction momentum source had to be determined by the use of the variables provided in the dataset, which corresponded with the kinematic viscosity (k), friction velocity (u_tau) and density (ρ). Currently, in the fluid, two forces govern the flow, these are the wall shear stress force and the pressure force:

$$Fw =< \tau > A_w \cdot 2$$
$$Fp = P_in_in - P_out \cdot A_oput = \Delta P \cdot A$$

To obtain the steady-state equilibrium, both the wall shear stress force and the pressure force must cancel each other. Therefore we can match and expand the two terms, obtaining the following expression:

$$Fw = Fp \rightarrow \Delta P \cdot A = \tau A_w \cdot 2 \rightarrow\rightarrow \Delta P \cdot Lx \cdot Lz = \tau Ly \cdot Lz \cdot 2 \rightarrow\rightarrow \frac{\Delta P}{Lx} = \frac{2 \cdot \tau}{Ly}$$

this expression, τ represents the wall shear stress which is defined by the product between the density (ρ) and the friction velocity squared (u^2_{tau}). By substituting this term in the expression we obtain the pressure gradient in the direction of the flow as equation (29) :

$$\frac{\Delta P}{Lx} = \frac{2 \cdot \rho \cdot u^2_{tau}}{Ly} \tag{29}$$

Couette Flow

The Couette Flow is characterised by being a shear drive flow rather than a pressure-driven flow. In this case, we are working with two parallel plates separated by a fixed distance which have different velocities. Although the case is usually defined by opposite velocities at the top and bottom wall, for example, -1 m/s

and 1 m/s, if we look at the x-Velocity component in DNS dataset in Figure 14, we can see that the initial velocity is at 0 m/s and the velocity at the channel half width is 1 m/s, therefore to replicate this case in the CFD software the velocity imposed at the bottom wall was 0 m/S and at the top wall 2 m/s.

3.1.0.2 Normalisation

Channel Flow Plots

After applying the mirroring and interpolation for the DNS Channel flow datasets we obtain the following plots, where the normalised velocity fields in the X and Z directions, all 6 covariances and the turbulent kinetic energy ($k = 0.5 * (u'u' + v'v' + w'w')$) are displayed through the full length of the channel:

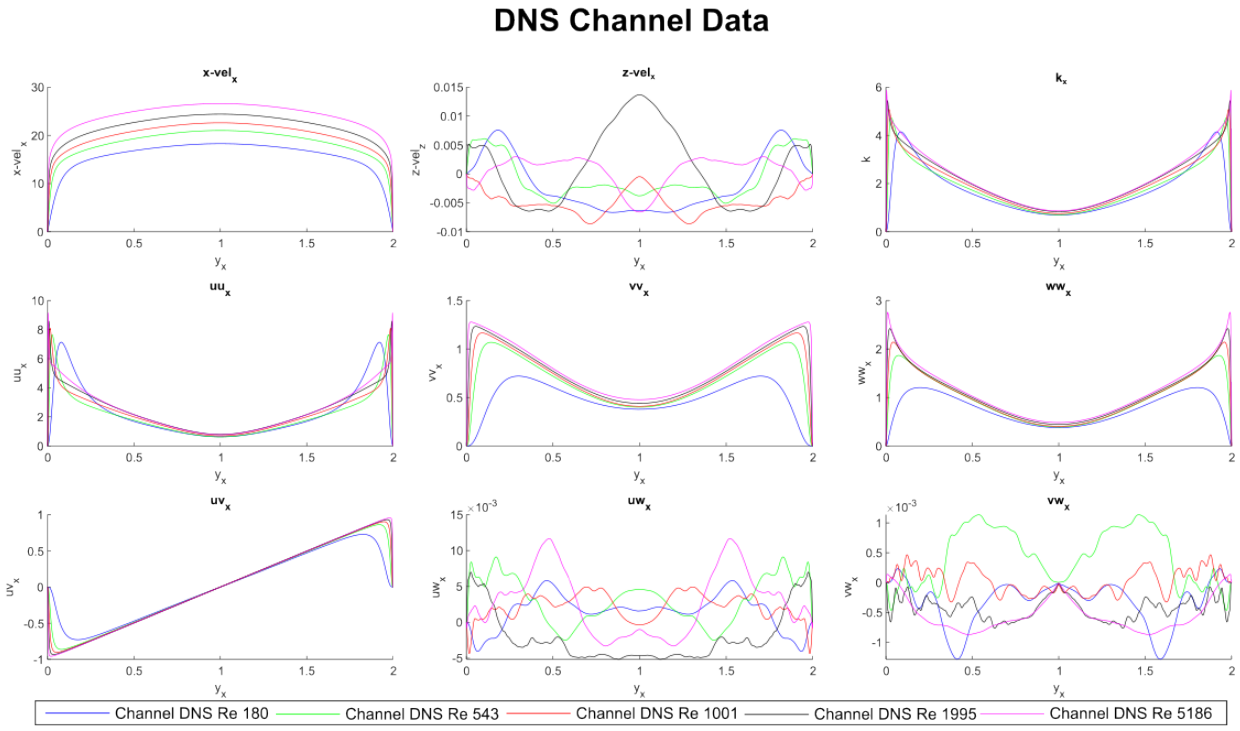

Figure 13: Channel Flow DNS Dataset Plots

From figure 13, we can see that the velocity in the Y and Z directions is negligible compared to the velocity field in the X direction. This also applies to the Reynolds Stresses where the covariance for X and Z (u'w'), and Y and Z are negligible compared (v'w'). By focusing on the near-the-wall region and by reducing the axis to the last 10% of the domain, we can see that all non-zero variables have the same properties, characterised by a first value of 0 both at the start and end of the domain and as a well as a steep gradient near the wall. If we focus on the evolution of the curves for each Friction Reynolds Number, we can observe that the maximum value as well as the gradients becomes greater as the Reynolds number increases. From the velocity in the X direction, we can see that although the normalised averaged velocity was 1 for all 5 cases, the velocity profile was not the same in all the cases because each case had its specific value of friction velocity.

Couette Flow Plots

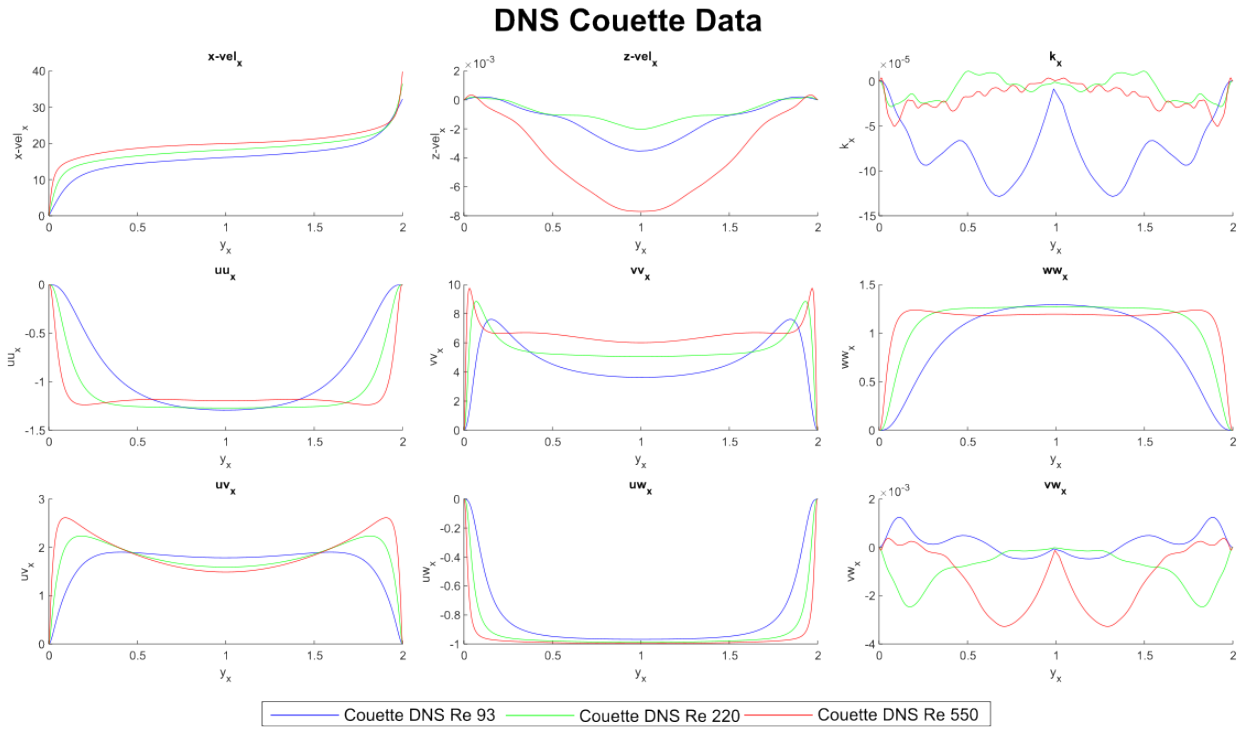

Figure 14: Couette Flow DNS Dataset Velocity Profiles

3.1.1 Data Structure

In order to use the datasets in the various programs that may be designed, a script has been created to read the different file formats of the datasets. This data was stored in DataSet objects containing other objects corresponding to the file and existing variables - this is mapped out in figure 15:

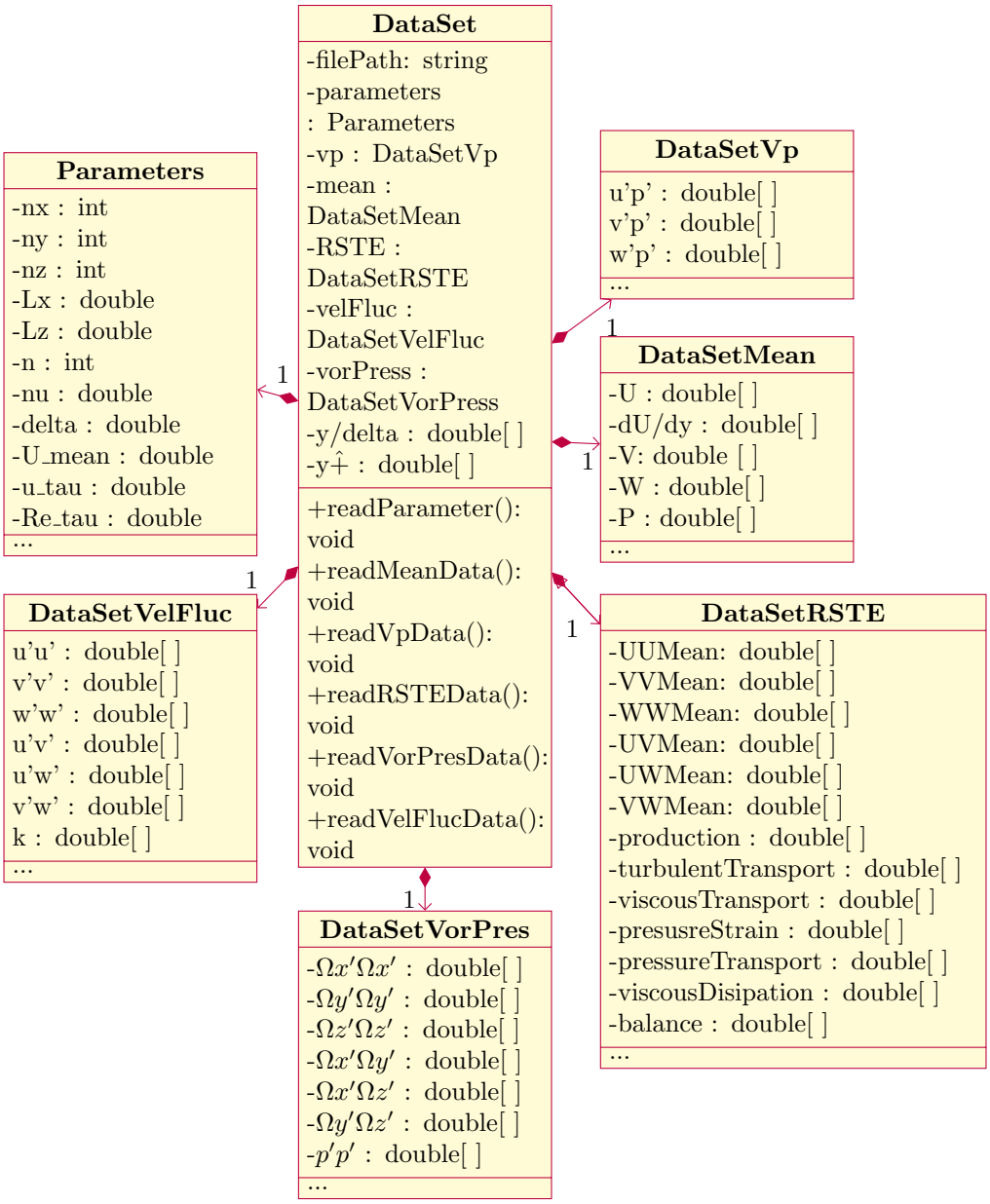

Figure 15: DNS DataSet Class Diagram for Couette & Channel

The description of useful attributes shown in figure 15 are given in the nomenclature.

3.1.1.1 Test Cases

To produce robust and quality code, it is important to test it. The test cases go as follows:

- Read parameter from utexas Channel files.
- Read parameter from utexas Couette files.

- Read parameter from hill simulation vtr format.
- Read mean data from utexas Couette file.
- Read mean data from hill simulation vtr format.
- Read mean data from hill simulation dat format.
- Read RSTE data from utexas Couette file.
- Read RSTE data from hill simulation vtr format.
- Read variances and covariance of velocity components from utexas datasets.
- Read velocity pressure correlation from utexas datasets.
- Read variances of vorticity components and pressure from utexas datasets.

3.1.2 Data Comparison

During this project two comparative approaches have been used to assess the quality of the outputs provided by each approach. Firstly, qualitative comparisons have been made by plotting two or more outputs against it. Most of the plots have been comparing the values against the normalised channel length (y^+) or unnormalised channel length (y), however, some plots located in the RANS comparisons section required logarithmic scales to properly analyse the near-the-wall region.

For the quantitative approach, two approaches were used. Firstly, the PySINDY utilised the R-squared error which was imported from $from sklearn.metrics import r2\_score$

$$r2\_train = r2_score(X\_dot\_train\_combined, predicted\_dot\_train)$$

$$r2\_test = r2\_score(X\_dot\_test\_combined, predicted\_dot\_test)$$

To evaluate the R square score, the $sklearn.metrics$ $r2\_score(ground\_truth, prediction)$ method was applied to evaluate both the R square score of our training results for evaluation and of our testing results for validation. On the other hand, both PINN and RANS errors were assessed with the Root Mean Squared Error (RMSE), which reduces the discrepancies between two models to a single value by the use of equation (30).

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{30}$$

Where:

n is the number of data points,
y_i represents the observed values,
$\hat{y}_i$ represents the predicted values.

The computations of this error required all values to have the same distribution of points, and because RANS and DNS were obtained with different meshes, therefore the values did not match on the same y-coordinate. This highlighted the requirement of Interpolation algorithms.

3.1.2.1 Data Interpolation

As the data may not have the same discretisation, it is important to have the same x-coordinates in order to quantify the differences in outputs.

The main idea behind interpolation is to find a function that passes exactly through a set of known data points. These points, often referred to as interpolation points, are discrete data that represent a certain relationship between two variables. The raw DNS data

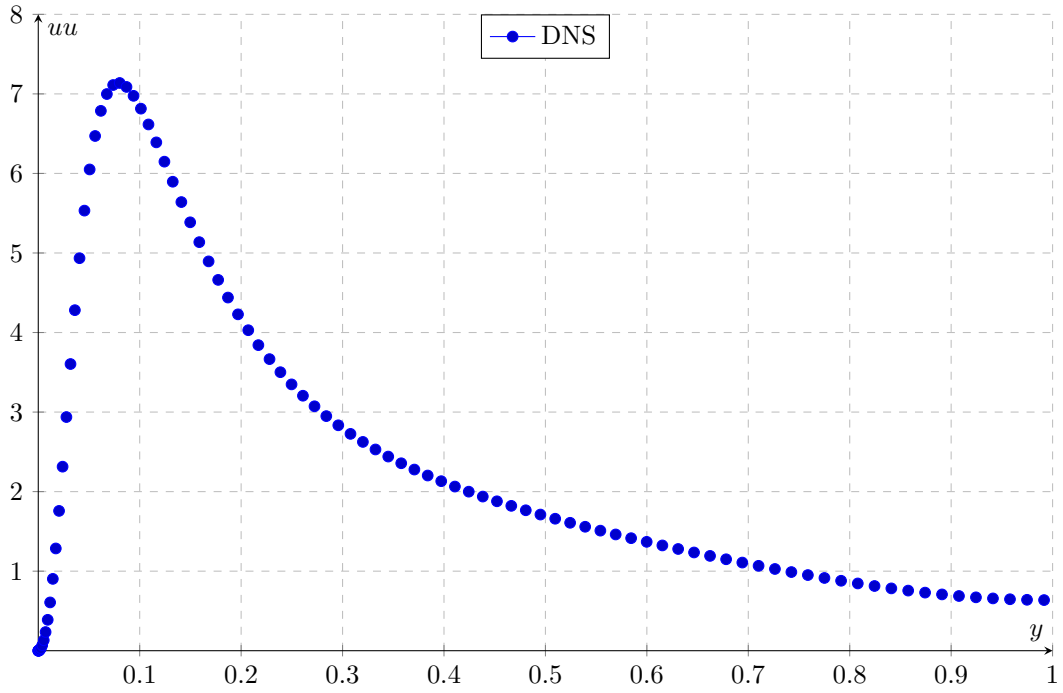

Figure 16: DNS Data

The goal of interpolation is to fill in the gaps between these data points by estimating intermediate values. This generates a smooth curve or continuous function that best represents the available data. The interpolation is done using the scipy library `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html`. We use cubic spline interpolation to achieve the highest level of accuracy possible and the idea is also to control the number of points we want. Each interval is represented by a cubic function, equation (31).

$$C_i(x) = a_i + b_i x + c_i x^2 + d_i x^3; \quad i = 1, ..., n \quad [34] \tag{31}$$

It means that it is necessary to determine each a_i, b_i, c_i, d_1. This is resolved by a system of equation of 4 elements with the boundary condition named "not-a-knot spline" which means that $C_1'''(x_1) = C_2'''(x_1)$ and

$C'''_{n-1}(x_{n-1}) = C'''_n(x_{n-1})$ [35] by default.

```
def interpolate(startPoint : float,
                endPoint : float,
                x : np.array,
                y : np.array,
                num : int) -> tuple[np.array,np.array]:
        # create the grid based on the num chosen
        x_new = np.linspace(startPoint, endPoint, num)
        interpolating_function = interp1d(x,y,kind="cubic")
        return (x_new,interpolating_function(x_new))
```

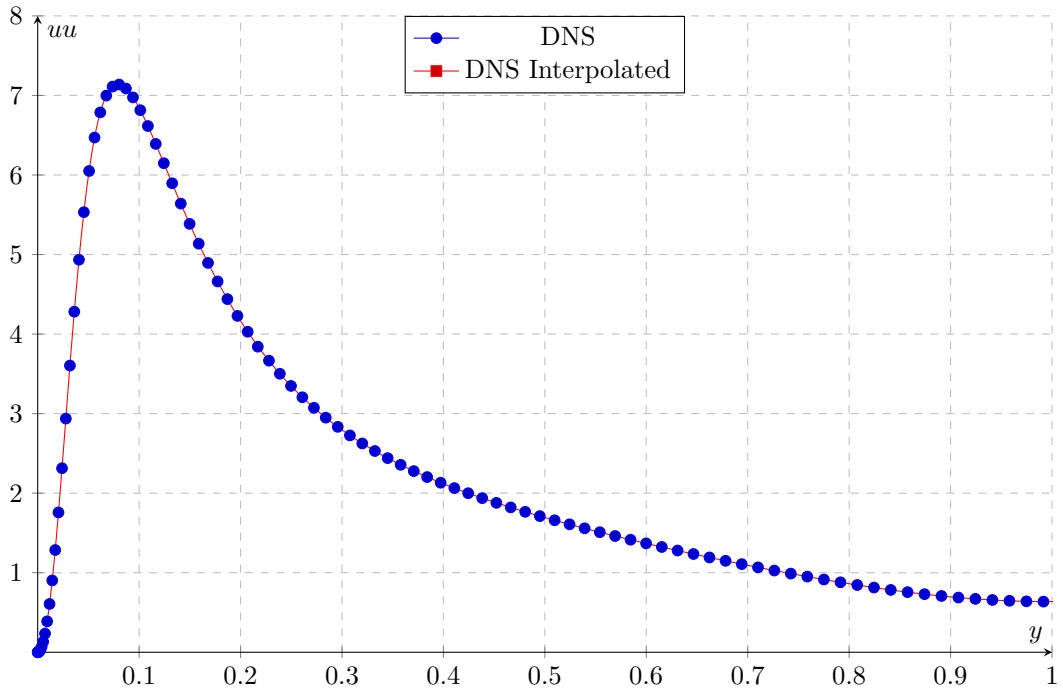

Figure 17: DNS Data Interpolated

The figure 17 show that the interpolated curve match really well the data, as it has a simple shape it is normal that a cubic spline follow perfectly the trend of the data. In the end, this function will be used to enable the comparison of the models to the reference, which are the DNS data.

3.1.2.2 Testing Data interpolation

The interpolation function has been tested on the following polynomial functions:

$Quadratic : f(x) = x^2$
$Cubic : f(x) = \frac{1}{7}x^3 + x^2 + x - 1$
$Quartic : f(x) = -5x^4 + 7x^3 + 3x^2 - 3x + 3$
$Quintic : f(x) = 0.988x^5 - 4.96x^4 + 4.978x^3 + 5.015x^2 - 6.043x - 1$

$Sextic : f(x) = x^6 + x^5 - 13x^4 - 0.9x^3 + 34x^2 - 14.5x - 5$

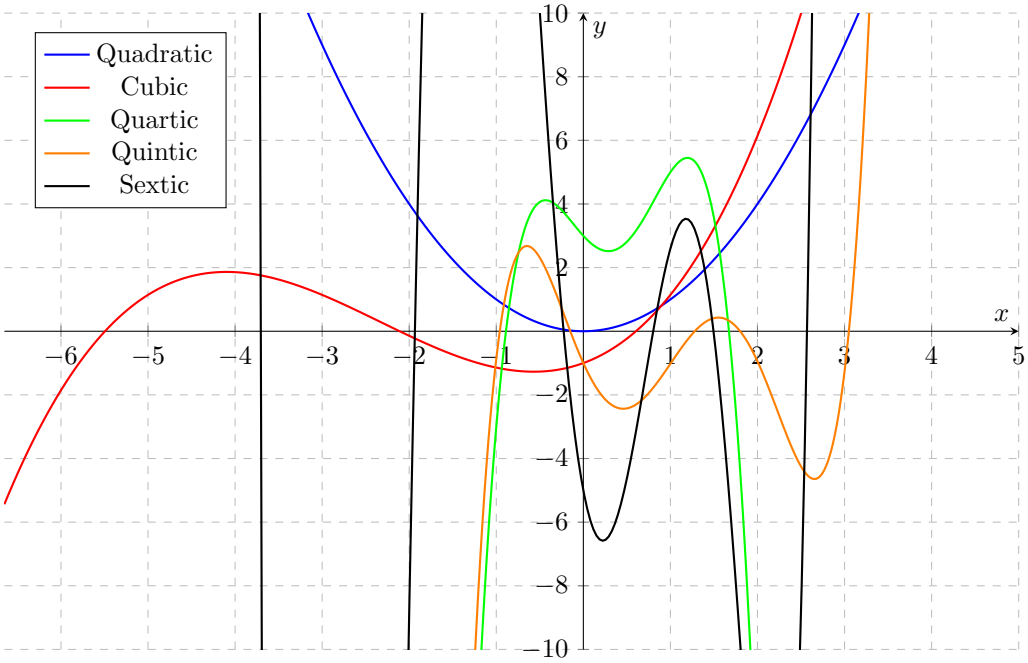

Figure 18: Graph of the given functions

Tests are designed with pytest, the code is given in the appendix 9.1 and are valid when the difference between real value and interpolated value is less than 10e-5:

Table 1: Minimum points needed to have correct interpolation

Function	Minimum number of points needed	Max step
Square	<5	>3.84
Cubic	<5	>3.84
Quartic	400	4.8e-2
Quintic	500	3.8e-2
Sextic	2000	9.6e-4

3.1.2.3 Data Mirroring

As the DNS data is only provided for half of the domain due to the simulation's symmetry, it was necessary to create a function to quantify the deviations across the entire domain compared to the model. The code is given in the appendix 9.2.

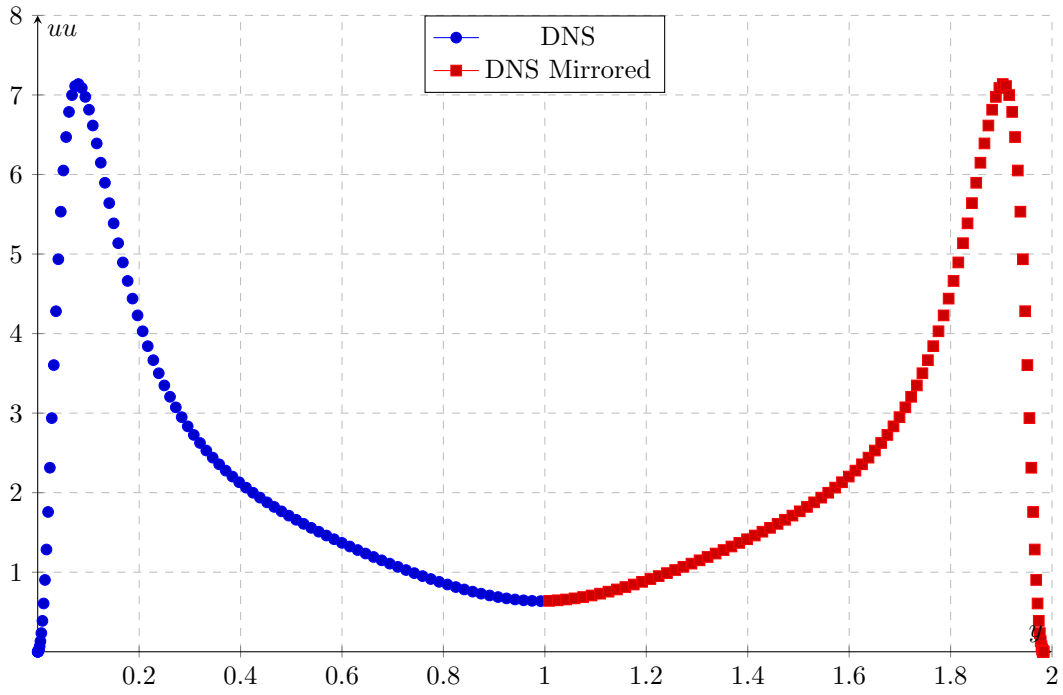

Figure 19: DNS Data Mirrored

3.2 RANS Reynolds Stresses Computation

3.2.1 Computer Fluid Dynamics RANS Simulations

To be able to simplify and automate the meshing as much as possible, CFD set-up and post-processing steps were included in a main script with the use of Matlab which allows setting up and launching the same cases but with different parameters - this was a requirement as the DNS dataset provided different cases for the same set-up.

For the CFD set-up several CFD software and packages, as well as the possibility of implementing the RANS equations by coding in Matlab and Python, were evaluated, this last option was discarded due to the complexity of implementation as well as the lack of knowledge and literature review compared to the modern CFD packages. Between the main candidates for the CFD software, there was the possibility of using Ansys Fluent, Siemens Star CCM+ and Open Foam as all of them had the possibility of being executed in Cranfields's High-Performance Computing Center for Students "*Crescent 2*". The final decision was to use Ansys Fluent for several reasons, the main ones being the availability of resources, the previous experience of the team with this software and its execution in batch mode using the terminal or system commands, the available test case and information provided by the lectures and the possibility of implementing User Define Functions in the future if the results were promising.

For that reason, to provide the meshes the software ICEM CFD was selected, which allowed the creation of macros or scripts by recording the steps used by manually creating and exporting the mesh. Additionally, this software also allowed to execution of those scripts in batch mode from the terminal or system commands.

Particularly, the versions used during this project were Matlab 2022b for implementing all the scripts into the same pipeline, Ansys Fluent 2023 R2 for setting up the simulations and journal files, Ansys 2023

R1 for the execution of jobs in the HPC centre and ICEM CFD 2023 R2 for the creation of meshes.

3.2.2 Mesh and CFD Set-Up

To recreate the DNS set-up, which consisted of the averaged results across the 3 planes of the domain in RANS, the domain was simplified to a 0.05x0.05x2 block, and the width and thickness had arbitrary values but the domain height was fixed as it had to be the double of the channel half-width specified in the DNS data, which had a value of 1. As the DNS data was averaged to a line, the domain for the RANS simulation was only meshed across the vertical line (y-axis).

To create the mesh replay file in ICEM, which allowed automatic creation and exporting of the mesh, firstly the geometry had to be defined. ICEM allowed the option to create a 3D-Block by manually inputting the 8 sets of coordinates which define the vertex of the domain; for the first simulations the domain had one of its corners in 0,0,0 and the opposite in (Lx,Lz,Ly), however on newer updates of the code, the domain was moved so its centre was aligned with 0,0,0 which allowed setting up simulations with a rotated domain.

Listing 1: Macro for the Mesh Coordinates - ICEM CFD 2023 R2

```
ic_point {} GEOM pnt.00 -0.02500,-1.00000,-0.02500
...
ic_point {} GEOM pnt.07 0.02500,1.00000,0.02500
```

After creating the vertex, the different edges and faces that formed the domain were defined and the surfaces were renamed to INLET, OUTLET, LEFT, RIGHT, TOP and BOTTOM.

Listing 2: Macro for the Mesh Surfaces - ICEM CFD 2023 R2

```
ic_geo_set_part surface srf.01 INLET 0
ic_delete_empty_parts
...
ic_delete_empty_parts
ic_geo_set_part surface srf.04 LEFT 0
```

The domain was created with the use of the blocking tool and the corners of the block were associated with their respective corners.

Listing 3: Macro for the Mesh Fluid Domain - ICEM CFD 2023 R2

```
ic_geo_new_family FLUID
ic_boco_set_part_color FLUID
ic_hex_initialize_blocking {curve crv.00 curve crv.01 ... surface srf.00...
surface srf.05} FLUID 1 101
```

For the mesh, only the parameters for the height of the domain had to be defined as, both the thickness and the width do not have any division. As can be seen in the.rpl instruction below, firstly the line needs to be defined (points 37 41)following the number of points, which was calculated with equation (33), needs to be imputed (n 60); additional parameters specific for the refinement at each wall boundary had to be defined, being h1rel and r1 the minimum wall distance (equation (32)) and growth rate for the bottom edge and

h2rel and r2 the minimum wall distance and growth rate for the top edge, and the maximum length which was set as default ($1e+10$). Following this, a final option must be inputted (*copy_to_parallel unlocked*) which copies these parameters to all 3 additional parallel edges and achieves a uniform mesh.

$$y = \frac{y^+ \cdot \mu}{\rho \cdot u_{tau}} = \frac{y^+ \cdot \nu}{u_{tau}} \tag{32}$$

$$n = 2 * \left[\left[\frac{log_{10}(\frac{\frac{L}{2} * gr - 1}{y} + 1)}{log_{10}(gr)} \right] - 1 \right] \tag{33}$$

Listing 4: Macro for the Mesh Refinement - ICEM CFD 2023 R2

```
ic_hex_set_mesh 37 41 n 60 h1rel 0.0001 h2rel 0.0001 r1 1.2 r2 1.2 lmax 1e+10 default...
copy_to_parallel unlocked
```

Finally, the unstructured mesh, figure 20 and project were saved and the mesh was exported to .msh files, which can be read and imported by fluent. This also involved changing the directory to the same directory as the current folder to allow the full automation of the script.

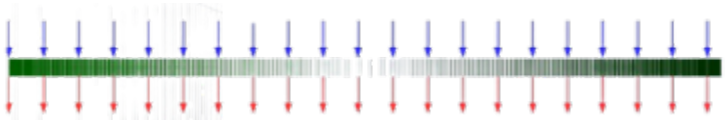

Figure 20: Channel Flow Reynolds 182 Mesh in Ansys Fluent

For the fluent simulation, the main structure of the journal file remained the same for both the channel and Couette Flow. The journal file was created with Ansys Fluent Text to User Interface (TUI) which allows fully automatising Ansys Fluent simulations and access parameters that could not be associated with the Graphic User Interface that Ansys provides. To set up the simulation, firstly the mesh is imported.

Listing 5: Macro for the Mesh Importation in Fluent - ANSYS Fluent 2023 R2

```
/file/read-case/fluent.msh
```

Following, the INLET, OUTLET, LEFT, and RIGHT surfaces were designated as Conformal Periodic Zones. These zones consist of identical boundaries applied to walls to model periodic conditions, where the fluid on one wall enters the other with the same properties. This allows for the modelling of larger domains with less computational power. Both the TOP and Bottom surfaces remained as walls, which was set by default by Ansys. Additionally, the density and dynamic viscosity of the material (named air by default) were changed to match the DNS cases.

On the one hand, to set up the boundary conditions of the channel flow, the x-momentum source term had to be defined. This term accounts for the pressure drop that drives the flow, as it was explained

in the Test Cases Section. To find out this term equation (**??**) had to be used.

Listing 6: Macro for Boundary Conditions definition for Channel Flow - ANSYS Fluent 2023 R2

```
/define/materials/change-create air air yes constant 1.0 no no yes constant 0.00035 no no no
/define/boundary-conditions/fluid 10 no yes 0 1 yes 0.00406161 1 yes 0 0 0 0 no no no ...
0 no 0 no 0 no 0 no 0 no no no no no
/mesh/modify-zones/make-periodic 12 15 no yes yes
/mesh/modify-zones/make-periodic 17 14 no yes yes
```

On the other hand, to set up the Couette flow, the pressure drop term remained 0 and the velocity at the walls was set up to be 0 m/s at the bottom wall and 2 m/s at the top wall.

Listing 7: Macro for Boundary Conditions definition for Couette Flow - ANSYS Fluent 2023 R2

```
/define/materials/change-create air air yes constant 1 no no yes constant 0.000666 no no no
/mesh/modify-zones/make-periodic 12 15 no yes yes
/mesh/modify-zones/make-periodic 17 14 no yes yes
/define/boundary-conditions/wall top yes motion-bc-moving no no yes no no 2.0 1 0 0 ...
no no 0 no 0.5
/define/boundary-conditions/wall bottom yes motion-bc-moving no no yes no no 0.0 1 0 0 ...
no no 0 no 0.5
```

Fluent, by default sets the viscous model to be the kw-sst, which is the standard set by the industry in terms of the relation between global accuracy and computational cost. For that reason, the model used for this simulation was the kw-sst with the default parameters set by fluent. Additionally, in terms of the convergence, the absolute convergence criteria were set up to be $1e-08$ for all the residuals (continuity, x-velocity, y-velocity, z-velocity, k and omega). The residuals are values that represent the error between the left-hand side of the RANS equations, which usually represent the temporal and spatial gradients of the field variables and the right-hand side, which usually represents the external terms such as the external influences, source terms, forcing functions, or other terms not directly related to the derivatives of the dependent variables; in RANS those terms can represent pressure gradients, body forces (such as gravity), viscous dissipation, and other sources or sinks of momentum and energy in the flow. This difference is called imbalance and the absolute convergence criteria define the minimum threshold for the numerical solution for that equation to be accepted.

Listing 8: Macro for the Viscosu Model (kw-sst) Definition - ANSYS Fluent 2023 R2

```
/define/models/viscous/kw-sst? yes
/solve/monitors/residual/converge-criteria 1e-08 1e-08 1e-08 1e-08 1e-08 1e-08
```

The main value that has been used to monitor the accuracy of the results has been the mean velocity profile. Because the average normalised velocity field in the DNS dataset was known to be 1 m/s for the Channel flow and Couette flow (with 0 m/s and 2 m/s velocity at the walls), the average velocity across the domain in the RANS was computed at each iteration and was used to evaluate whether the set-up worked or not. To monitor the value, a report which computed the volume average of the x-velocity across the entire domain was created, which was later included in a monitor plot that also printed the value in the console log as the iterations progressed.

Listing 9: Macro for Mean Velocity Report Definition- ANSYS Fluent 2023 R2

```
/solve/report-definitions/add mean-velocity volume-average zone-names fluid () ...
report-type volume-average field x-velocity quit
/solve/report-files/add/mean-velocity report-defs mean-velocity () print yes quit
/solve/report-plots/add mean-vel-final4 report-defs mean-velocity () x-label iteration ...
y-label volume plot-instantaneous-values? yes print? yes quit
quit
```

For the convergence of the simulation, the solving process was divided into three stages. Firstly, a standard initialisation followed by a hybrid initialisation was performed to reduce the computation time. The standard initialisation refers to manually specifying the starting value to all cells of the domain, which is set to 0. In these particular cases, as the mean velocity was one in the two cases, the x-velocity field was set to 1 m/s, while the other variables remained set up as 0; whereas, the hybrid initialisation computes the first 10 iterations and provided a more accurate initial guess to all cells in the domain.

Secondly, a first solving of the case was made with a first-order scheme applied to the k equation, the omega equation and the three momentum equations. In this case, the first-order upwind scheme was used because it provided initial numerical stability to solve the first iterations of the gradients as well as to approximate the boundary layer to the final solution, as well as they are less numerically expensive. This was set up for 100 iterations, and then the numerical scheme of all equations was changed to a second-order-upwind scheme, characterised for being more computationally demanding and more accurate than the first order, allowing to better resolve the boundary layer and gradients, achieving a better solution.

Listing 10: Macro for Solution Criteria definitio- ANSYS Fluent 2023 R2

```
/solve/initialize/set-defaults/x-velocity 0
/solve/set/discretization-scheme/k/0
/solve/set/discretization-scheme/omega/0
/solve/set/discretization-scheme/mom/0
/solve/initialize/compute-defaults/fluid 10
/solve/initialize/initialize-flow
/solve/iterate 100
/solve/set/discretization-scheme/k/1
/solve/set/discretization-scheme/omega/1
/solve/set/discretization-scheme/mom/1
/solve/iterate 1000000
```

Finally, once the simulations converged the main parameters were exported into files to be later read by Matlab and the case and data were saved to later analysed in case there were abnormal values or more data needed to be obtained.

Listing 11: Macro for the Data Exportation from Fluent - ANSYS Fluent 2023 R2

```
/plot/plot yes "x_velocity" yes no no x-velocity yes 0 1 0 int_fluid ()
...
/plot/plot yes "y-plus" yes no no y-plus yes 0 1 0 int_fluid ()
/file/write-case "Re_93cas"
/file/write-data "Re_93dat"
exit ok
```

3.2.3 Viscous Model Selection

To be able to solve the RANS equations numerically, additional equations must be introduced to close the model and provide an accurate prediction of the turbulent properties of the flow. As executing all 8 DNS cases (5 Reynolds for Channel and 3 for Couette) with all viscous models, an initial benchmark study was performed with the Channel Flow Re 182. The first and main criteria to evaluate whether the viscous models could be used to predict the Reynolds Stresses correctly was to evaluate the area-weighted average velocity in the x-direction which was computed with the use of the report created with the Fluent Macro and updated with each iteration. By analysing the characteristics of the flow, the average velocity had to be equal to 1.

For that reason, by changing the viscous models defined in the Listing 8, different cases were executed and the evolution of the velocity with the iterations was monitored to evaluate if the model could be accepted or not. Additionally, for each model, different solution methods were performed, including the combination of first and second-order discretisation methods, initialisation with previously converged models and the specification of wall functions. In figure 21, the average velocity convergence of the different models that were tested is displayed. All models were converged with a residual convergence criteria of $1e-10$, except the two cases where the k-omega SST model was used to initialise a converged solution which was set up with a converge criteria of $1e-8$ and followed up by the new viscous model which was then set to $1e-10$. The first 1000 Iterations of each case were performed with a first-order upwind scheme for all equations except the pressure and then switched to a second-order scheme, except for the Reynolds Transport Equation with and without Wall Treatment Enhancement which started directly with a second-order scheme.

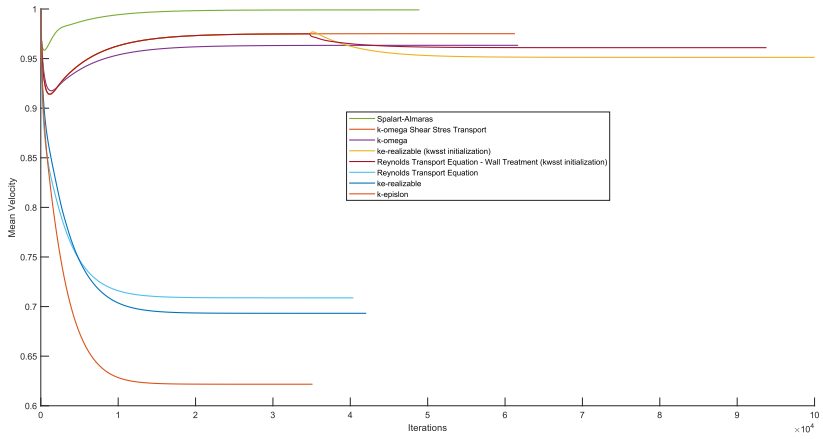

Figure 21: Unnormalised Mean Velocity Convergence for all Tested RANS Visocus Models - Reynolds 182 Channel Flow

From figure 21, two groups can be distinguished. Firstly, we have 3 viscous models which have not achieved an acceptable convergence, which are the standard k-epsilon model, the realisable k-epsilon model and the Standard Reynolds Transport Equation. On the other hand, the standard k-omega, SST k-omega and Spalart-Almaras models converged with any additional constraint. Additionally, by applying the wall enhancement treatment option to the Reynolds Stress Transport model, the solution converged with a final average velocity of 0.96101 instead of the initial 0.7088 (obtained without the Wall Treatment

Enhancement). Finally, there were two cases corresponding to the realisable k-epsilon model and the Reynolds Transport Equation with Wall Treatment Enhancement, which were initialised with a converged SST k-omega model achieving a significant improvement for the realisable k-epsilon and the same result for the Reynolds Transport model.

Additionally, from all the models that converged additional simulations for the Channel Flow Re 5200 to compared with the highest Reynolds case for a better understanding of the RANS shortcomings when predicting highly turbulent flows. The mean convergence of all models can be seen in the figure 22, where lines of the same colour are from the same model, where the solid line is for the 182 Reynolds case and the dashed line for the 5200 Reynolds case.

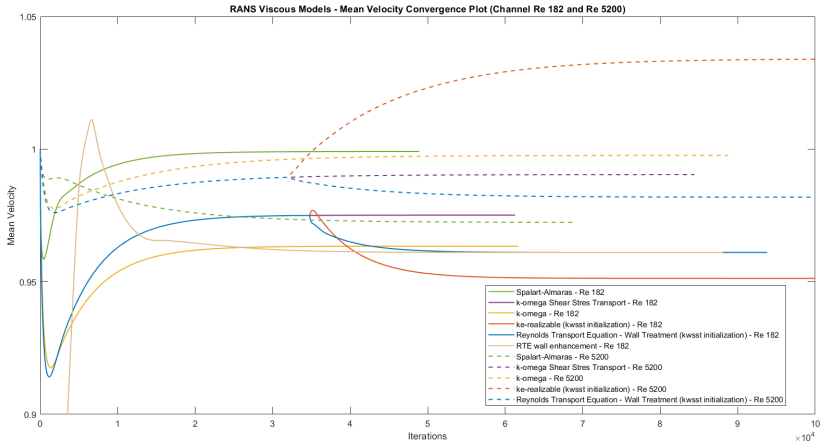

Figure 22: Unnormalised Mean Velocity Convergence for All Tested RANS Visocus Models - Reynolds 5200 Channel Flow

As the Spalart-Alamaras model does not include a transport equation for the turbulent kinetic energy, the computation of the normal Reynolds Stresses ($u'u'$, $v'v'$ and $w'w'$ using the standard Boussinesq Hypothesis, could not be performed. Therefore the results for the mean velocity profiles and Reynolds Stresses for the Spalart-Asmara's model were calculated by omitting the term associated with the normal stress, as explained in the Turbulence Modelling. Additionally, for the 5200 Reynolds Case, the Reynolds Transport Model with Wall Treatment Enhancement and without the initial convergence with the K-w SST model diverged after performing 329 Iteration, so any results could have been obtained from those simulations. However, from the 182 Reynolds Case, the results obtained from the simulations with and without a K-w SST initialisation gave the same results, so it could be expected to have the same behaviour for 5200. For that reason, this model without the initialisation was discarded from the selection. As it was done with the DNS data, all plots that were negligible have not been included, such as the Y and Z velocity profiles and the $u'w'$ and $v'w'$. The most relevant results for both the Re 182 and Re 5200 Channel flow are shown in figure 23.

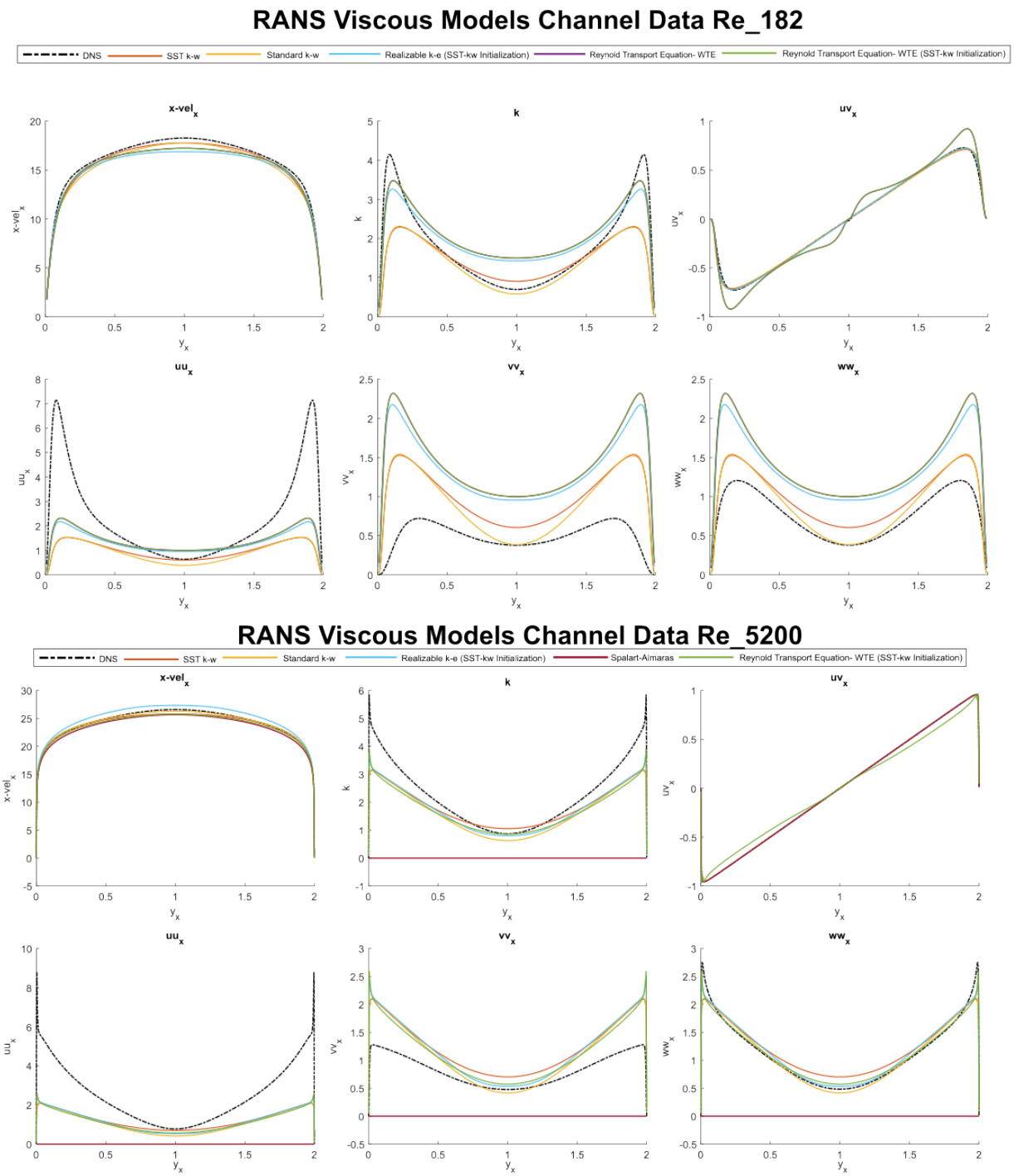

Figure 23: RANS Viscous Models Results for Reynolds 182 and Reynolds 5200 Channel Flow

This section aimed to select the best model to obtain the Reynold Stresses which will be compared to the ones obtained with the new framework, so a quantitative analysis regarding the accuracy of the models compared with the DNS has been carried out; however in the results, subsection, a more in-depth qualitative analysis for all viscous models has been performed to provide a better understanding of the shortcomings of each method. For the quantitive analysis, the RMSE (RSME) for all models were computed, so by using

the Interpolation algorithms provided and the RSME computation.

For the Reynolds 182 Channel Flow, the results obtained are seen in table 2:

	RTE_wte	kwsst_ke_rel	kw	kwsst	kwsst_RTE_WTE
U	0.681	0.860	0.624	0.449	0.681
K	0.541	0.516	0.721	0.725	0.541
uu	1.693	1.760	2.087	2.076	1.693
vv	1.022	0.940	0.547	0.575	1.022
ww	0.689	0.608	0.192	0.239	0.689
uv	0.095	0.016	0.024	0.022	0.095
Averaged	1.376	0.783	0.699	0.681	0.787
RST Averaged	1.515	0.768	0.714	0.727	0.808

Table 2: RMSE for Reynolds 182 Channel Flow

For the Reynolds 5200 Channel Flow, the results obtained are seen in table 3:

	SA	kwsst_ke_rel	kw	kwsst	kwsst_RTE_WTE
U	0.683	0.839	0.162	0.415	0.508
K	2.514	0.589	0.678	0.639	0.662
uu	2.927	1.653	1.709	1.674	1.702
vv	0.867	0.459	0.410	0.450	0.411
ww	1.264	0.103	0.116	0.172	0.091
uv	0.003	0.003	0.006	0.007	0.060
Averaged	1.376	0.608	0.514	0.559	0.572
RST Averaged	1.515	0.561	0.584	0.576	0.585

Table 3: RMSE for Reynolds 5200 Channel Flow

These results show that when computing the averaged Root Mean Square Error (RSME) for all variables and non-zero components of the Reynolds Stress Tensor, errors for most variables are within a similar range across models, except for the Spalart-Allmaras model. This is due to inaccuracy when predicting the normal stresses, as the term accounting for the turbulent kinetic energy was assumed to be 0 and the normal gradients ($\partial U/\partial X$, $\partial V/\partial Y$ and $\partial W/\partial Z$) are 0. Of all the models, the ones that provide the best results are the standard k-w and SST k-w, being the latter one more accurate when predicting the normal stress in the y-direction ($v'v'$).

When analysing each field output, individually when comparing the velocity field with the DNS dataset, the prediction for the 5200 case was more accurate than the 182 case, as well as it is observed that almost every model (except the RTM for the Reynolds 5200 Case), tended to underestimate the velocity field across all the channel.

Regarding the Reynolds Stresses, all models accurately predicted $u'v'$ except for the Reynold Transport Model, which showed an error increase of 80% for the Reynolds 182 case and about 90% for the Reynolds 5200 case. This discrepancy may come from the way the Reynolds stress components are retrieved, as Fluent allows exporting the value directly, as they are computed in individual transport

equations. Regarding the normal stresses, discrepancies were observed due to local turbulent isotropy, with similar values for the three normal components. This could be because the turbulent viscous stress term of the Boussinesq hypothesis (influenced by velocity gradients) is negligible compared to the turbulent pressure term, which is driven by turbulent kinetic energy. For the Reynolds 182 case, we can see the difference between the RANS models when computing the normal stresses and turbulent kinetic energy are much greater than the ones predicted in the 5200 cases. However, it can be seen that the realisable k-e and RTM give much higher values than both k-w models, having better results when predicting the $u'u'$ component and worst results for $v'v'$ and $w'w'$. However, when comparing the values of the turbulent kinetic energy, it can be seen that both models have more inaccurate predictions than the k-w.

From all the 5 models tested, the Shear Stress Transport (SST) K-w model was selected as a benchmark as it was one of the models that had better results together with the standard k-w, as well as the current standard model used in the industry.

3.2.4 Pipeline and Main Script

The code starts with a loop function which stores the main parameters that will be input to a main script, which is the function to obtain all the parameters and export them in files. The main parameters defined in the loop function are the fluid properties for each different Reynolds Number Case such as the Friction Reynolds Number (Re_{tau}), the kinematic viscosity (k_visc) and friction velocity (u_{tau}. Additional parameters specific for the Ansys Fluent Simulation such as the growth rate ($growth\_rate$), number of Iterations for each discretisation method ($Iters$), domain dimensions (L_x,L_y and L_z) and the density (ρ), which has been assumed to be one; were used as inputs to create the different files necessary to execute the simulations. The function has a built-in loop which calls the main script and inputs the specific data for each case.

Listing 12: Matlab Launcher or Loop Script

```
clear all;
Re=[182.088, 543.496, 1000.512, 1994.756, 5185.897] %Friction Reynolds Number
u_tau=[6.37309e-02, 5.43496e-02, 5.00256e-02, 4.58794e-02, 4.14872e-02 ] %Friction Velocity
k_visc=[3.50000e-04, 1.00000e-04, 5.00000e-05, 2.30000e-05, 8.00000e-06] %Kinematic Visc
Lx = 0.05; %Length
Lz = 0.05; %Width
Ly = 2; %Height
U=1.000; %Mean velocity
dens=1; % Density
y_plus=1; %y_plus
g_rate=1.1; %Growth Rate
Iters_f=1000; %Iterations First Order Scheme
Iters_s=1000000; %Iterations Second Order Scheme
Residuals = 1e-08;%Residualas Convergence

s=size(Re)
for i=1:s(2)
    Main(Re(i), u_tau(i), k_visc(i),Lx,Ly,Lz,dens,y_plus,g_rate,Iters_f,Iters_s, Residuals)
end
```

The main script consists of two different sets of functions which aim to modify and export a template for both the mesh replay (scripts for ICEM CFD) and journal files, which are later executed by the use of system commands, which access the terminal and allows to automatically obtain a fluent mesh file from the replay file which is then used by fluent to set-up and execute the simulation. The Fluent Journal file, as it will be explained later, contains some commands which allow to export of the desired field variables in separate files. The last step of the script is to read those variables, post-process them and export them in .txt or .csv files. This whole pipeline has been condensed and represented in a diagram in figure 24.

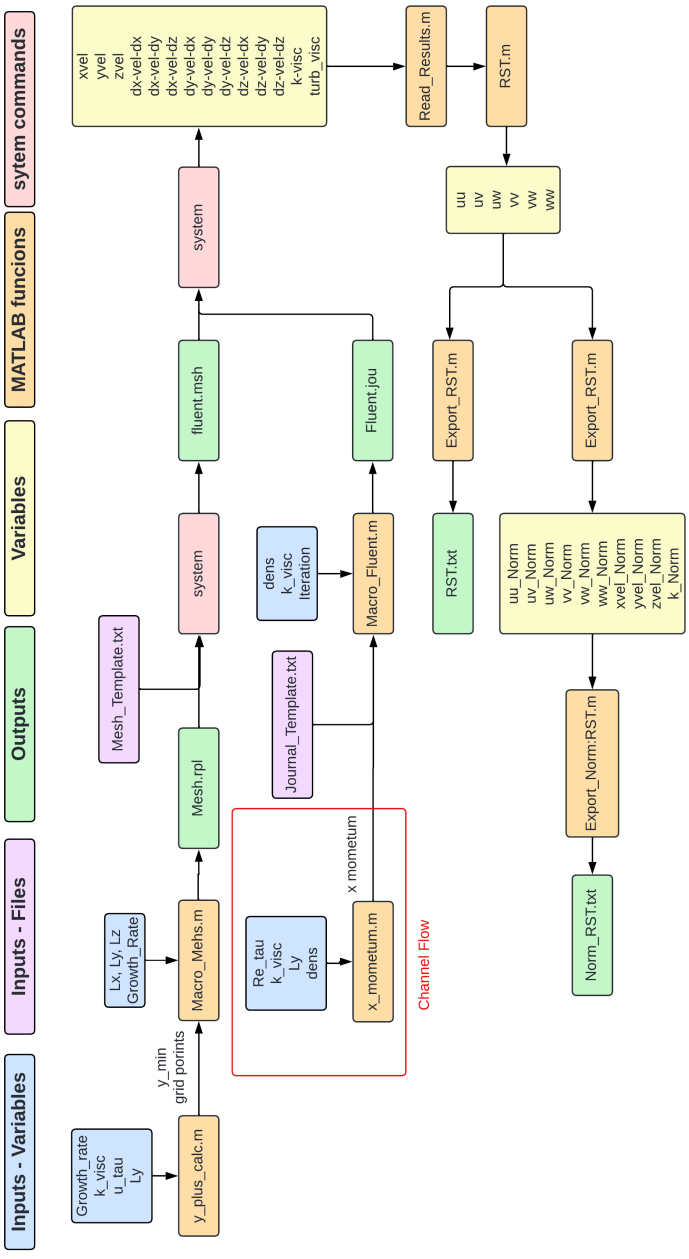

Figure 24: Iterative Process

Listing 13: Matlab Main Script

```
function []= Main(Re, u_tau, k_visc,Lx,Ly,Lz,dens,y_plus,growth_rate,Iters_f,Iters_s,...
Residuals)
Re_str=int2str(Re);
child_folder_name = strcat('Files_',Re_str);

%Macro- Mesh-------------------------------------------------------------------
[y,n] = y_plus_calc(y_plus,k_visc,u_tau,Ly, growth_rate)
[S,child_folder_name]=Macro_Mesh(Lx, Ly, Lz, y, growth_rate,n,child_folder_name)

%Fluent Macro -----------------------------------------------------------------
xM = momentum(Re,k_visc,Ly,dens);
Macro_Fluent(dens,k_visc,xM,Iters_f,Iters_s,child_folder_name,Re)

%Execute ICEM CFD -------------------------------------------------------------

cd(child_folder_name);
ansys_executable = '"C:\Program Files\ANSYS Inc\v232\icemcfd\win64_amd\bin\icemcfd.bat"'
macro_script = 'Mesh_New.rpl'
command = [ansys_executable, ' -batch -script ', macro_script];
system(command);

%Execute Fluent-- ------------------------------------------------------------

ansys_executable = '"C:\Program Files\ANSYS Inc\v232\fluent\ntbin\win64\fluent.exe"'
macro_script = 'Case_new.jou'
command = [ansys_executable, '3d', '-g', '-i', macro_script]
system(command)

% currentFolder = cd; %
% parentDirectory = fileparts(currentFolder);
% cd(parentDirectory);

% Results ----------------------------------------------

[y,dxdx,dxdy,dxdz,dydx,dydy,dydz,dzdx,dzdy,dzdz,xvel,yvel,zvel,k,t_visc]=...
    Read_Results(child_folder_name);
[uu,uv,uw,vv,vw,ww]=RST(y,dxdx,dxdy,dxdz,dydx,dydy,dydz,dzdx,dzdy,dzdz,t_visc,k);
[uu_norm,uv_norm,uw_norm,vv_norm,vw_norm,ww_norm,xvel_norm,yvel_norm,zvel_norm]=...
    Normalization(y,uu,uv,uw,vv,vw,ww,xvel,yvel,zvel,u_tau)

%Post-Process-----------------------------------------------------------------------------

Export_RST(y,uu,uv,uw,vv,vw,ww,xvel,yvel,zvel,Re);
Export_RST_Norm(y,uu_norm,uv_norm,uw_norm,vv_norm,vw_norm,ww_norm,...
    xvel_norm,yvel_norm,zvel_norm,Re);

end
```

3.2.4.1 Mesh Macro Script

For the mesh automation script, both Channel and Couette flows had the same structure, where an initial script computed the minimum y distance to the wall as well as the number of grid points, and a second

script replaced specific lines of the original ICEM replay template with the specific information for each case.

The first script received as input the targeted y^+, the kinematic viscosity (ν), the friction velocity (u_tau), the channel half width (L) and growth rate (gr), and with the y^+ formula shown in equation (32) it obtained the minimum wall distance, and with the number of points was found out with the geometric series sum expression displayed in equation (33).

Listing 14: Matlab y plus Calculator

```
function [y,n] = y_plus_calc(y_plus,k_visc,u_tau,L,gr)
    y=(y_plus*k_visc)/u_tau;
    n=2*(round(log(((L/2)*(gr-1))/(y)+1)/log(gr))-1);
end
```

The second script takes as input the data obtained in the previous script as well as the growth rate and the 3 dimensions of the domain: height (Ly), width (Lx) and thickness (Lz). The first part of the code allows the creation of folders named after the different Reynolds Numbers and includes a while loop which prevents the creation of folders with the same name. After reading the templates and storing in an array of strings it uses a function called *ReplaceLine.m* which replaces a section of the string determined by the number of the line, the position of the first character and the number of characters. An additional input was added to set both the format of the variable as well as the desired number of decimals to required.

Both scripts as well as the template .rpl file can be found in the Technical Work Submission.

3.2.4.2 Fluent Journal File Script

The Journal files script follows the same structure, but it has different cases for both the Couette and Channel flow.

Channel Flow

As explained before, the Channel Flow is a pressure-driven flow and expression for the pressure gradient in the X direction is given by equation (29). This expression has been explained and transformed into a script that takes into account the Friction Reynolds Numbers (Re), kinematic viscosity (ν), height (Ly) and density (ρ).

Listing 15: Matlab Pressure Gradient Calculator

```
function xM= momentum(Re,k_visc,Ly,dens)
    u_t=Re*k_visc/(Ly/2);
    tau=dens*u_t^2;
    xM=(2*tau)/(Ly/2);
    xM=xM/2;
end
```

Secondly, as it was done with the macro mesh function, a script was made to replace specific parameters on a template journal case for the Channel flow. This involves changing specific parameters such as the density, xMomentum Source, dynamic viscosity, number of iterations or convergence criteria. The same

methodology of using the *replaceline.m* script was used.

Similarly, for the Couette flow the same approach was made as the Channel Flow, with the main difference being that the Couette flow did not require any additional function to compute additional terms as the setup remained the same for all the cases and the main change between them was changing the viscosity. Additionally, other values as the convergence criteria and Iterations were made.

3.2.4.3 Results Post-Processing Scripts

The Ansys Fluent Simulations results obtained are laid out in table 4:

Fluent Variable Name	**Matlab Variable**	**Description**
x_velocity	xvel	Velocity in the X direction
y_velocity	yvel	Velocity in the Y direction
z_velocity	zvel	Velocity in the Z direction
dx_velocity_dx	dxdx	X Velocity Gradient in the X Direction
dx_velocity_dy	dxdx	X Velocity Gradient in the Y Direction
dx_velocity_dz	dxdy	X Velocity Gradient in the Z Direction
dy_velocity_dx	dxdz	Y Velocity Gradient in the X Direction
dy_velocity_dy	dydy	Y Velocity Gradient in the Y Direction
dy_velocity_dz	dydz	Y Velocity Gradient in the Z Direction
dz_velocity_dx	dzdx	Z Velocity Gradient in the X Direction
dz_velocity_dy	dzdy	Z Velocity Gradient in the Y Direction
dz_velocity_dz	dzdz	Z Velocity Gradient in the Z Direction
turb-kinetic-energy	k	Turbulent Kinetic Energy
viscosity-turb	turb_visc	Turbulent Viscosity

Table 4: Main Outputs of Fluent

These results were stored in arrays in Matlab and post-processed to obtain the Reynold Stress Tensor for each position along the y-axis. To compute the Reynold Stress Tensor, the Bousinnesq Hypothesis, displayed in Equation (11) was expanded to all 6 individual expressions:

$$\tau_{xx} = \overline{u'u'} = -2\rho\nu_t \left(\frac{\partial u}{\partial x} - \frac{1}{3} \cdot k \cdot \mathbf{u} \right)$$
$$\tau_{yy} = \overline{v'v'} = -2\rho\nu_t \left(\frac{\partial v}{\partial y} - \frac{1}{3} \cdot k \cdot \mathbf{u} \right)$$
$$\tau_{zz} = \overline{w'w'} = -2\rho\nu_t \left(\frac{\partial w}{\partial z} - \frac{1}{3} \cdot k \cdot \mathbf{u} \right)$$
$$\tau_{xy} = \overline{u'v'} = -\rho\nu_t \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)$$
$$\tau_{xz} = \overline{u'v''} = -\rho\nu_t \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \right)$$
$$\tau_{yz} = \overline{v'w'} = -\rho\nu_t \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \right)$$

These expressions were defined in Matlab, using the variables obtained from fluent Matlab and obtained the unnormalised Reynolds Stresses across the domain.

Listing 16: Matlab Bousinnesq Hypothesis Calculator

```
function [uu,uv,uw,vv,vw,ww]=RST(y,dxdx,dxdy,dxdz,dydx,dydy,dydz,dzdx,dzdy,dzdz,visc,k)
s=size(y);
s=s(1);
for i=1:s
    uu(i)=-visc(i)*(dxdx(i)+dxdx(i))-2/3*k(i);
    vv(i)=-visc(i)*(dydy(i)+dydy(i))-2/3*k(i);
    ww(i)=-visc(i)*(dxdz(i)+dzdz(i))-2/3*k(i);
    uv(i)=-visc(i)*(dxdy(i)+dydx(i));
    uw(i)=-visc(i)*(dxdz(i)+dzdx(i));
    vw(i)=-visc(i)*(dydz(i)+dzdy(i));
end
end
```

To be able to compare the result with DNS datasets, the data needs to be normalised; specifically, all the velocities were normalised by the friction velocity (u_{tau}) and the stress terms as well as the energy terms are normalised by the squared of the friction velocity (u_{tau}^2).

Listing 17: Matlab Normalisation Script

```
function [uu_norm,uv_norm,uw_norm,vv_norm,vw_norm,ww_norm,xvel_norm,yvel_norm,zvel_norm,...
k_norm]=Normalization(y,uu,uv,uw,vv,vw,ww,xvel,yvel,zvel,u_tau,k)
s=size(y);
s=s(1);
for i=1:s
    uu_norm(i)= -uu(i)/(u_tau)^2;
    vv_norm(i)= vv(i)/(u_tau)^2;
    ww_norm(i)= ww(i)/(u_tau)^2;
    uv_norm(i)=uv(i)/(u_tau)^2;
    uw_norm(i)=uw(i)/(u_tau)^2;
    vw_norm(i)=vw(i)/(u_tau)^2;
    xvel_norm(i)=xvel(i)/u_tau;
    yvel_norm(i)=yvel(i)/u_tau;
    zvel_norm(i)=zvel(i)/u_tau;
    k_norm(i)=k(i)/(u_tau)^2;
end
```

Finally, once all the variables were computed, all the outputs were exported in three documents to help organise all the information. Those files were:

- The Normalised Reynolds Stress Tensor and Velocity fields
- The Unnormalised Reynolds Stress Tensor and Velocity fields
- The Velocity field and Gradients

3.2.4.4 High Performance Computing

Due to the number of resources and time required to execute all cases for the Channel Flow and Couette flow, it was decided to make some modifications to the original main script as it was more complex to adapt the current script in Crescent 2 than two modify the structure of the code to manually launch all the cases. For that reason, the main code was split into two functions where the first one created the meshes and the journal and submission files necessary to submit the simulations, and the second script read all the outputs from Ansys Fluent and computed and exported the RST.

3.2.4.5 Rotated Domain and Data Enhancement

As was explained before, to obtain more data to have a more complete dataset and be able to define a more complete PINN model for all cases, the previous script was modified to allow rotating the domain along the Z-axis by maintaining the same global axis. This allowed us to obtain different values for the Reynolds Stress Tensor for each angle. This script was only created for channel flows as the idea was later discarded due to time constraints.

Firstly, instead of the coordinate $(0, 0, 0)$ being located at a corner, the whole domain was translated in all three axes to ensure, that the $(0, 0, 0)$ coordinate matched the centre of the domain, therefore facilitating the implementation of a function that rotated the domain in all three axes. To achieve this, all the coordinates were stored in a matrix ($Vert$) where the columns corresponded to the X, Y and Z coordinates and each row represented each vertex. Then three rotation matrices (R_x, R_y and R_z) were defined to compute the rotations in the ZY plane (α), ZX plane (β) and XY plane (γ). The rotated domain (R_Vert) was computed with equation (34):

$$R\_Vert = Vert * R_x * R_y * R_z \tag{34}$$

where:

$$Vert = \begin{bmatrix} -\mathrm{Lx}/2 & -\mathrm{Ly}/2 & -\mathrm{Lz}/2 \\ \mathrm{Lx}/2 & -\mathrm{Ly}/2 & -\mathrm{Lz}/2 \\ -\mathrm{Lx}/2 & \mathrm{Ly}/2 & -\mathrm{Lz}/2 \\ \mathrm{Lx}/2 & \mathrm{Ly}/2 & -\mathrm{Lz}/2 \\ -\mathrm{Lx}/2 & -\mathrm{Ly}/2 & \mathrm{Lz}/2 \\ \mathrm{Lx}/2 & -\mathrm{Ly}/2 & \mathrm{Lz}/2 \\ -\mathrm{Lx}/2 & \mathrm{Ly}/2 & \mathrm{Lz}/2 \\ \mathrm{Lx}/2 & \mathrm{Ly}/2 & \mathrm{Lz}/2 \end{bmatrix}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Moreover, the x-momentum source had to be decomposed both in the X and Y direction using gamma as an angle:

Listing 18: Matlab Pressure Gradient Decomposition

```
xM_y=-sin(gamma)*xM;
xM_x=cos(gamma)*xM;
```

Finally, the last modification involved creating a line that had a starting point at the centre of the bottom face and an ending point at the centre of the top wall. This allowed exporting the results in a specific direction using the plot command inside Fluent. To achieve this, both the coordinates of the centre top and bottom faces had to be automatically computed, as well as the normalised vector of the line.

Listing 19: Macro for the Data Exportation in a Rotated Domain from Fluent - ANSYS Fluent 2023 R2

```
/surface/line-surface line-7 -0.86602540 -0.50000000 0 0.86602540 0.50000000 0
...
/plot/plot yes "x_velocity_l7" yes no no x-velocity yes 0.86602540 0.50000000 0 line-7 ()
```

Once some results for different angles were obtained using ANSYS FLuent, they were used to create a script that allowed the correct transformation of the results exported from unrotated domains into the ones from rotated domains without the execution of any simulation. This script could be used both for DNS and RANS results. The rotated domain comparison is captured in figure 25.

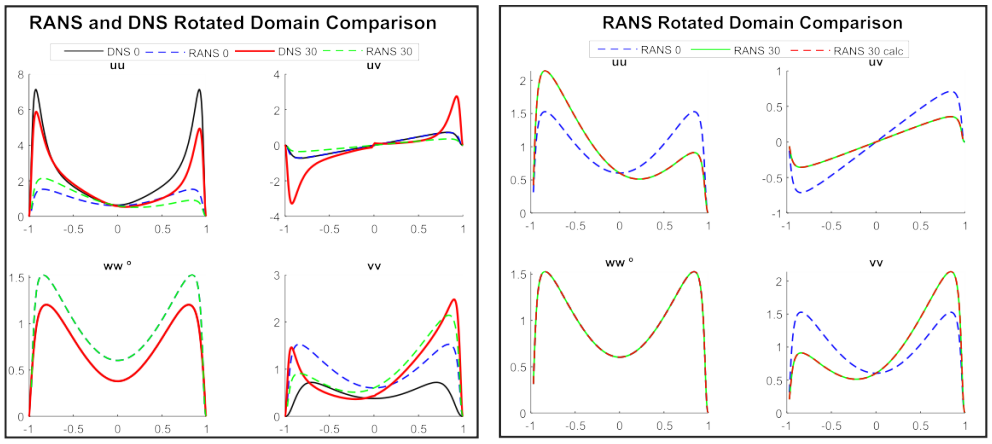

Figure 25: RANS and DNS Reynolds Stress Tensor for 0º and 30º

3.3 Discovering Governing Equations for Spatial Dynamical Systems with SINDy

3.3.1 Introduction to SINDy Approach

When applying the Sparse Identification of Nonlinear Dynamics (SINDy) to spatial dynamical systems such as turbulent flows in a channel, the objective is to discover the form of the function $f(\mathbf{U}(y))$ that best represents the relationship between the spatial derivative of velocity vector $\mathbf{U}$ with respect to the spatial variable y, denoted by equation (35):

$$\frac{d\mathbf{U}}{dy} = f(\mathbf{U}(y)) \tag{35}$$

The vector $\mathbf{U}(y) = [u_1(y), u_2(y), \ldots, u_n(y)]^T$ represents the state of the system in the space of the spatial variables $y_1, y_2, \ldots, y_n$ at a given moment. To determine the form of the function f from the data, collect a spatial history of the states $\mathbf{U}(y)$ and their derivatives with respect to space $\frac{d\mathbf{U}}{dy}$, sampled at multiple locations $y_1, y_2, \ldots, y_n$. These data are then arranged into two large matrices, equations (36) and (37):

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}^T(y_1) \\ \mathbf{U}^T(y_2) \\ \vdots \\ \mathbf{U}^T(y_n) \end{bmatrix} = \begin{bmatrix} u_1(y_1) & u_2(y_1) & \ldots & u_n(y_1) \\ u_1(y_2) & u_2(y_2) & \ldots & u_n(y_2) \\ \vdots & \vdots & \ddots & \vdots \\ u_1(y_n) & u_2(y_n) & \ldots & u_n(y_n) \end{bmatrix} \tag{36}$$

$$\frac{d\mathbf{U}}{dy} = \begin{bmatrix} \left(\frac{du}{dy}\right)^T(y_1) \\ \left(\frac{du}{dy}\right)^T(y_2) \\ \vdots \\ \left(\frac{du}{dy}\right)^T(y_n) \end{bmatrix} = \begin{bmatrix} \frac{du_1}{dy}(y_1) & \frac{du_2}{dy}(y_1) & \ldots & \frac{du_n}{dy}(y_1) \\ \frac{du_1}{dy}(y_2) & \frac{du_2}{dy}(y_2) & \ldots & \frac{du_n}{dy}(y_2) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{du_1}{dy}(y_n) & \frac{du_2}{dy}(y_n) & \ldots & \frac{du_n}{dy}(y_n) \end{bmatrix} \tag{37}$$

Each data set is represented by a column vector in $\mathbf{U}$, while the corresponding spatial derivatives are represented in the matrix $\frac{d\mathbf{U}}{dy}$. These matrices provide a snapshot of the spatial dynamic state of the system across the entire domain under consideration.

A library of potential functions, denoted as $\Theta(\mathbf{U})$, is then constructed, which plays a crucial role in the SINDy algorithm. This library includes a range of potential polynomial functions that aim to capture the dynamics of the system as described by $f(\mathbf{U}(y))$. The library's composition is carefully selected to include constant terms, linear terms, higher powers of the spatial variable $\mathbf{U}$, and other relevant spatial functions, ensuring it is comprehensive enough to span the space of the dynamics of interest.

The function library $\Theta(\mathbf{U})$ can be represented by a vector of terms that includes the constant term and higher-degree monomials of $\mathbf{U}$. This vector is defined by equation (38):

$$\Theta(\mathbf{U}) = \begin{bmatrix} 1 & \mathbf{U} & \mathbf{U}^2 & \ldots & \mathbf{U}^n \end{bmatrix}^T \tag{38}$$

where each element represents a function term considered in the model. This structured compilation of

terms enables the discovery of the most parsimonious model that characterises the spatial dynamics of the system under investigation.

With our function library $\Theta(\mathbf{U})$ in place, the next task within the SINDy framework is to determine the coefficient matrix Ξ, which linearly maps the function library to the spatial derivatives of the state $\frac{d\mathbf{U}}{dy}$. This coefficient matrix is sparse, meaning that many of its entries are zero, reflecting the principle that only a few library functions are active in the true dynamics of the system. The sparse relationship between the library of functions and the spatial derivative of the state are represented in equation (39):

$$\frac{d\mathbf{U}}{dy} = \Theta(\mathbf{U})\Xi. \tag{39}$$

To identify the coefficient matrix Ξ, wsparse regression techniques are used that promote sparsity while maintaining accuracy in representing the system's dynamics. The approach employed utilises the Forward Regression Orthogonal Least Squares (FROLS) algorithm. This method iteratively selects terms from the function library $\Theta(\mathbf{U})$ based on their contribution to reducing the prediction error. In each iteration, FROLS chooses the term that, when added to the current model, results in the greatest decrease in the sum of squared errors. Compared to other optimisers, FROLS can be adapted to different constraints, which is useful when working with turbulent models. It is relatively robust against noise, allowing it to produce stable and reliable results even when the data has uncertainties. Based on these characteristics, it was chosen because it captures the characteristics of turbulent models better than other optimisers. This process can be conceptually described by the following iterative scheme in equation (40):

$$\xi_{k+1} = \arg\min_{\xi} \left\| \frac{dU}{dy} - \Theta_k(U)\xi \right\|_2^2 \tag{40}$$

where $\Theta_k(\mathbf{U})$ represents the function library at the k-th iteration, including the terms already selected, and ξ is the vector of coefficients corresponding to $\Theta_k(\mathbf{U})$. The process continues until a predefined stopping criterion is met, such as a maximum number of terms or a threshold below which additional terms do not significantly improve the model.

3.3.2 Introduction to PySINDy

Our methodology employs the PySINDy library which is a Python model for constructing interpretable models of dynamical systems. It works especially well for our method, which depends on the function library's richness and the precision of numerical differentiation. The `pysindy.differentiation.FiniteDifference` method is notable for its precision in approximating derivatives from data, which will help us with DNS and RANS data.

The `pysindy.feature_library.Polynomial` library is well-suited for modelling the intricate behaviours of complex systems, enabling us to consider a broad spectrum of potential explanatory variables. The optimization process relies on an algorithm called `pysindy.optimizers.FROLS`, which is designed to identify and retain only the most critical features. This ensures our model is both efficient and powerful, embodying a balance between simplicity and predictive accuracy.

The diagram in figure 26) illustrates the integration of PySINDy's modules within our study. Numerical

differentiation is used for initial data processing, the Polynomial library is used to construct features, and the FROLS optimizer is used to distill the model. This combination yields a model that is elegantly straightforward yet effectively captures the system's dynamics. The effective use of PySINDy's features demonstrates our methodical approach to identifying system behaviours, adhering to the principle of parsimony in scientific modelling.

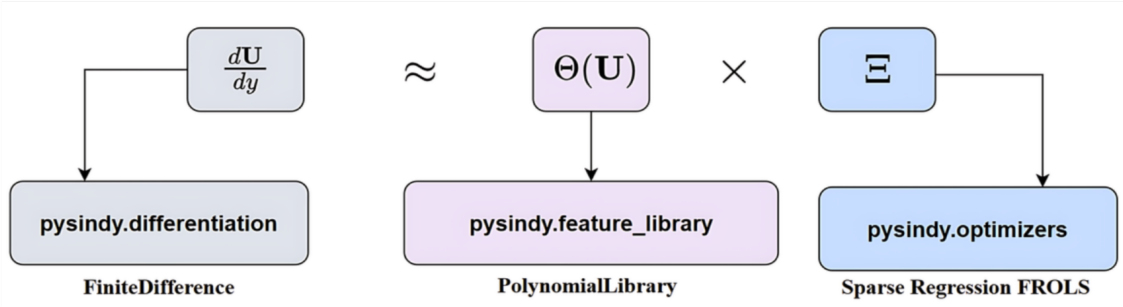

Figure 26: Correspondence between (SINDy) method and the sub-modules of PySINDy

3.3.3 Data Preparation

To use PySINDy for finding the governing equations in the model, it was necessary to prepare the appropriate data. The focus was on analyzing the direct numerical simulations of Channel and Couette flow data, which were made available by the Oden Institute for Computational Engineering and Sciences at the University of Texas at Austin. This data consists of 5 different Channel flow datasets, and 6 different Couette flow datasets, each consisting of 4 .dat data files (Mean of velocity and pressure, Covariances of velocity components, Variances of vorticity components and pressure, and Velocity pressure correlation). All this data has been normalised and only the data of the bottom half of the channel is presented, as some sort of implicit symmetry exists for both the Channel and Couette flow. This symmetry is not guaranteed, and never physically perfect, especially for Couette flow, but the concept of mirroring the data around y = 1 as a vertical symmetry axis will help with data augmentation and help train our PySINDy and Neural Network models. This is a slight trade-off between accuracy and data diversity. Due to limited data availability, the mirroring approach was selected after initial iterations that utilized data from only half the channel's width.

This part also served as an additional layer of data analysis. DNS and RANS data was thoroughly compared as well as the interpolation and mirroring methods and results. Modifications to the methods were made accordingly. For example, some variables required additional horizontal mirroring and some variables had unclear normalisation methods. This also let us detect potential problems with the RANS simulation results as well as evaluate the difference between DNS and RANS data.

For the final approach, the normalisation of RANS data was applied as opposed to the unnormalisation of DNS data to match the PINN model.

3.3.4 Equation Discovery

The project progressed and different approaches were pursued as the pipelines changed and evolved. Three different approaches were studied, and the final approach gave the most robust results and was totally integrated into the final model.

In the first two approaches, an attempt was made to get governing equations that would give direct results from basic features. The idea was to use PySINDy to find governing equations that would provide better results than the current RANS models at a fraction of the DNS computational costs.

To achieve this, the target variables and features to select those equations were first understood. The focus was on the current Eddy viscosity equation (**??**):

$$-\overline{v_i' v_j'} = \nu_t \left(\frac{\partial \overline{v_i}}{\partial x_j} + \frac{\partial \overline{v_j}}{\partial x_i} \right) - \frac{2}{3} k \delta_{ij} \tag{41}$$

In the final approach, the aim was to improve the results of the PINN model by finding governing equations. The focus was on using PySINDy to find momentum equations that were suitable for the PINN model. The first step involved studying the existing RANS equations and simplifying them mathematically based on the configuration of the problem. Under the guidance of Doctor Tamás Józsa, the x and y momentum equations were derived, which are represented by equations (8) and (9) respectively.

$$-\frac{dP}{dx} - \rho \frac{duv}{dy} + \nu \frac{d^2 U}{dy^2} = 0 \tag{42}$$

$$-\rho \frac{dv'v'}{dy} - \frac{dP}{dy} = 0 \tag{43}$$

An idea of the equations to improve using PySINDy, as well as the corresponding features and targets, was established.

3.3.4.1 Approach 1. Covariances Equations

In the pursuit of enhancing the performance of RANS simulations, the establishment of an equation for each covariance stands as a crucial step. Such equations are constructed as polynomials, incorporating velocities and their respective gradients across various derivative orders. This method employs RANS features coupled with DNS targets to discover the governing equations that describe the relationship between velocity gradients and covariances. By doing so, the limitations inherent in RANS simulations are potentially mitigated, bringing their predictive accuracy more in line with the high-quality results of DNS simulations. The model chosen for this study is designed with dual objectives: to predict the Reynolds Stress Tensor and to ensure the symmetric nature of the covariance matrix derived from velocity fluctuations. The Reynolds Stress Tensor, equation (**??**) is expressed as:

$$\tau = -\rho \begin{bmatrix} \langle u'u' \rangle & \langle u'v' \rangle & \langle u'w' \rangle \\ \langle v'u' \rangle & \langle v'v' \rangle & \langle v'w' \rangle \\ \langle w'u' \rangle & \langle w'v' \rangle & \langle w'w' \rangle \end{bmatrix} \tag{44}$$

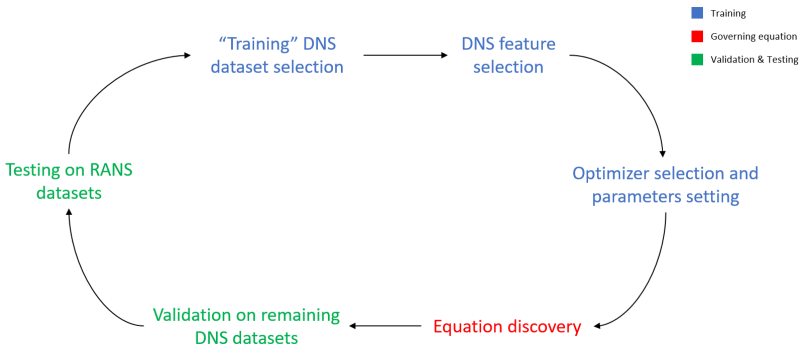

Figure 27: Iterative process

Here, ρ denotes the fluid density. The elements such as $u'u'$, $v'v'$, etc., represent the variances and covariances of velocity fluctuations across the i-th and j-th axes. This matrix's symmetry is crucial to our analysis. Furthermore, given the specific assumptions of the scenario, it is noted that the cross-product terms $u'w' = w'u' = 0$ and $v'w' = w'v' = 0$ hold true.

3.3.4.2 Approach 2. K Equation

The objective of the approach was to derive an equation for the turbulent kinetic energy (k) as a polynomial of velocities and velocity gradients. DNS features and targets were utilized to determine governing equations between velocity gradients and k. This led to the creation of an equation with physical meaning that could be generalized to most cases. Replacing the current k equation used by RANS with a more precise one allowed for improved accuracy in the Boussinesq hypothesis equation (11)) and computation of the covariances.

The basic features used were velocities and velocity gradients, along with three initial parameters; Reynolds number Re_tau, Fiction velocity u_tau, and Kinematic viscosity nu. An iterative training and analysis process was followed to find the best-fitting equation. The aim was to identify RANS predictions that were as close as possible to the DNS results. Figure **??** illustrates this iterative process.

Although the approach was interesting and improved results at low computational costs, its limitations became apparent as the results showed.

3.3.4.3 Approach 3. Momentum Equations for the PINN Model

For this approach, we modified our general interpolation method to work on normalised data. Indeed, the first two approaches were done using unnormalised data to be able to compare easily with the RANS results. However, we found better results with normalised data using PINN, so we needed equations that would work with that data. We also used this opportunity to improve our interpolation and smoothing methods, adding points to our curves as it helps with neural networks training. We also applied those transformations and normalised the RANS data to be able to compare results.

It can be an iterative process to find the best combination of features. We believe creating a loop through different feature combinations was not necessary, and hand selecting them was an appropriate approach. Then, we needed to select the right parameters. These parameters play an important role in controlling the form of the resulting equation: One parameter is linked to the polynomial library and controls the degree, and the two other parameters are linked to the FROLS optimiser and are alpha and

max_iterations, controlling the magnitude of the coefficients in the equation and the maximum number of terms of the equation respectively.

To find the right parameters, we proceeded to a triple nested loop, where for each 3 parameters we have chosen a list of different values that the loops will go through. The parameters for each result are then saved to be able to replicate and visually plot the resulting equation that we judge interesting.

The general approach for this method follows Figure 28, which shows the general process that goes into the discovery of one equation of one target, therefore in our case this process was followed for both x and y momentum equations.

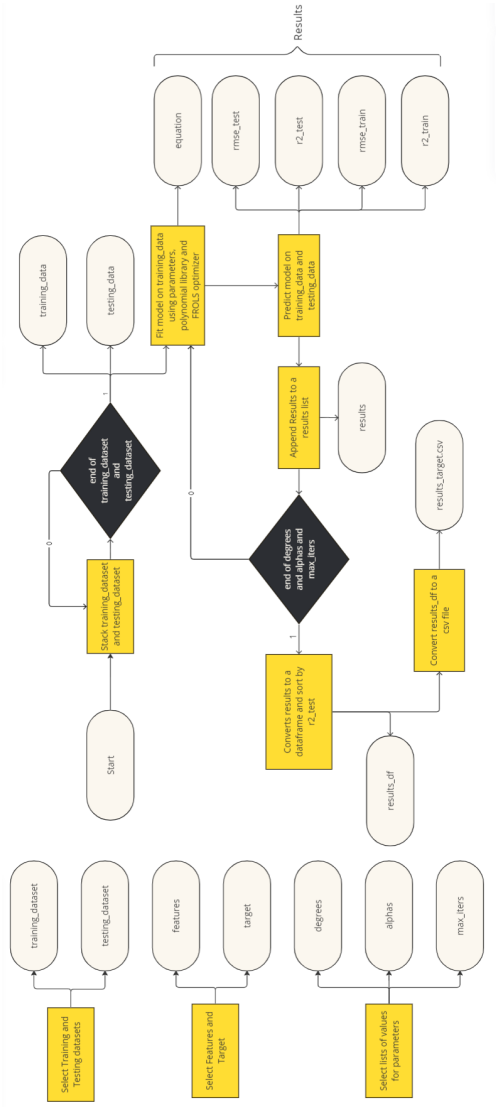

Figure 28: PySINDy Momentum Equation Discovery Flowchart

To begin with, we selected our training and testing datasets:

- For training, we used 3 Channel and 1 Couette setups DNS results. We used Channel flows with

Reynolds numbers of 5200, 1000, and 180 and Couette flow with a Reynolds number of 220 and streamwise domain size Lx = 100 PI.

- For testing, we used 2 Channel and 5 Couette setups DNS results. We tested on Channel flows with Reynolds numbers of 2000 and 550, and Couette flows with Reynolds numbers 93 and 500 for both Lx = 100 PI and 20 PI, and a Reynolds number of 220 for Lx = 20 PI.

All trained and testing data is DNS data. Training data is always DNS data to make sure our equation is the most precise possible and makes physical sense. We also use DNS data for testing and validation to make sure our equation is generalised for other flow configurations.

Listing 20: Train and Test Configurations

```
    train_dataset = {
        "Channel": [180, 1000, 5200],
        "Couette": [220100]
    }
    test_dataset = {
        "Channel": [550, 2000],
        "Couette": [9320, 93100, 22020, 50020, 500100]
    }

```

Next, we selected our target and a set of features. We could have automated the feature selection by including additional nested loops, but we judged it unnecessary given the fact we already had a general idea on which features to use. We hand selected our features and therefore proceeded to a few iterations for each momentum equations.

Listing 21: Feature Selection

```
    selected_features = {
        #'U': True,
        #'dU/dy': True,
        #'d2U/dy2': True,
        #'W': True,
        #'dW_dy': True,
        #'d2W/dy2': True,
        #'P': True,
        #'dP_dx': True,
        'dP_dx_adam': True,
        #'dP_dy': True,
        #'d2P/dy2': True,
        #'u\'u\'': True,
        #'duu_dy': True,
        #'d2uu/dy2': True,
        #'v\'v\'': True,
        #'dvv_dy': True,
        #'d2vv/dy2': True,
        #'w\'w\'': True,
        #'dww_dy': True,
        #'d2ww/dy2': True,
        #'u\'v\'': True,
        'duv_dy': True,
        #'d2uv/dy2': True,
        #'Re_tau': True,
        #'nu': True, #for best test r squared
        #'u_tau': True
```

```
        #'inv_nu': True,
        #'inv_u_tau': True,
    }
    target = 'd2U_dy2'
    rans_target = 'd2xdy2'

```

Finally, we selected our parameters. We created a list of value for each parameters.

Listing 22: Parameter Defenitions

```
    # Polynomial parameter
    degrees = [1,2,3,4,5]
    # FROLS parameters
    alphas = [0.1, 0.01]
    max_iters = [1,2,3,4,5,6,7,8,9,10]

```

Our code trains and evaluates our model for each combination of those parameters, giving the written equation of each model as well as insightful validation metrics such as r squared and RMSE values. Those results are then saved in a table and sorted by descending r squared value for the testing dataset. This table is then converted and exported to a CSV file, where we can evaluate the different equations and how well they fit the DNS data.

Listing 23: Code Overview

```
    # Initialize lists to hold combined training features and targets
    X_train_combined = []
    X_dot_train_combined = []
    X_test_combined = []
    X_dot_test_combined = []

    # Define a list to store models for comparison
    models = []
    performance_metrics = []

    # Dynamically build feature names list based on selected features
    feature_names = [feature for feature in selected_features if selected_features[feature]]

    # Function to construct features based on selected_features dictionary
    def construct_features(int_data):
        features = np.hstack([int_data[[feature]].values for feature in selected_features if selected_features[feature]])
        return features

    # Process each dataset for training and testing
    for dataset_type, dataset in {'train': train_dataset, 'test': test_dataset}.items():
        for flow_type, ids in dataset.items():
            for id in ids:
                filename = channel_int_augmented_datafilenames[id]

                if flow_type == "Channel"
                else
                couette_int_augmented_datafilenames[id]
                params_df_selected = params_df if flow_type == "Channel" else couette_params_df
                params = params_df_selected[params_df_selected[flow_type.lower()] == id].iloc[0]

                int_data = pd.read_csv(filename)

                # Append simulation parameters as new columns
                for param in ['Re_tau', 'nu', 'u_tau']:
```

```
                int_data[param] = params[param]
            int_data['inv_nu'] = 1/int_data['nu']
            int_data['inv_u_tau'] = 1/int_data['u_tau']

            # Construct features
            features = construct_features(int_data)

            # Extract target values for the model
            if 'd2U_dy2' in int_data.columns:
                targets = int_data[[target]].values
            else:
                continue  # Skip this dataset if the target variable isn't available

            # Depending on the dataset type, append data to the corresponding list
            if dataset_type == 'train':
                X_train_combined.append(features)
                X_dot_train_combined.append(targets)
            elif dataset_type == 'test':
                X_test_combined.append(features)
                X_dot_test_combined.append(targets)

# Convert lists of arrays to single numpy arrays
X_train_combined = np.vstack(X_train_combined)
X_dot_train_combined = np.vstack(X_dot_train_combined)
X_test_combined = np.vstack(X_test_combined)
X_dot_test_combined = np.vstack(X_dot_test_combined)

results = []  # List to store the results

# Iterate over each combination of degree, alpha and max_iter
for degree in degrees:
    for alpha in alphas:
        for max_iter in max_iters:
            # Initialize and fit the model
            optimizer = ps.FROLS(normalize_columns=False, alpha=alpha, max_iter=max_iter)
            feature_library = ps.PolynomialLibrary(degree)
            model = ps.SINDy(optimizer=optimizer, feature_library=feature_library, feature_names=feature_names)
            model.fit(X_train_combined, x_dot=X_dot_train_combined)

            # Predict and compute metrics on training set
            predicted_dot_train = model.predict(X_train_combined)
            r2_train = r2_score(X_dot_train_combined, predicted_dot_train)
            rmse_train = np.sqrt(mean_squared_error(X_dot_train_combined, predicted_dot_train))

            # Predict and compute metrics on testing set
            predicted_dot_test = model.predict(X_test_combined)
            r2_test = r2_score(X_dot_test_combined, predicted_dot_test)
            rmse_test = np.sqrt(mean_squared_error(X_dot_test_combined, predicted_dot_test))

            # Redirect stdout to capture the equation print
            old_stdout = sys.stdout
            sys.stdout = io.StringIO()
            model.print()
            equation = sys.stdout.getvalue()
            sys.stdout = old_stdout
            equation = ' '.join(equation.split())

            # Store results
            results.append([degree, alpha, max_iter, equation, r2_train, rmse_train, r2_test, rmse_test])
            print(f"Model with degree={degree}, alpha={alpha}, max_iter={max_iter}: R_train={r2_train}, R_test={r2_test},
            RMSE_train={rmse_train}, RMSE_test={rmse_test}")

# Convert results to DataFrame and sort by R2 Test in descending order
columns = ['Degree', 'Alpha', 'Iters', 'Equation', 'R2 Train', 'RMSE Train', 'R2 Test', 'RMSE Test']
results_df = pd.DataFrame(results, columns=columns)
```

```
    results_df = results_df.sort_values(by='R2 Test', ascending=False)

    # Save to CSV
    csv_filename = f"results\model_results_final2_{target}.csv"
    results_df.to_csv(csv_filename, index=False)

```

To confirm the validation from the r square and RMSE metrics, we further used visual validation, plotting the PySINDy results versus DNS and RANS data for each testing setup for a given parameter configuration. Our previous table that ranks the best equations also notes for which parameters it was computed, allowing quick and easy plotting. We then analysed the results from the mathematically derived RANS momentum equations to compare against our PySINDy discovered equations. We also implemented a Genetic Algorithm model to validate our models, and plotted its results to compare with the DNS results. To have a more general idea, we built a final plot that represents all the results from the testing dataset for PySINDy, DNS and RANS normalised, averaged, and compared.

3.3.5 Validation and Testing

3.3.5.1 RMSE

Evaluation, testing and validation were done first by evaluating the models on the training dataset with metrics such as r squared and RMSE, and then by evaluating those same models but on the testing dataset both by using the r squared and RMSE results and visually by analysing the predicted results from our equations against the actual DNS results and the original RANS results. For Approach 2: Equation Discovery for K, we also analysed the predicted results from k before analysing the computed covariances with the new Boussinesq equation as another step of testing.

3.3.5.2 Genetic Algorithm

Genetic algorithms were used to conduct the validation of a new governing equation from PySINDy. In this approach, the initial population of possible governing equations was generated by initialising the coefficients with a variance, allowing for a wide range of starting points within the solution space. To maintain a diverse population and avoid premature convergence, a mutation rate was applied, ensuring slight variations in the coefficients during each generation. The crossover operation, which involved three parents, allowed for a broader exploration of the solution space, combining traits from multiple individuals to create new equations. This setup facilitated an iterative process where the best-fit governing equations were selected based on a fitness function, allowing the most promising solutions to emerge and evolve over successive generations. The combination of variance-based initialisation, mutation, and three-parent crossover ensured that the genetic algorithm could efficiently validate and refine potential governing equations derived from PySINDy, leading to robust and accurate results. The genetic algorithm configuration is seen in table 16.

Parameter	Value
Population	200
Mutation	0.01
Generations	20
Crossover	three-parents
Selection	tournament
Fitness Function	RMSE

Table 5: Model Configuration for the Genetic Algorithm

With this approach, these benefits can be expected when validating a new governing equation from DNS data. The robust optimisation provided by Genetic Algorithms (GAs) allows for an extensive exploration of possible solutions, ensuring that the best-fit model is identified even in complex landscapes. The flexibility to incorporate various constraints, such as those enforcing physical laws or specific structures, adds rigour to the validation process. Additionally, GAs' resistance to local optima and ability to handle multi-objective optimisation contribute to creating models that are both accurate and efficient. The maintenance of a diverse population through stochastic operations reduces the risk of premature convergence, fostering a broader range of solutions. This resilience, combined with automated hyperparameter tuning, ensures that the optimal configuration is achieved with minimal manual intervention, leading to a scalable and efficient validation process. As a result, this approach offers a compelling framework for validating governing equations derived from DNS data, combining flexibility, robustness, and efficiency.

3.4 Implementing a Simple Neural Network

The first attempt at predicting the RSTs made use of a simple neural network. This preliminary approach was chosen to to see how well a relatively straightforward implementation of machine learning would handle the provided DNS data and make predictions based on it. At the time of when the model was created, the interpolation data did not yet exist, therefore the model was trained and tested only on the raw Channel and Couette flow files. For the Couette flow, Reynolds numbers 93, 220 and 500 for both 20 and 100 data points were included and for the Channel flow, Reynolds number 180, 550, 1000, 2000 and 5200 were included. Initial iterations of the simple neural network split the model into two version, one training and predicting only on Channel Flow, and the other only on Couette flow, however the most refined version of the model incorporated both flow types into the training and testing pools to create a single model to predict either flow type.

3.4.1 Defining the Keras Model

A sequential Keras model was used due to the straight forward setup as illustrated in the code for the Keras model setup:

Listing 24: Keras Model Defenition

```
1 def trainModel(X_train, y_train):
2     model = Sequential()
3     model.add(Dense(32, input_dim = 2, activation='relu'))  # Input dimension changed to 2
```

```
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adamax')
    model.fit(X_train, y_train, epochs = [10, 50, 100], batch_size = 1, verbose = 1)
    return model
```

The model takes in two input dimensions for each stress tensor, and produces a prediction for each. For each inputted file into the model 6 prediction are made corresponding to each stress tensor: uu, uv, vv, ww, uw and vw. The training and testing was done on all the DNS files and a split model was used for a 80% training and 20% testing split. The specifics of the model setup are outlined in table 6.

Table 6: Keras Model Specifications

Component	**Details**
Input Layer	Input dimension: 2 features
Hidden Layer 1	Number of neurons: 32 Activation function: ReLU
Hidden Layer 2	Number of neurons: 32 Activation function: ReLU
Output Layer	Number of neurons: 1 Output: Single stress tensor component
Loss Function	Mean Squared Error
Optimizer	Adamax
Training Parameters	Epochs: 10, 50, 100 Batch size: 1 Verbose: 1 (progress updates enabled)

The Keras model uses the Mean Squared Error (MSE) to evaluate model performance and this relationship is outlined in equation (45)

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)})^2 \tag{45}$$

In Keras, the MSE is a commonly employed metric for assessing regression model performance. It calculates the average of the squared differences between predicted and actual values, providing a simple yet effective measure of how well the model is performing. MSE is advantageous in Keras due to its compatibility with optimisation algorithms like gradient descent. Furthermore, its ability to penalise larger errors more heavily makes it suitable for scenarios where outliers may be present in the data.

3.4.2 K Optimisation

Such as the second equation discovery pySINDY approach, it was decided that a neural network should attempt to optimise K from RANS. The optimisation of the turbulent kinetic energy (K) parameter involved training a neural network model using aggregated data from various simulations. The process beared the following steps:

1. **Data Preprocessing**: Each simulation data file was processed by calculating the second derivative of velocity over the y-direction using finite differences. Adjustments were made to the velocity field if

necessary, such as subtracting 1 from the 'xvel' column for Couette flow simulations.

2. **Data Aggregation**: The processed data from all simulations was combined into a single dataset.

3. **Dataset Preparation**: The aggregated dataset was divided into input features (X) and the target variable (y), where X consisted of the following features:

   - **Grid Spacing ($dzdz$)**: The distance between grid points in the computational domain was represented. Grid spacing influenced the accuracy of numerical simulations and impacted the distribution of turbulence.
   - **Velocity ($xvel$)**: Flow field characteristics, including velocity gradients and turbulence intensity, were reflected. Variation in velocity affected turbulent kinetic energy distribution.
   - **Turbulent Viscosity (t_visc)**: The eddy viscosity in turbulent flows was represented, essential for capturing energy transfer mechanisms. Turbulent viscosity influenced turbulence model performance and turbulence characteristics.

4. **Model Training**: The dataset was split into training and testing sets. A neural network model, MLPRegressor, was trained with specified architecture (e.g., hidden layer sizes, activation function, solver) using the training data.

5. **Model Evaluation**: The trained model was evaluated using the testing data by calculating the MSE between the predicted and actual K values and by visually observing the plots.

6. **Prediction and Validation**: The trained model was utilised to predict K values for additional simulations not included in the training data. DNS K, original RANS K, and predicted RANS K were plotted for validation purposes.

The neural network model architecture used for the K optimisation is laid out in table 7:

Parameter	Value
Hidden Layer Sizes	(1000, 1000)
Activation Function	ReLU
Solver	Adam
Maximum Iterations	100000
Random State	42

Table 7: K Optimisation Neural Network Model Specifications

This machine learning K Optimisation approach was disregarded however due to it being redundant due to limitations in the Bousinessq Hypothesis, more specifically local isotropic turbulence, resulting in the same curves being generated for normal stresses. Despite this, the model prediction of K on DNS data as the ground truth and using RANS data as the input is seen in Appendix C2.

3.5 Implementing a PINN

3.5.1 PINN for Enhancing RANS Result

RANS simulations offer computational efficiency compared to DNS, making them the more attractive choice for many applications. However, they possess limitations since they often give results that deviate from the higher-fidelity DNS simulations. To address these limitations machine learning models are leveraged to enhance the accuracy of RANS predictions - this is provided that they have the correct inputs and a well defined loss function.

The selected neural network model for this problem is designed with two objectives. First, predict the Reynolds Stress Tensor, equation (46):

$$\tau = \rho \cdot \begin{bmatrix} u'_x u'_x & u'_x u'_y & u'_x u'_z \\ u'_y u'_x & u'_y u'_y & u'_y u'_z \\ u'_z u'_x & u'_z u'_y & u'_z u'_z \end{bmatrix} \tag{46}$$

where, ρ corresponds to the fluid density. The elements τ_{ij} for i, j in $\{1, 2, 3\}$ represent the variances and covariances of the velocity fluctuations in the i-th and j-th directions. In addition, it is worth noting that this matrix is symmetric and based on the scenario's assumptions, the following components have null value: $u'_z u'_x = u'_x u'_z = 0$ and $u'_z u'_y = u'_y u'_z = 0$.

Next, infer the turbulent kinetic energy since there is a fundamental relationship between the variances which are components of the Reynolds Stress Tensor - this is expressed in equation (47):

$$k = \frac{1}{2}\left(u'u' + v'v' + w'w'\right) \tag{47}$$

Through this approach, the aim is to bridge the gap between RANS predictions and DNS results through the utilisation of machine learning.

3.5.2 Defining The Loss Function

One of the most important aspects when it comes to employing a PINN is to fully define the loss function. It is through the use of the loss function that physics patterns and laws can be introduced to the model. In this scenario, the loss function is seen in equation (48):

$$\mathrm{MSE}(\Theta) = \mathrm{MSE}_{\mathrm{GT}}(\Theta) + \mathrm{MSE}_{\mathrm{BC}}(\Theta) + \mathrm{MSE}_{\mathrm{PDE}}(\Theta) \tag{48}$$

The model's loss function is composed of three losses. The first loss corresponds to the basic mean square error (MSE) between the predicted Reynolds Stress Tensor and the Ground Truth Reynolds Stress Tensor, equation (49):

$$Loss_{GT} = RST_{NN} - RST_{GT} \tag{49}$$

The second loss, equation (50) corresponds to the boundary conditions:

$$Loss_{BC} = RST(0)_{NN} - RST(0)_{GT} \tag{50}$$

The third loss corresponds to the physical aspect as it uses the mean square error on both the momentum equation in the x-direction and y-direction which come from the RANS equation - it is seen in equations (51) and (52) respectively:

$$Loss_{momentum_x} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \nu_t \frac{\partial^2 U_x}{\partial y^2} - \frac{\partial \langle u'_x u'_y \rangle}{\partial y} \tag{51}$$

$$Loss_{momentum_y} = -\frac{1}{\rho}\frac{\partial P}{\partial y} - \frac{\partial \langle u'_y \rangle^2}{\partial y} \tag{52}$$

This third loss changes when using the PySindy equations, as explained in the past sections (equation **??**):

$$Loss_{PySindy} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \nu_t \frac{\partial^2 U_x}{\partial y^2} - \frac{\partial \langle u'_x u'_y \rangle}{\partial y} \tag{53}$$

The same concept applies when using the model to predict the Couette flow as described in equation (54):

$$Loss_{PySindy} = -\frac{\partial^2 U_x}{\partial y^2} + \frac{\partial \langle u'_x u'_y \rangle}{\partial y} - 0.001 \tag{54}$$

3.5.3 Regularisation

A conventional deep neural network model needs to be fed with a huge amount of data so it can capture complex patterns and be able to generalise. The PINN allows to prevent the lack of data but can be further optimised by adding regularisation.

Different methods exist in order to implement regularisation which can, for instance, prevent from overfitting. For the chosen model, L2 regularisation also known as Ridge regularisation has been selected. The L2 regularisation adds a penalty term to the loss function penalising large coefficients by computing the square of the magnitude of the coefficients as shown in equation (55):

$$\text{Loss}(\theta) = \text{Loss}(\Theta) + \lambda \cdot \mathbf{\Omega}(\theta) \tag{55}$$

where, λ represents the strength parameter of the regularisation. A low value of this parameter means a weaker regularisation and vice-versa. θ represents the coefficient of the neural network model. Ω represents the regularisation function, where in this study, it is supposed to be the squared L2 norm of the coefficient θ.

3.6 Custom PINN Model

The main aim of the custom neural network is to predict the Reynolds Stress Tensor. Note that there are two different types of flows, Channel and Couette, therefore the model is expected to generalise and to be able to predict these two flows once trained using both Channel and Couette flow data.

3.6.1 PINN Model Architecture

The opted architecture is a residual neural network. This architecture allows complex relationships to be captured within the data by using residual connections or skip connections as illustrated in figure 29. This facilitates the flow of information through the network and helps mitigate the vanishing gradient problem commonly encountered in deep neural networks.

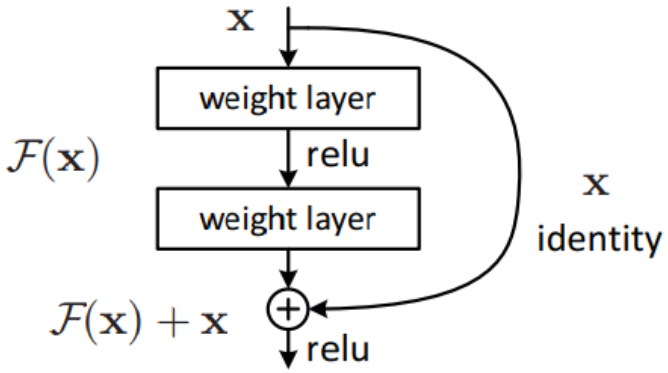

Figure 29: A Residual Connection

As depicted in figure 30, the skip connections create shorter paths that allow gradients to flow not only through all the layers of the network but through the skip connection paths as well. This enables efficient updates by providing alternate paths for gradient propagation, which in turn helps reduce the vanishing gradient problem. As a result of this, during the training process the model's parameters are updated more reliably and informatively which not only improves the stability of the training process but also ensures that the model can effectively capture complex patterns and relationships within the data.

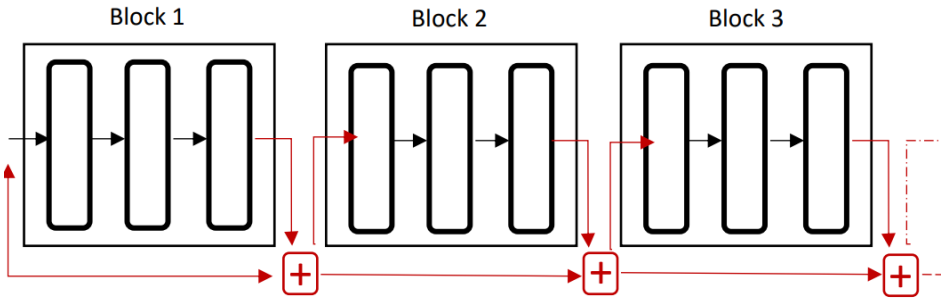

Figure 30: The Residual Block

3.6.2 PINN Model Tuning

When customising a neural network, fine-tuning hyperparameters is an essential task in order to maximise its accuracy. The most important parameters for optimisation typically include the learning rate, the choice of optimiser and number of epochs. The learning rate is the parameter that enables the Gradient Descent algorithm to progressively converge to the minimum point on the curve facilitating optimal adjustments to the model's parameters. Besides, the learning rate can also be fine tuned thanks to optimisers such as Adam, or its variants, which is an adaptative learning rate optimisation algorithm. It leverages data properties to efficiently find the best value for it dynamically while training. Finally, the number of epochs determines how comprehensively the model can capture data patterns by iterating through the entire training set. However, it is important to select the number of epochs judiciously as it can lead to overfitting if too high or lead to underfitting if the value is too low.

3.7 PINN Pipeline

Following the selection of the hyperparameters and defining the architecture of the custom neural network, the last step involves integrating everything. The custom neural network is coupled with the established equations (27) and (28). This integration completes the Physics Informed Neural Network model as illustrated in Figure 31.

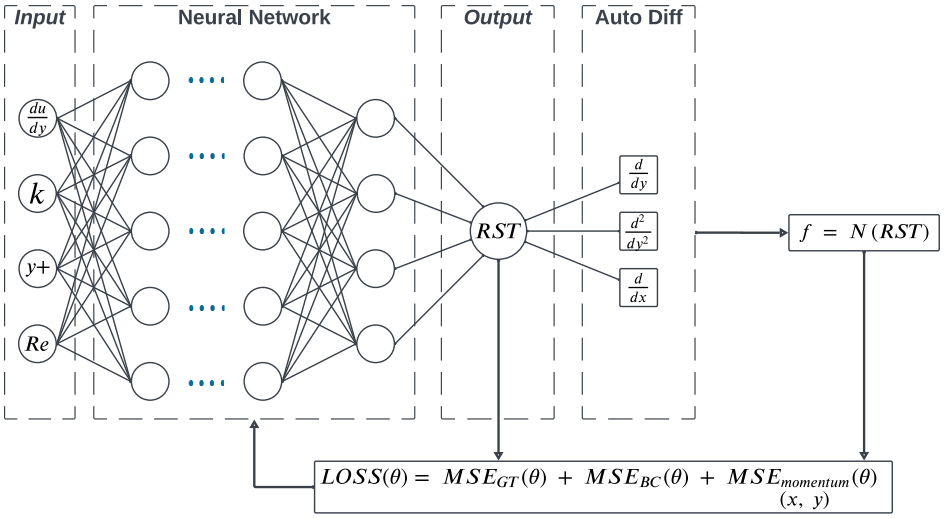

Figure 31: The PINN model

3.8 PINN Data Preprocessing

Data preprocessing plays an essential role in ensuring the efficacy of our model. In this scenario, the dataset encompasses two distinct flow types: Channel flow and Couette flow. Within each flow type, 5 sets of data originating from DNS simulations are available.

The initial step involved the extraction of the data needed for the model to be trained. The dataset contains several columns which are not used by the model, thus, only the columns needed were extracted as shown in Figure 32 in a Numpy array. The boxes include extra data than such as the pressure, mean

velocity, kinematic viscosity, streamwise domain size columns as they are used for computing the momentum equations when using the RANS based PINN. It is then converted into a tensor so it can be used by the PINN model. Regarding the additional data, extra operations needed to be done, particularly on renaming the name of columns in order to match those of the already used data. One additional operation was to set two components from the Reynolds Stress Tensor to zero to respect the scenario's assumption.

Used for training part

	y^+	dU/dy	k	P	U	Re	nu	Lx
0	0.000000	1.000000	2.203288e-34	0.000000e+00	0.000000	550	0.0001	25.12
1	0.002696	0.999995	8.414171e-07	-7.394299e-15	0.002696	550	0.0001	25.12
2	0.013479	0.999975	2.101485e-05	-4.606107e-12	0.013478	550	0.0001	25.12
3	0.037739	0.999931	1.643998e-04	-2.810221e-10	0.037738	550	0.0001	25.12
4	0.080868	0.999851	7.520443e-04	-5.847961e-09	0.080862	550	0.0001	25.12
...	...	...	...	...	...	...	...	...
378	1082.382843	0.999851	7.520443e-04	-5.847961e-09	0.080862	550	0.0001	25.12
379	1082.425972	0.999931	1.643998e-04	-2.810221e-10	0.037738	550	0.0001	25.12
380	1082.450233	0.999975	2.101485e-05	-4.606107e-12	0.013478	550	0.0001	25.12
381	1082.461016	0.999995	8.414171e-07	-7.394299e-15	0.002696	550	0.0001	25.12
382	1082.463711	1.000000	2.203288e-34	0.000000e+00	0.000000	550	0.0001	25.12

Figure 32: Selected Model Features

The second task, however, was challenging on account of the incomplete nature of the dataset. Each data entry only provided half of the flow profile, as depicted in figure 33. Consequently, it was imperative to perform a mirroring process to augment and complete the dataset.

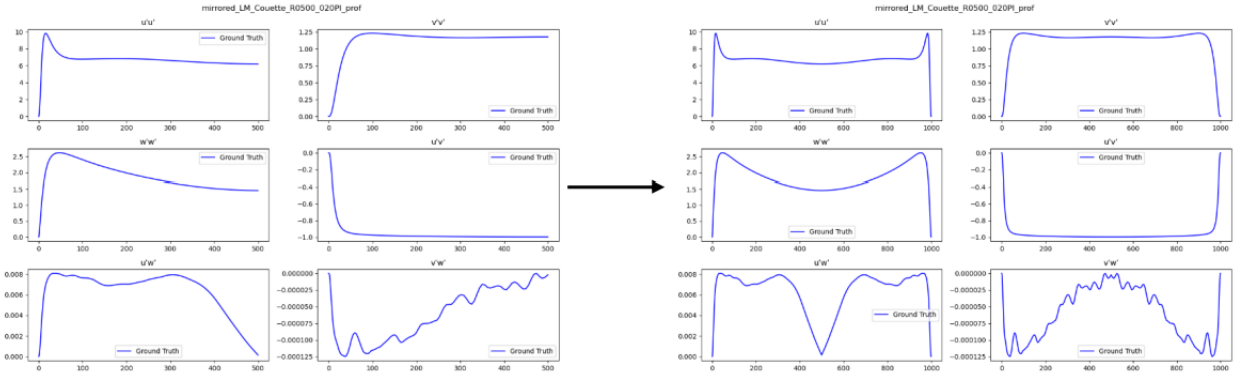

Figure 33: Data Mirroring

This data preprocessing step is specifically tailored for implementing a RANS-based PINN model. In contrast, when employing a PySindy-based PINN model, the dataset goes through a distinct preprocessing process including interpolation, smoothing, and most importantly, completion through the addition of derivatives. This completed dataset is then used within the loss function when addressing the PDE loss term.

4 Results

4.0.1 CFD RANS Results

4.0.1.1 Qualitative Comparative of Viscous Models

On the viscous model selection, a brief qualitative comparison was performed to compare the results obtained for the Reynolds 182 and 5200 Channel flow for all RANS viscous models computed. However, the main aim of that section was to establish the viscous model that it would be used as a comparison to the results obtained with the new frameworks, so the discussion was focused more on numerical accuracy rather than the qualitative behaviours of the models.

Firstly, it is important to note that all the plots comparing the viscous models were plotted against the dimensional wall distance y^+ rather than the standard wall distance. This value is independent for each simulation as it is proportional to the Friction Reynolds Numbers. Additionally, the logarithmic scale in base 10 has been performed on the x-axis, corresponding to the y+ value, to be able to observe the near-the-wall region.

If the velocity profile is analysed, it can be observed that for the two Reynolds Case math the same pattern is shown in Figure 34, where the velocity field in the Near the wall region was displayed. By examining the mean streamwise velocity profile, it can be seen that all viscous models accurately predict the same behaviour as the DNS dataset. This profile is characterised by three main sections as described by the Law of the Wall. Firstly, from $y+ = 1$ up to $y+ = 30$, corresponding with the viscous sublayer, the dimensionless velocity matches the dimensionless wall distance in equation (56):

$$U^+ = y^+ \tag{56}$$

In a normal plot, this behaviour will be characterised by a straight line, however, in a semi-logarithmic plot, the relation is a curve. Additionally, in the buffer layer, for $30 > y^+ > 120$, the prediction of RANS differs from the DNS. Also, it can be seen that for the log-law region, corresponding to $y^+ > 120$ the velocity has a linear behaviour and can be approximated as equation (57):

$$U^+ = \frac{1}{\kappa} * ln(y^+) + \beta^+ \tag{57}$$

Lee and Moser, creators of the DNS dataset used in this project, estimated the values of κ and β to be 0.384 ± 0.004 and 4.27 respectively for the 5200 Reynolds Case. Although predicting these values for each viscous model was out of the scope of this project, it can be observed that the models that better predicted the flow, were the standard and SST k-omega models, especially for the Reynolds 182 case, seen in figure 34 which matches the error obtained and displayed in Table 9.

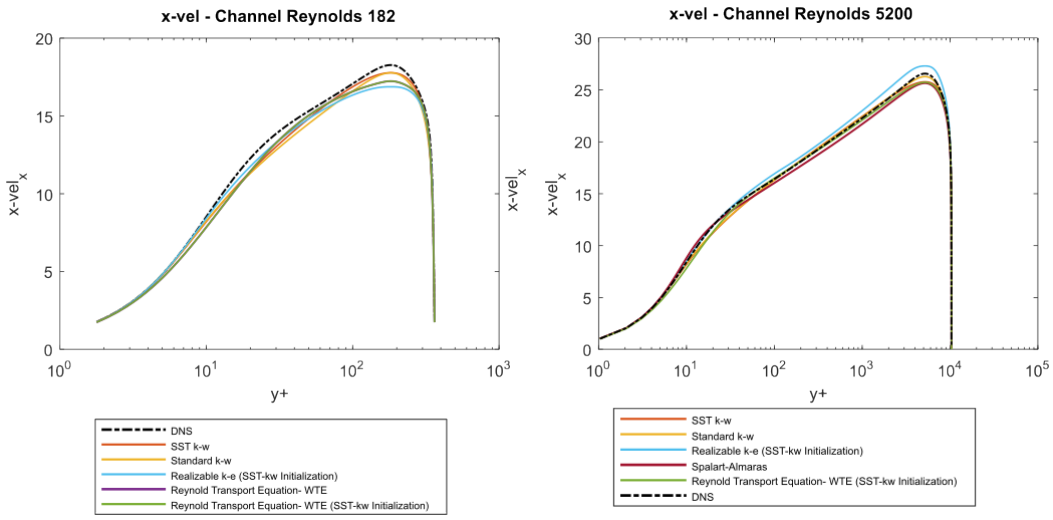

Figure 34: Mean Velocity Profile Comparison from different RANS Viscous Models for Reynolds 182 and Reynolds 5200 Channel Flow

When comparing the $u'v'$ results with a logarithmic plot, a better understanding of the predictions of the different viscous models can be made. Firstly, it can be observed that for the viscous sub-layer region, the RTM and realisable k-epsilon models had better predictions due to the implementations of wall treatment enhancement and standard wall functions. However, as we get far away from the wall, the same behaviour explained in the previous section can be observed where the RTM model provided inaccurate predictions and fluctuated along the y distance. Focusing on the log layer, the results were almost identical to the DNS, unlike the buffer layer where the results were less accurate, especially in the Reynolds 182 case, figure 35.

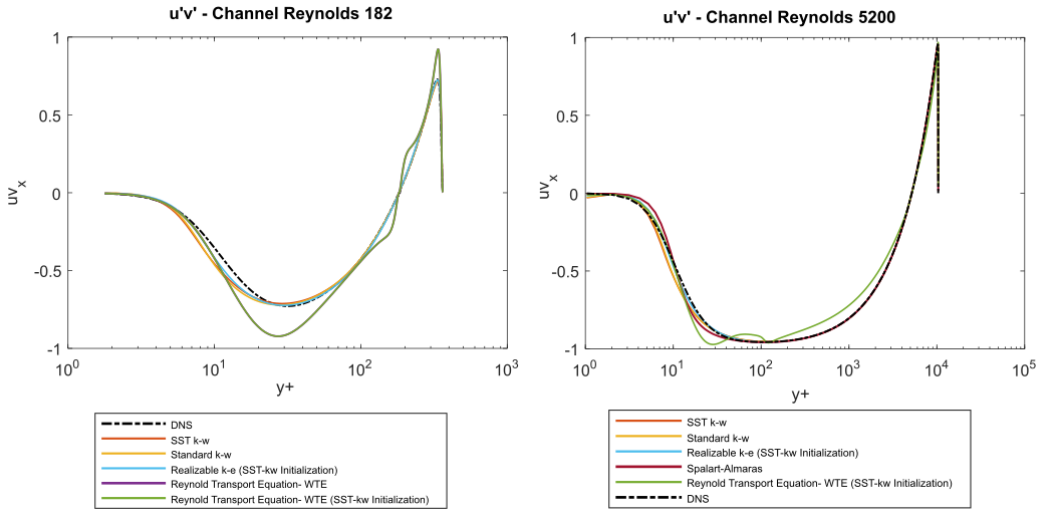

Figure 35: $u'v'$ Comparison from Different RANS Viscous Models for Reynolds 182 and Reynolds 5200 Channel Flow

Finally, for the Normal Stresses it can be seen that the same behaviour as explained in the previous section, where due to the heavy influence that the prediction of the turbulent kinetic energy has in the calculation of these values as well as the condition of local isotropic turbulence that characterises the Boussinesq Hypothesis, the results are highly inaccurate, although the behaviour of the curves determined by a steep gradient and then a decrement of the value around y+ is depicted in figure 36.

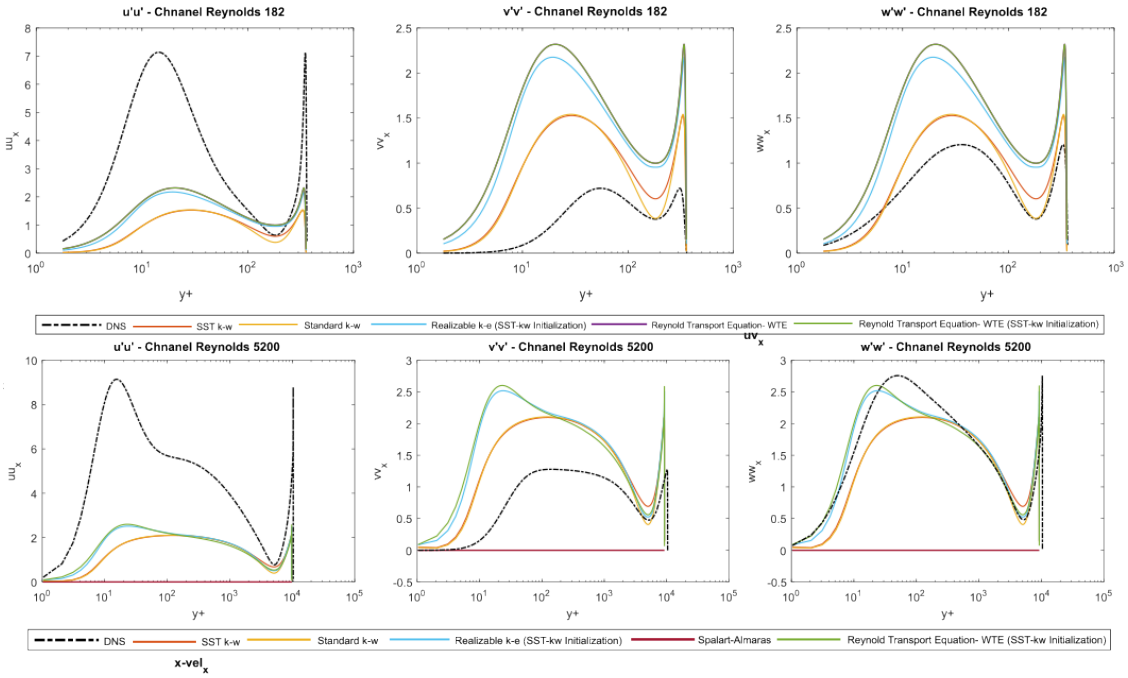

Figure 36: Normal Stresses Comparison from Different RANS Viscous Models for Reynolds 182 and Reynolds 5200 Channel Flow

4.0.1.2 RANS SST K-omega Results for All Cases

To obtain the SST kw RANS predictions of the covariance of the Reynolds Stress Tensor, which will serve as a benchmark for comparison for the models explained before, the code for automatising the Channel CFD set-up, was executed for all 5 cases of (Reynolds 180, Reynolds 500, Reynolds 1000, Reynolds 2000 and Reynolds 5200) and 3 cases of Couette Flow (Reynolds 90, Reynolds 220 and Reynolds. For each simulation, the boundary conditions and fluid properties were obtained from the dataset and are summarised in table **??**.

Name	Reynolds Number	Kinematic Viscosity	Friction Velocity	Pressure Gradient	Top Wall Velocity (m/s)	Bottom Wall Velocity (m/s)
Channel 180	182.08	3.5E-4	0.0637	0.00406	0.0	0.0
Channel 550	543.49	1.0E-4	0.0543	0.00295	0.0	0.0
Channel 1000	1000.51	5.0E-5	0.0500	0.00250	0.0	0.0
Channel 2000	1994.75	2.3E-5	0.0459	0.00210	0.0	0.0
Channel 5200	5185.89	1.0E-8	0.0415	0.00172	0.0	0.0
Couette 93	92.91	6.7E-4	0.0619	0.0	2.0	0.0
Couette 220	219.47	2.5E-4	0.0549	0.0	2.0	0.0
Couette 500	501.37	1.0E-4	0.0501	0.0	2.0	0.0

Table 8: Boundary Conditions and Fluid Properties for all Channel and Couette Flows

Each simulation was converged to a value of $1e-08$ in all residuals monitors and the Reynolds Stresses were computed with the standard Boussinesq Hypothesis.

Channel Flow

When observing the Channel Flow results plots, it becomes evident that the same behaviours explained in the previous section are present. The most accurate predictions were obtained for the $u'v'$ term, while there were significant discrepancies in the prediction of the normal stresses compared to the ones in the DNS dataset, generated by the local turbulent isotropic assumption and the inaccurate prediction of the turbulent kinetic energy. Furthermore, it can be observed in figure 37 that for all Reynolds Numbers studied, RANS underestimated the velocity, especially in the free stream area.

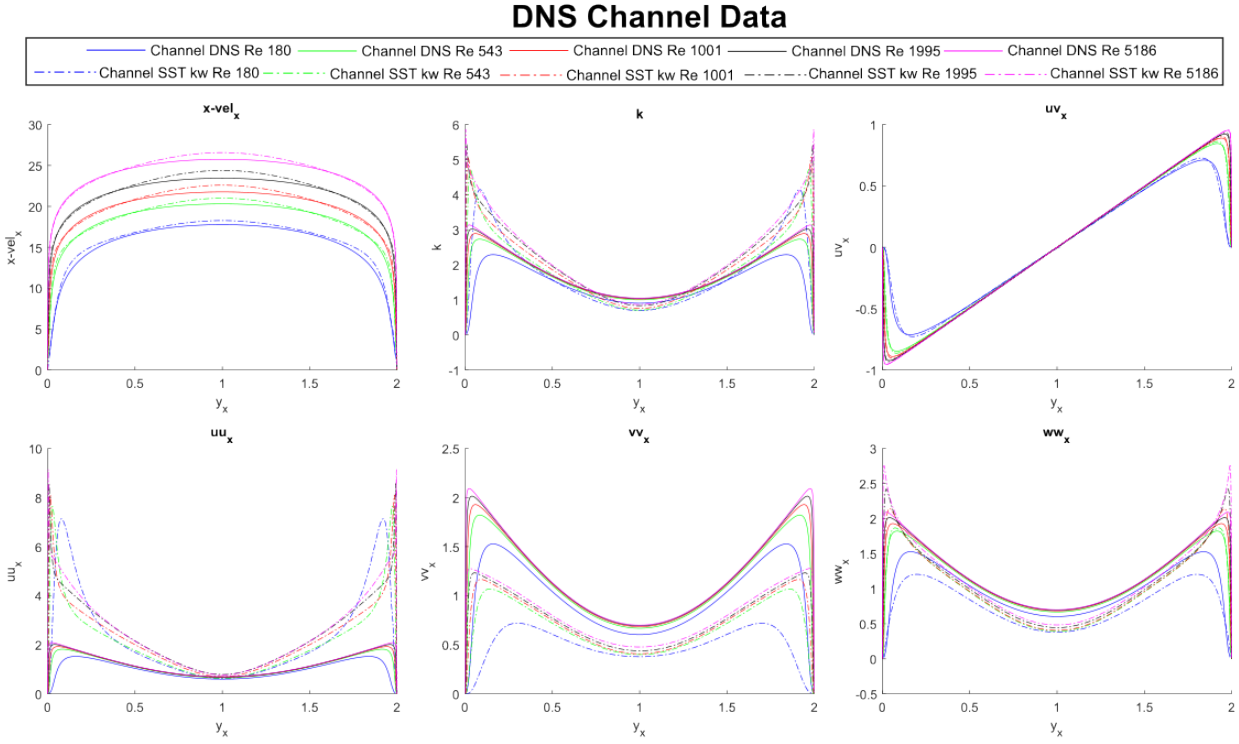

Figure 37: SST k-w and DNS Comparison for all Chanel Flows (182, 550, 1000, 2000 and 5200 Reynolds Number)

The RMSE of the SST kw and DNS values, displayed in table 9, reveals several tendencies. Firstly, if we focus on each separate variable, the covariance for the normal stress in the X direction ($u'u'$) has the biggest error followed by the turbulent kinetic energy, the reason behind this big error could be that the DNS values of this components are higher than the normal stresses in the Y and Z direction, as both $v'v'$ and $w'w'$ have a similar error as the velocity. However, it is important to note that the velocity ranges much higher values $(0-25)$ than the stresses $(0-2.5)$. Finally, the predictions for the $u'v'$ were the most accurate. The reason behind these accurate results is based on the fact that this component does not rely upon the prediction of the turbulent kinetic, and only the X-Velocity gradient on y. So by having a highly refined mesh in y and setting the simulation with a very small convergence rate, very good results were obtained in this variable.

On the other hand, if the cases are looked at globally, the accuracy in the predictions increases with the Reynolds number, indicating that the RANS setup aims to predict better flows with high turbulence rather than more laminar. Another option could be the refinement of the mesh, as the cases with higher Reynolds numbers had a more refined mesh due to the calculation of the minimum wall distance, which was computed with the use of the equation (32), and stated that the minimum wall distance was indirectly proportional to the Reynolds Number, by assuming a constant y^+ and density. it also can be observed that the gaps between cases decrease as the Reynolds Number increases. This can be better showcased if the gap of error for $u'u'$ between Re 182 and Re 543 and the gap between Re 1995 Re 5186 are compared. For the former, there is an increment in the Reynolds number by 361, with a decrement in the error of 0.544. For the latter, there is a decrement of error of 0.082 for an increment of the Reynolds number by 3191.

	Re 180	Re 543	Re 1001	Re 1995	Re 5186
U	0.452	0.378	0.438	0.516	0.431
uu	2.070	1.526	1.515	1.594	1.676
uv	0.022	0.013	0.010	0.007	0.006
vv	0.572	0.507	0.488	0.470	0.448
ww	0.238	0.180	0.195	0.187	0.171
k	0.725	0.563	0.550	0.582	0.642

Table 9: RMSE for all Channel Flow Cases

Couette Flow

In the analysis of three cases of Couette flow, similar behaviours were observed, but with lower accuracy. The velocity predictions were found to be accurate in the region nearest to the wall and the middle section, but there were some mismatches between them. One of the main differences between the Channel and Couette Flows predictions is the accuracy obtained for the $u'v'$ component; whereas the channel predictions matched almost perfectly the DNS values in all the range of the channel, the Couette RANS predictions perfectly matched the near the wall region however they deviated from the DNS values in the middle section, obtaining a similar behaviour but with an offset. Furthermore, it can also be seen that all three DNS cases converge at the same value, around -1 and similarly, the RANS data also converges to a value of around -0.85 creating a gap compared to DNS.

Examining the normal stresses and turbulent kinetic energy plots, all values plotted corresponding to the RANS prediction of the four variables were the same as seen in figure 38 As it occurred in the channel flow, the values of the normal stresses are defined exclusively by the turbulent kinetic energy, as the normal velocity gradients are 0. As explained before, the addition of the turbulent isotropy as well as the inability to correctly predict the value of k, generates this discrepancy.

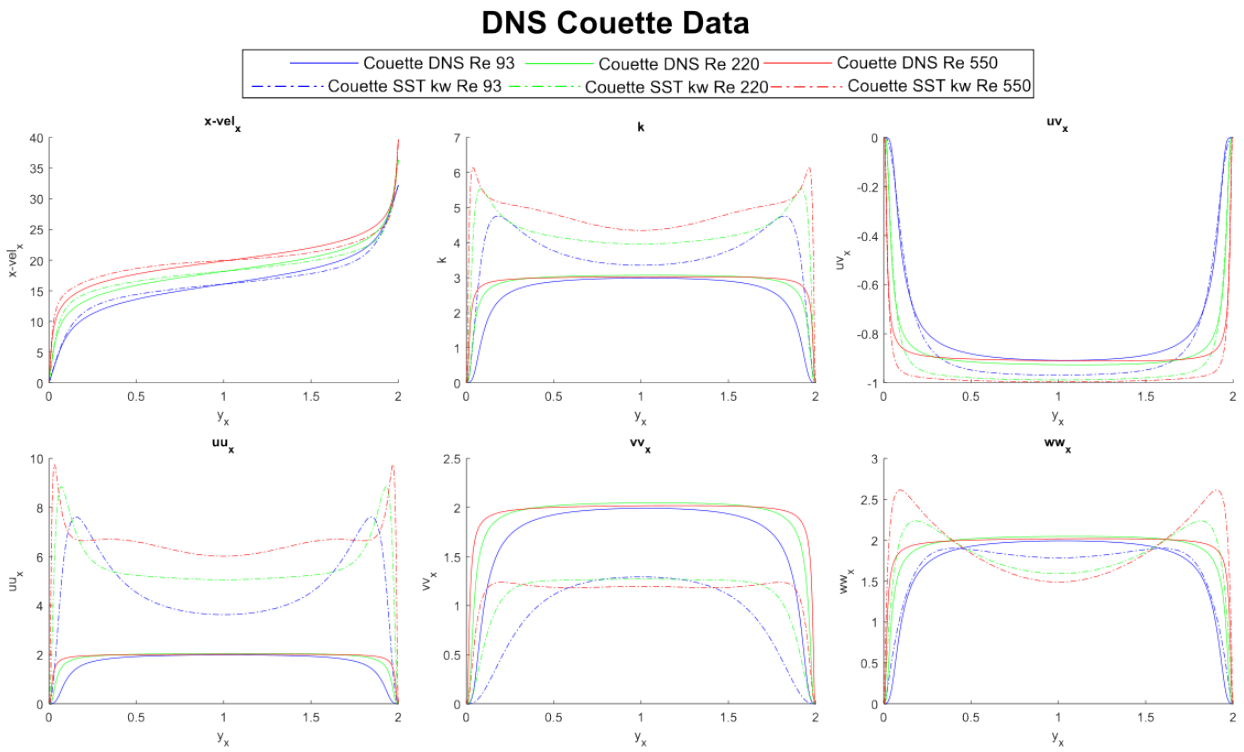

Figure 38: SST k-w and DNS Comparison for all Couette Flows (93, 220 and 550 Reynolds Number)

When looking at the computation of the RMSE computations for Couette flow in table 10 we can observe similar tendencies as the ones observed in the Channel Flow.

	Re 93	Re 219	Re 501
U	0.680	0.707	0.839
uu	3.384	3.865	4.693
uv	0.135	0.060	0.084
vv	0.829	0.806	0.807
ww	0.140	0.315	0.423
k	1.317	1.543	1.964

Table 10: RMSE for all Couette Flow Cases

When examining the errors, similar tendencies were observed as those in the Channel Flow results. Firstly, the $u'u'$ has also been the output which presented the most discrepancy from the DNS dataset, which was caused by the difference in the maximum values in the stresses in the X direction than the Y and Z, where similar errors were obtained. The velocity has also been accurately predicted, obtaining an average error of 0.7 for ranges between 0 and 40. Finally, it can be observed that in most of the variables, the cases that obtained the highest error compared to the DNS were the Reynolds 501, followed by Reynolds 220, indicating that the error increases as the Reynolds Numbers increase.

Comparison and Conclusion of the RANS Results

To conclude the RANS result section, figure 39 shows the error obtained for each variable for all cases of Channel and Couette Flows. In this figure, the tendencies explained in the last section can be seen, where the worst results were obtained for $u'u'$ and the best ones $u'v'$, and the same tendencies were obtained for the rest of the variables, confirming the consistency of the RANS predictions in both unidirectional flows test cases. When comparing both results, two main ideas can be extracted. The first one is that, overall in almost all of the cases, except the DNS 93 prediction for $w'w'$, RANS predicted more accurately, pressure driven flows rather than shear-driven flows, especially in the central section of the channel, where the most mismatches were produced for the Couette Flow. Secondly, opposite tendencies were seen in both flows regarding the accuracy and Reynolds numbers, where for the Channel flow, more accurate predictions were made for more turbulent flows, and for the Couette, the error increased with the Reynolds Number.

Overall, the RANS results were acceptable. However, they were highly affected by the assumptions governing RANS and the Boussinesq Hypothesis. In the following sections, an attempt will be made to overcome these discrepancies and obtain a model capable of improving these results.

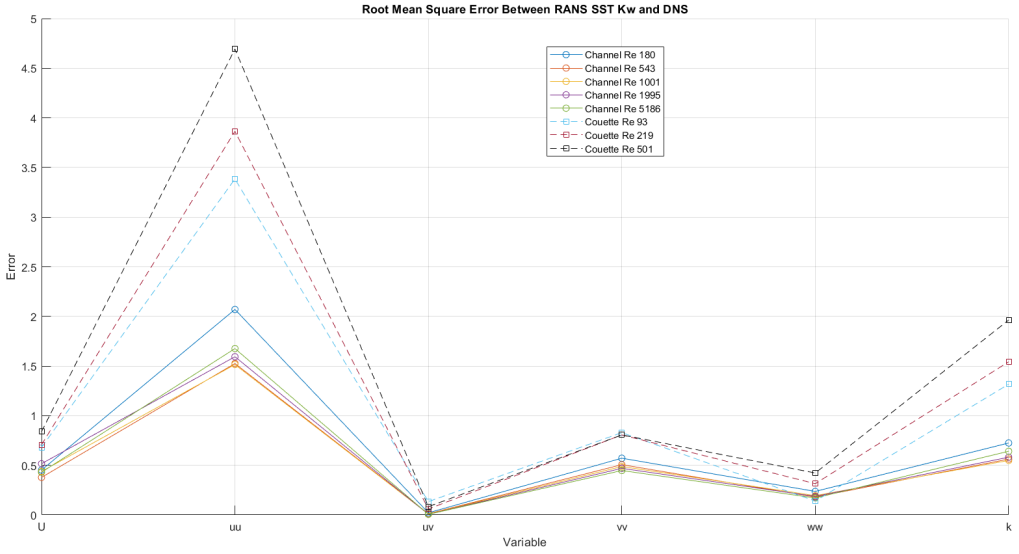

Figure 39: RMSE between SST k-w and DNS for all Channel and Couette FLows

4.1 PySINDy Results

4.1.1 Results for Approach 1: Equation Discovery for Covariances

The PySINDy model was trained on a selected dataset comprising DNS data for both Couette and channel flows, specifically at Reynolds numbers of 5200, 1000, and 180 for channel flow, and 93 and 220 for Couette flow. The model was subsequently tested on unseen data for channel flow at Re = 550 and for Couette flow at Re = 500 to assess the predictive capability of the derived equations.

The obtained SINDy equations, representing the covariance derivatives as functions of the flow parameters and their interactions, are seen in equations (58) through to (58):

$$(u'u')' = 1.176 \cdot k - 1.655 \cdot \tau_{\text{visc}} \cdot Re_\tau \cdot \nu + 134.711 \cdot \left(\frac{\partial u}{\partial y}\right)^2 \cdot \nu^2 + 18998.796 \cdot k^4 + 102817.778 \cdot \left(\frac{\partial u}{\partial y}\right)^2 \cdot \tau_{\text{visc}} \cdot \nu \cdot k + 746.893 \cdot \left(\frac{\partial u}{\partial y}\right)^2 \cdot \tau_{\text{visc}} \cdot k^2, \tag{58}$$

$$(v'v')' = 0.297 \cdot k - 81141.214 \cdot \left(\frac{\partial u}{\partial y}\right)^2 \cdot \tau_{\text{visc}} \cdot \nu \cdot k - 85.605 \cdot \left(\frac{\partial u}{\partial y}\right)^2 \cdot \tau_{\text{visc}} \cdot u_\tau \cdot k + 29.391 \cdot \tau_{\text{visc}} \cdot Re_\tau^2 \cdot \nu^2, \tag{59}$$

$$(w'w')' = 0.555 \cdot k - 78092.689 \cdot \left(\frac{\partial u}{\partial y}\right)^2 \cdot \tau_{\text{visc}} \cdot \nu \cdot k - 508.328 \cdot \left(\frac{\partial u}{\partial y}\right)^2 \cdot \nu \cdot k^2 - 205.272 \cdot \tau_{\text{visc}} \cdot Re_\tau \cdot k^3, \tag{60}$$

$$(u'v')' = -0.989 \cdot \left(\frac{\partial u}{\partial y}\right) \cdot \tau_{\text{visc}} - 351.066 \cdot \left(\frac{\partial u}{\partial y}\right) \cdot \tau_{\text{visc}} \cdot k^2 - 22308.210 \cdot \left(\frac{\partial u}{\partial y}\right) \cdot \nu^2 \cdot k - 3145.319 \cdot \tau_{\text{visc}} \cdot k^3 + 27241.033 \cdot \left(\frac{\partial u}{\partial y}\right) \cdot \nu \cdot u_\tau \cdot k^2. \tag{61}$$

- **Channel Flow at Re = 550 :** The performance of the PySINDy model for Channel flow at Re = 550 is presented in Figure 40. The figure consists of four subplots, each comparing the actual DNS data with the model's predictions for the covariances $u'u'$, $v'v'$, $w'w'$, and $u'v'$.

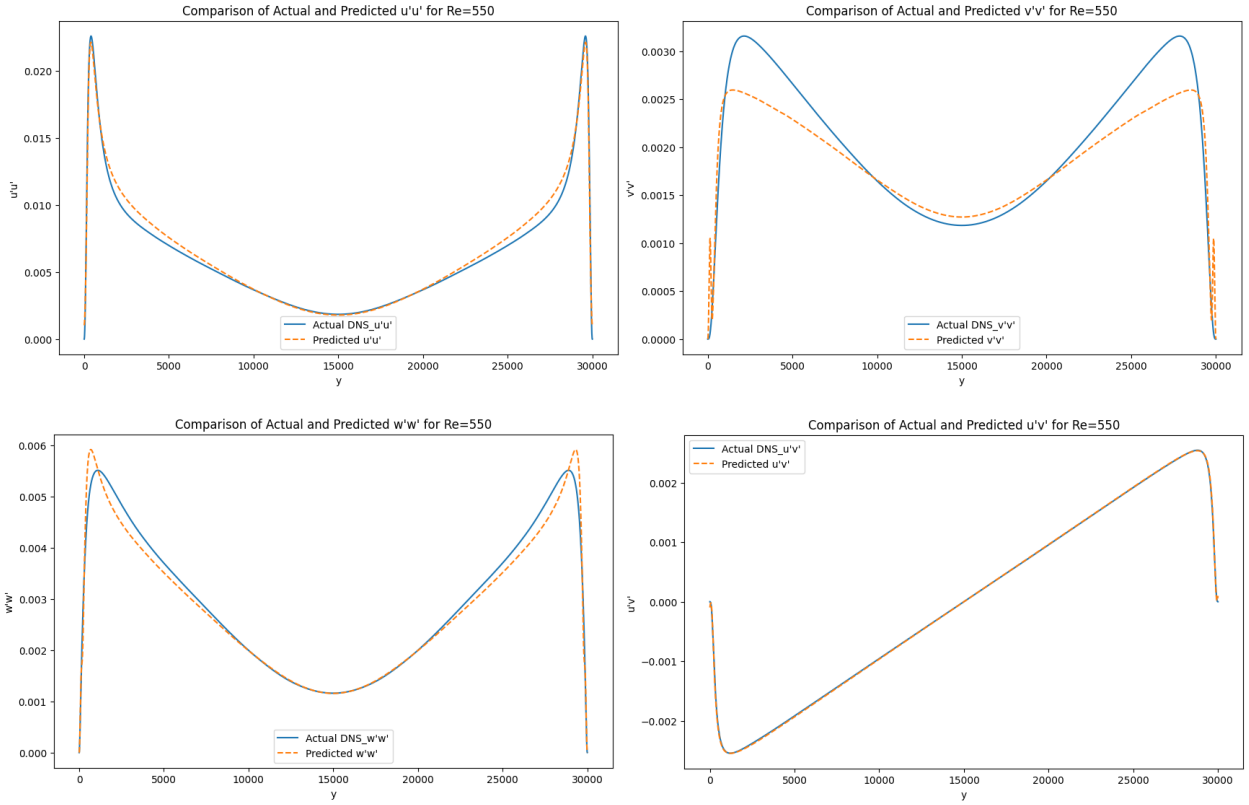

Figure 40: Comparison of Actual and Predicted Covariance Components for Channel flow at Re = 550.

The subplots reveal a strong correlation between the predicted and actual values across all terms. Notably, the $u'u'$ and $w'w'$ predictions align closely with the DNS results, indicating the model's effectiveness in capturing the energy distribution across different scales of motion. The $v'v'$ and $u'v'$ terms also show good agreement, albeit with slight deviations, which may be attributed to the model's extrapolation to unseen data.

- **Couette Flow at Re = 500 :** Similarly, the PySINDy model's predictions for Couette flow at Re = 500 are depicted in Figure 41. This figure includes four subplots for the covariances, mirroring the structure of the Channel flow analysis.

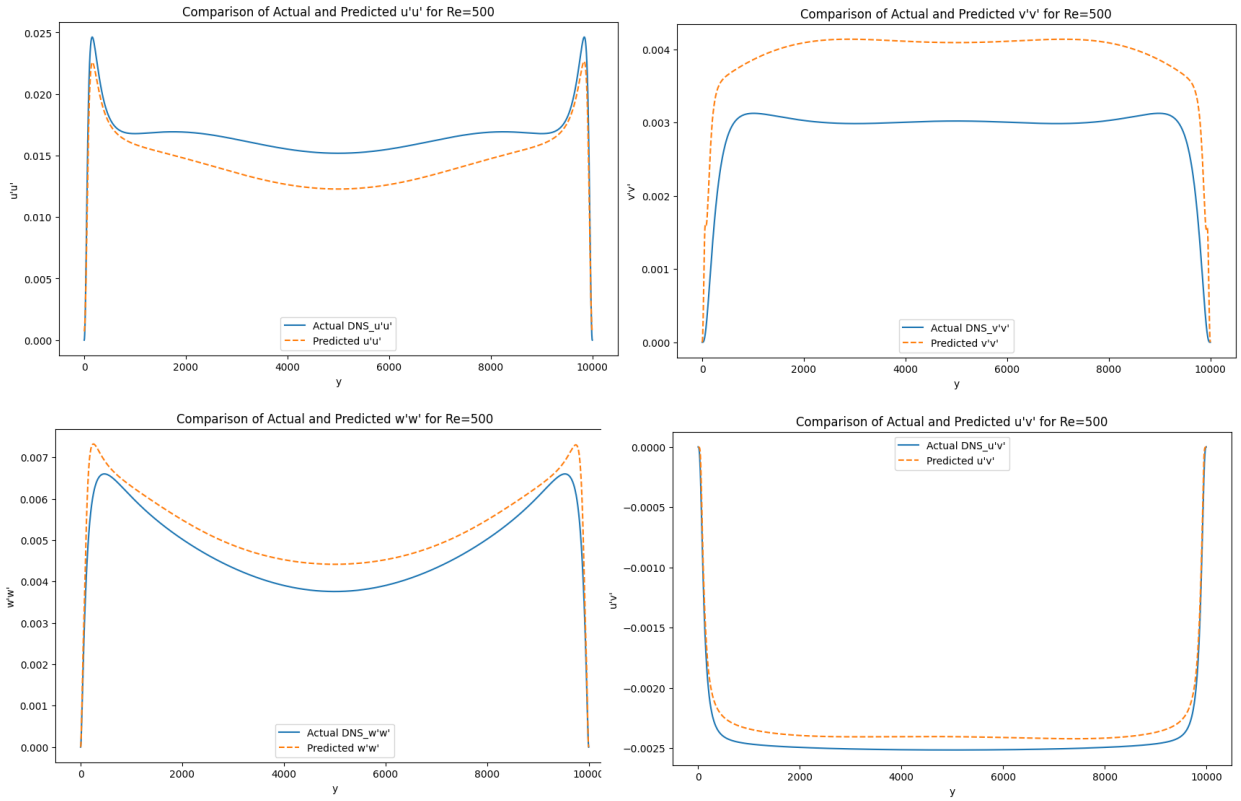

Figure 41: Comparison of Actual and Predicted Covariance Components for Couette flow at Re = 500.

The model's predictions for the Couette flow showcase its generalisation capability, with the covariances being accurately forecasted, as evidenced by the overlap of the predicted curves with the actual data points. The minor variances observed suggest the potential for further model refinement but do not detract from the overall accuracy of the predictions.

- **Comparative Discussion :** When comparatively analysing the results for Channel and Couette some important insights show up concerning the model's performance. The PySINDy model is highly accurate for Channel flow across all covariances but shows slight discrepancies in Couette flow predictions although still reasonably accurate compared to Channel flow. This difference might

be because the model was trained with a lot more data for Channel flow compared to Couette flow. Moreover, The differences in boundary conditions between Couette and Channel flows probably affect how well the model works. These findings underscore the need for a diverse training dataset that encompasses a wide spectrum of flow conditions and underscores the importance of considering boundary conditions in turbulence modelling. Such considerations are essential for refining the SINDy model and improving its generalisability across different fluid dynamics scenarios.

4.1.2 Results for Approach 2: Equation Discovery for K

4.1.2.1 Results for the Turbulent Kinetic Energy K

The governing equation linking velocities and velocity gradients with k is equation (62):

$$\begin{aligned} k =& 0.170\mathrm{Re}_\tau\nu + 1416.174\frac{dU}{dy}\nu^2 + 19470943.105W\nu^2 \\ & - 893.130\left(\frac{dU}{dy}\right)^2\nu^2 + 1.594\mathrm{Re}_\tau^2\nu^2 \\ & - 0.153U^4u_\tau + 10.366U^2u_\tau^3 - 10.347\frac{dU}{dy}^2\nu u_\tau^2 \end{aligned} \tag{62}$$

This equation shows the equation linking k with velocity gradients and initial parameters found using PySINDy on DNS data. It has been observed that U is a part of the equation. As previously discussed, this means that k will have different results depending on the reference point we consider. For example, k will be different for the same couette flow whether it has been considered the bottom plate static, and U between 0 and 2, or whether consider the bottom and top plates moving at the same speed in opposite directions, U therefore being between -1 and 1.

Figure 42 compares the predictions from the PySINDy equation using RANS inputs with the DNS data and the RANS data for the turbulent kinetic energy k for the channel testing datasets. We observe the PySINDy prediction is closer to DNS than the RANS data is. However, we observe inconsistencies, and behaviour that is not always symmetric, for example for the test with Re_tau = 180, the predicted k has different behaviours at the bottom and top plate (near y = 0 and y = 2).

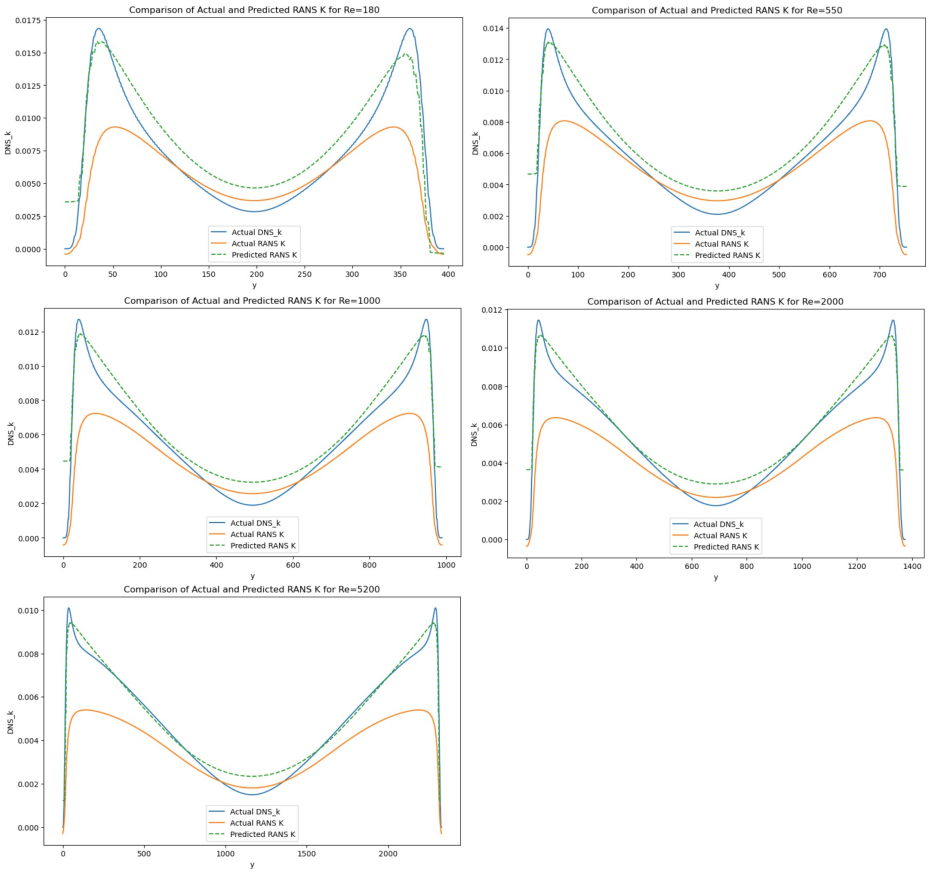

Figure 42: Results for Channel K

Figure 43 compares the predicted, DNS, and RANS obtained k for the Couette testing datasets. The results look symmetric, although a little noisier near the top wall. It has been observed that the predicted k curve is similar to the DNS' near the walls, but it loses the right shape as it does not match the DNS behaviours in the center. There is supposed to be a drop in k, reaching a local minimum at the middle of the channel and therefore creating two maximums around it. However, this behaviour does not happen with the predicted k, where the curve plateaus at its maximum and drops only near the plates instead of also dropping in the middle. The shape of our predicted curve resembles the RANS shape, but the magnitude is closer to the DNS values.

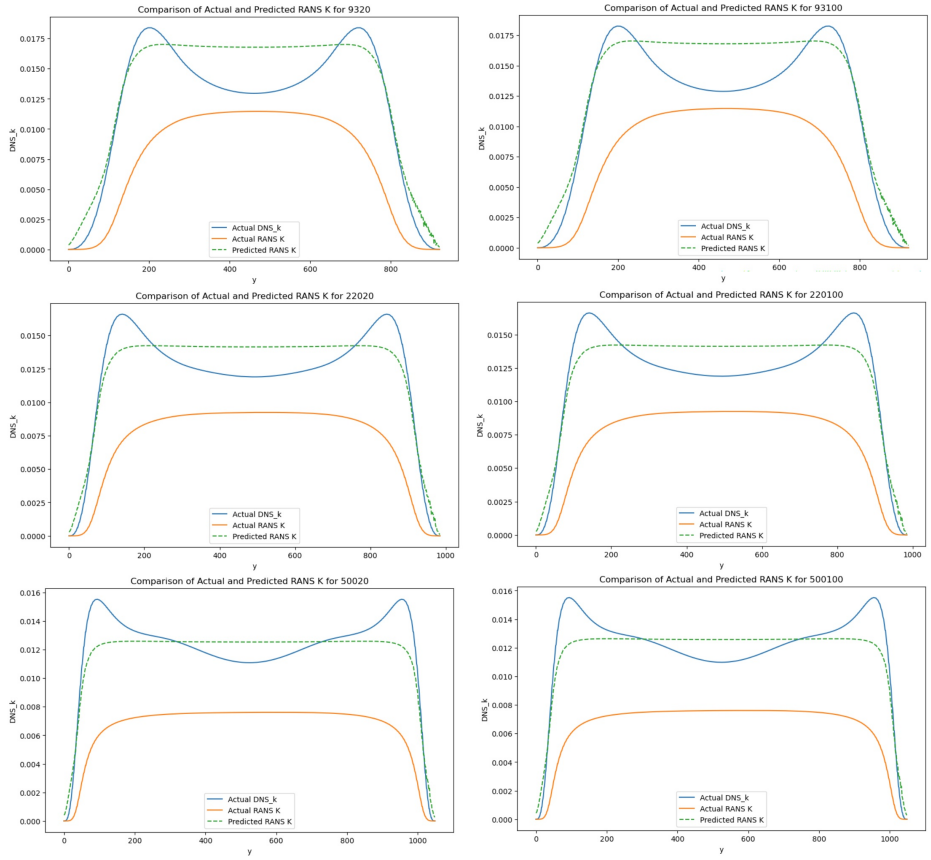

Figure 43: Results for Couette K

the analysis conducted, it was found that the PySINDy equation produced better results than the RANS method, particularly for Channel flow. However, the lack of symmetry for certain test cases and the fact that the equation uses U as a feature is problematic, and gives us only a model that is flawed physically. However, this still achieves good general approximations and is usable for at least Channel and Couette flows.

4.1.2.2 Results for the Covariances from the New Boussinesq Hypothesis

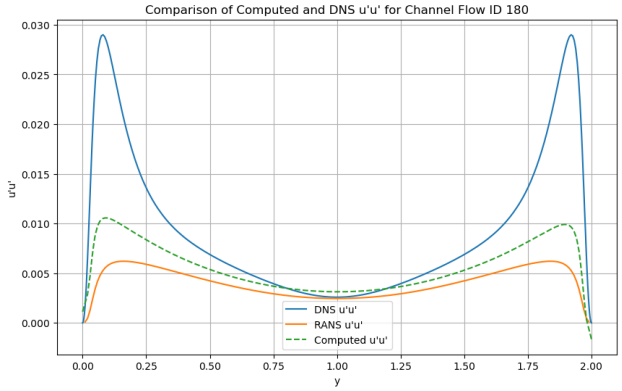

Figure 44: Results for u'u' from the Boussinesq Hypothesis with K Replaced with our PySINDy Equation

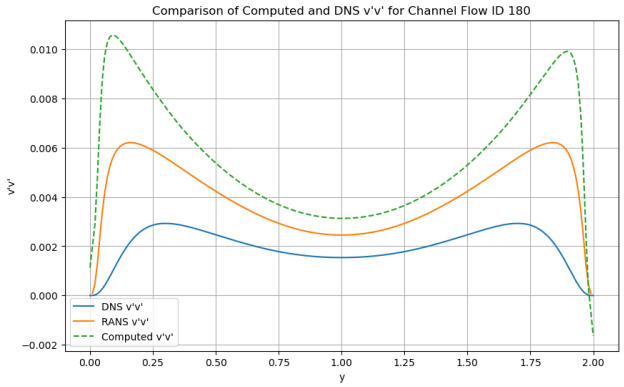

Figure 45: Results for v'v' from the Boussinesq Hypothesis with K Replaced with our PySINDy Equation

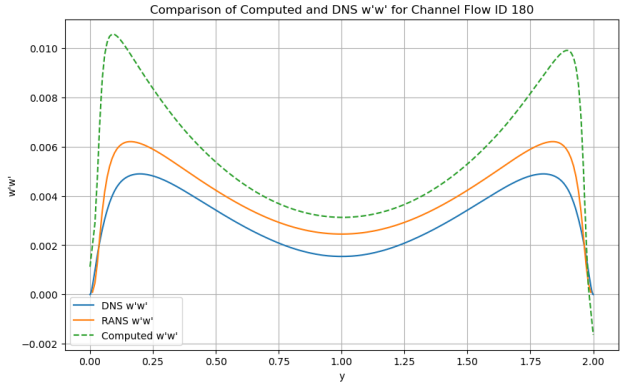

Figure 46: Results for w'w' from the Boussinesq Hypothesis with K Replaced with our PySINDy Equation

Figures 44, 45 and 46 show the results for u'u', v'v' and w'w' calculated through the Boussinesq hypothesis eddy viscosity equation with the PySINDy equation replacing k, and using RANS values. it has been observed that results differ from the DNS k. More precisely, the same phenomenon has been observed which is happening with the RANS k results: For a given Reynolds Number, u'u', v'v', and w'w' are equal to each other. Only u'v' is different as shown in Figure 47 because k is not involved in its computation.

It has been observed that the existing RANS results suffer from the same limitations. The velocity gradients are negligible in comparison to k, resulting in equivalent results for u'u', v'v', and w'w' for a given test configuration. This represents a limitation of the eddy viscosity model.

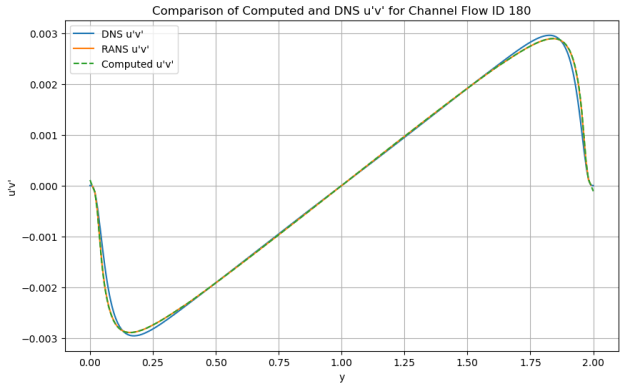

Figure 47: Results for u'v' from the Boussinesq Hypothesis with K Replaced with our PySINDy Equation

4.1.2.3 Approach 2 Results Analysis

A great fit for the variable k was not obtained using PySINDy. Nevertheless, the equation found is still correct and provides better results than RANS. Despite an improved k, the issue with this method arises from the limitations of the eddy viscosity model itself. The model is unable to work well when the covariances are insignificant compared to k.

4.1.3 Results for the Final Approach: Equation Discovery for the x and y Momentum

The following section showcases the most intriguing and well-matched results of the final approach. The objective is to ensure that these results are easily replicable.

4.1.3.1 X Momentum Equation

The governing equations have been found by using PySINDy that give good results for the x momentum. Equations (63) and (64) show the lower order and higher order solutions respectively:

$$0.008\frac{dP}{dx} + 1.007\frac{duv}{dy} - \frac{d^2U}{dy^2} = 0 \tag{63}$$

$$1.053\frac{duv}{dy} - 267.289\left(\frac{dP}{dx}\right)^2\frac{duv}{dy} - 1898.639\left(\frac{dP}{dx}\right)^3\frac{duv}{dy} + 161291.025\left(\frac{dP}{dx}\right)^3\left(\frac{duv}{dy}\right)^2 - \frac{d^2U}{dy^2} = 0 \tag{64}$$

Following testing and validation plots will show the results of the second equation.

Table 11: X Momentum PySINDy Equation Results

Equation	R2 Train	RMSE Train	R2 Test	RMSE Test
$\frac{d^2U}{dy^2} = 0.008\frac{dP}{dx} + 1.007\frac{duv}{dy}$	0.951356904	0.001599296	0.99655843	0.000661763
$\frac{d^2U}{dy^2} = 1.053\frac{duv}{dy} - 267.289\frac{dP}{dx}^2\frac{duv}{dy} - 1898.639\frac{dP}{dx}^3\frac{duv}{dy} + 161291.025\frac{dP}{dx}^3\frac{duv}{dy}^2$	0.983182652	0.000940366	0.995996848	0.000713715
$\frac{d^2U}{dy^2} = 1.057\frac{duv}{dy} - 272.974\frac{dP}{dx}^2\frac{duv}{dy} - 1779.846\frac{dP}{dx}^3\frac{duv}{dy} + 178958.541\frac{dP}{dx}^3\frac{duv}{dy}^2$	0.981260571	0.00099265	0.995668152	0.000742439
$\frac{d^2U}{dy^2} = 1.028\frac{duv}{dy} - 276.623\frac{dP}{dx}^2\frac{duv}{dy} - 1982.288\frac{dP}{dx}^3\frac{duv}{dy} - 8057.727\frac{dP}{dx}^2\frac{duv}{dy}^2$	0.981739621	0.00097988	0.995440652	0.000761685
$\frac{d^2U}{dy^2} = 1.028\frac{duv}{dy} - 103.557\frac{dP}{dx}^2\frac{duv}{dy} + 146597.095\frac{dP}{dx}^3\frac{duv}{dy}^2$	0.975593849	0.001132838	0.995413536	0.000763947
$\frac{d^2U}{dy^2} = -0.001 + 1.000\frac{duv}{dy}$	0.9578	0.0015	0.9950	0.0008
$\frac{d^2U}{dy^2} = 1.006\frac{duv}{dy} - 106.158\frac{dP}{dx}^2\frac{duv}{dy} - 7038.904\frac{dP}{dx}^2\frac{duv}{dy}^2$	0.9734	0.0012	0.9949	0.0008
$\frac{d^2U}{dy^2} = -0.001 + 1.048\frac{duv}{dy} - 244.934\frac{dP}{dx}^2\frac{duv}{dy} - 1806.924\frac{dP}{dx}^3\frac{duv}{dy} + 141119.326\frac{dP}{dx}^3\frac{duv}{dy}^2$	0.9851	0.0009	0.9946	0.0008
$\frac{d^2U}{dy^2} = -0.001 + 1.025\frac{duv}{dy} - 251.150\frac{dP}{dx}^2\frac{duv}{dy} - 1870.954\frac{dP}{dx}^3\frac{duv}{dy} - 6960.211\frac{dP}{dx}^2\frac{duv}{dy}^2$	0.9839	0.0009	0.9938	0.0009
$\frac{d^2U}{dy^2} = 0.997\frac{duv}{dy} - 99.571\frac{dP}{dx}^2\frac{duv}{dy} + 154.067\frac{dP}{dx}\frac{duv}{dy}^2$	0.9672	0.0013	0.9934	0.0009
$\frac{d^2U}{dy^2} = 1.000\frac{duv}{dy}$	0.9406	0.0018	0.9927	0.0010
$\frac{d^2U}{dy^2} = 1.055\frac{duv}{dy} - 100.176\frac{dP}{dx}^2\frac{duv}{dy}$	0.9618	0.0014	0.9920	0.0010
$\frac{d^2U}{dy^2} = -0.001 + 0.999\frac{duv}{dy} - 60.478\frac{dP}{dx}^2\frac{duv}{dy} + 90.388\frac{dP}{dx}\frac{duv}{dy}^2$	0.9749	0.0011	0.9919	0.0010
$\frac{d^2U}{dy^2} = -0.001 + 1.087\frac{duv}{dy} - 271.194\frac{dP}{dx}^2\frac{duv}{dy} - 1979.052\frac{dP}{dx}^3\frac{duv}{dy} - 8912.016\frac{dP}{dx}^2\frac{duv}{dy}^2 + 2855.738\frac{dP}{dx}\frac{duv}{dy}^3$	0.9854	0.0009	0.9916	0.0010
$\frac{d^2U}{dy^2} = -0.001 + 1.086\frac{duv}{dy} + 5.729\frac{dP}{dx}\frac{duv}{dy}$	0.9666	0.0013	0.9913	0.0011
$\frac{d^2U}{dy^2} = -0.001 + 1.062\frac{duv}{dy} + 4.432\frac{dP}{dx}\frac{duv}{dy}$	0.9735	0.0012	0.9891	0.0012
$\frac{d^2U}{dy^2} = -0.001 + 1.057\frac{duv}{dy} + 4.409\frac{dP}{dx}\frac{duv}{dy} - 0.183\frac{duv}{dy}^2$	0.9736	0.0012	0.9890	0.0012
$\frac{d^2U}{dy^2} = -0.001 + 1.066\frac{duv}{dy} + 4.367\frac{dP}{dx}\frac{duv}{dy}$	0.9734	0.0012	0.9888	0.0012
$\frac{d^2U}{dy^2} = -0.002 + 1.000\frac{duv}{dy}$	0.9687	0.0013	0.9884	0.0012
$\frac{d^2U}{dy^2} = 1.115\frac{duv}{dy} + 7.653\frac{dP}{dx}\frac{duv}{dy}$	0.9585	0.0015	0.9865	0.0013
$\frac{d^2U}{dy^2} = -0.002 + -0.016\frac{dP}{dx} + 1.059\frac{duv}{dy} - 0.051\frac{dP}{dx}^2 + 4.632\frac{dP}{dx}\frac{duv}{dy} + 0.062\frac{duv}{dy}^2$	0.9742	0.0012	0.9840	0.0014
$\frac{d^2U}{dy^2} = -0.003 + 1.009\frac{duv}{dy} - 61.701\frac{dP}{dx}^2\frac{duv}{dy} + 48.916\frac{dP}{dx}\frac{duv}{dy}^2$	0.9804	0.0010	0.9787	0.0016
$\frac{d^2U}{dy^2} = -0.003 + 1.030\frac{duv}{dy} + 1.487\frac{dP}{dx}\frac{duv}{dy} - 47.977\frac{dP}{dx}^2\frac{duv}{dy} + 25.009\frac{dP}{dx}\frac{duv}{dy}^2$	0.9805	0.0010	0.9786	0.0017
$\frac{d^2U}{dy^2} = -0.003 + 1.028\frac{duv}{dy} - 51.411\frac{dP}{dx}^2\frac{duv}{dy} + 4.732\frac{dP}{dx}^3\frac{duv}{dy} + 0.191\frac{dP}{dx}^2\frac{duv}{dy}^2 - 0.055\frac{dP}{dx}\frac{duv}{dy}^3$	0.9793	0.0010	0.9763	0.0017
$\frac{d^2U}{dy^2} = -0.003 + 1.000\frac{duv}{dy}$	0.9753	0.0011	0.9715	0.0019

For the x momentum equation, Table 11 shows different uncovered equations sorted by descending r squared value for the testing dataset.

Analysing equations (63) and (64), it is evident that they have different complexities. The first equation has an order of 1 whereas the second equation has an order of 5, but both approaches give similar r^2 results for the testing dataset. This demonstrates the adaptability of our method to find multiple kinds of governing equations. It is also observed that for the training dataset, the second and more complex equation fits better, as it presents a higher r-squared result.

To have a more reliable interpretation of the fitting of the equations, more data is required. Data augmentation and data extrapolation have been a key concern, and approaches such as such as channel rotation have been developed to increase data diversity and generate more robust equations and results.

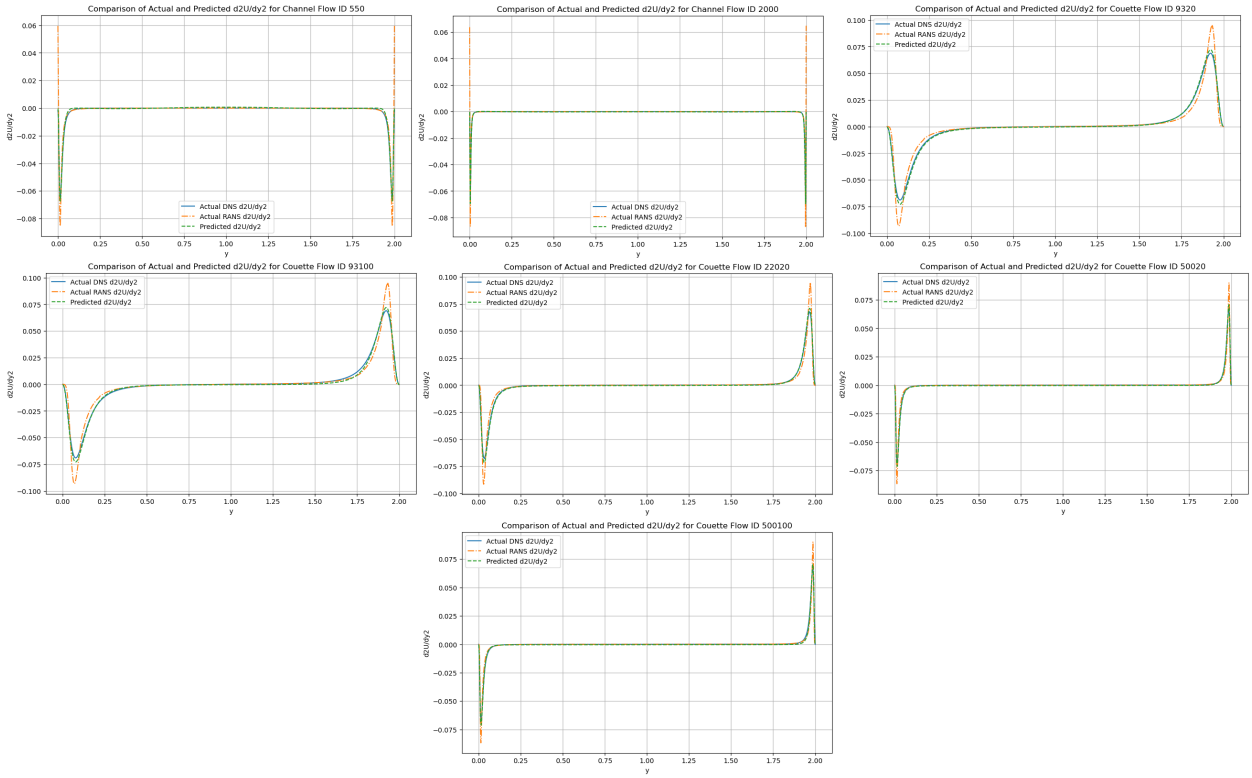

Figure 48: Results for d^2U/dy^2

Figure 48 shows $\frac{d^2U}{dy^2}$ as a calculation of $1.053\frac{duv}{dy} - 267.289\left(\frac{dP}{dx}\right)^2\frac{duv}{dy} - 1898.639\left(\frac{dP}{dx}\right)^3\frac{duv}{dy} + 161291.025\left(\frac{dP}{dx}\right)^3\left(\frac{duv}{dy}\right)^2$ versus y and plots it against the DNS and RANS data. It is observed that the PySINDy and blue curves behaviours match closely. However, there is some slight misalignment in the steep curves especially for lower Reynolds Numbers.

Figure 48 helps to visualise those slight differences by plotting the results for each testing dataset setup, allowing us to observe how our equations react to differences between flow types (Channel and Couette) as well as to differences between Reynolds Numbers, where it looks like our equation fits higher Reynolds Numbers better than lower ones. This is better than the opposite, as we are trying to find an equation for turbulent flows and higher Reynolds Numbers mean higher turbulence.

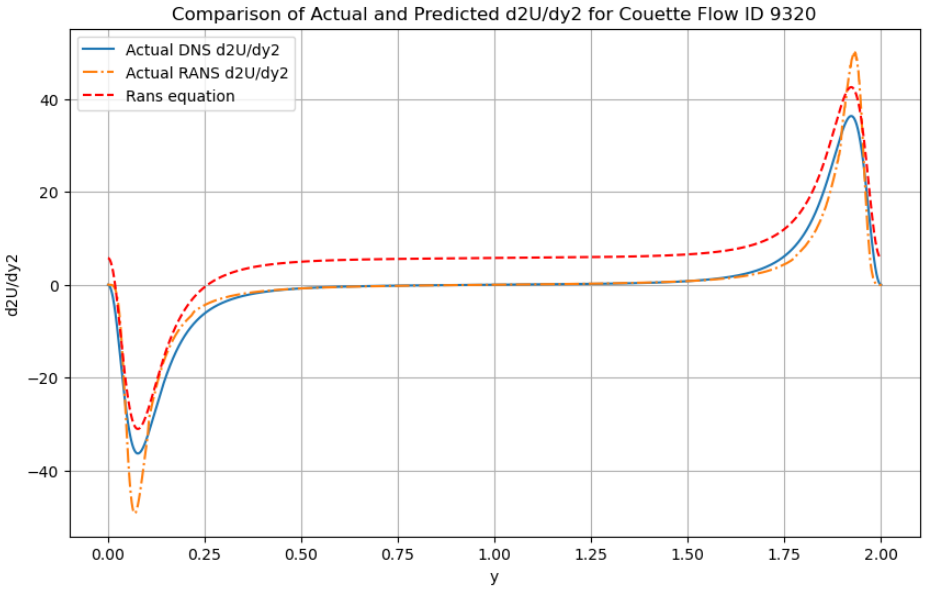

Figure 50: Results from the RANS Derived Equation for Unnormalised Data

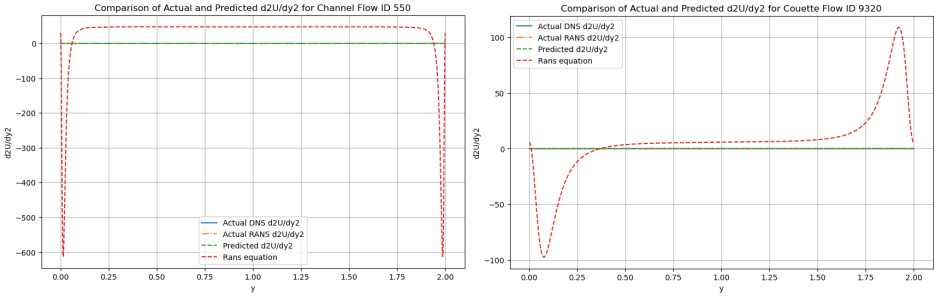

Figure 49: Results from the RANS Derived Equation for Normalised Data

Figure 49 shows in the red curve $\frac{d^2U}{dy^2}$ as a calculation of the mathematically derived RANS x momentum equation versus y and compares it with DNS and PySINDy data. The results obtained were not as expected, since the RANS equation works for unnormalised data, but is therefore not adequate to be used directly on normalized data. Moreover, this result comes from the equation using $\frac{dP}{dx} = u_t a u^2$ as it is intended to be calculated. This should cause problems for the PINN model, as there had been a misunderstanding on the calculation of $\frac{dP}{dx}$ between the members of the group (discussion), and the wrong calculation of $\frac{dP}{dx} = \frac{P}{Lx}$ in the DNS interpolated data had been included. This also explains the first equation shown in table 11. This also explains why the coefficient before $\frac{dP}{dx}$ in the first equation shown in the table labelled "X momentum PySINDy equation results" was not what was expected.

The equation $\frac{dP}{dx} = u_t a u^2$ is used to calculate the shape of the curve, but it is apparent that the curve is not symmetrical when observed. This indicates that the equation is not suitable for the PINN model and is also slightly incorrect in general. However, our PySINDy equation fits better. The comparison of unnormalised data with the correct $\frac{dP}{dx} = u_t a u^2$ against the unnormalised RANS and DNS data also confirms the unsuitability of the RANS equation, as depicted in Figure 50. Since the $\frac{dP}{dx} = \frac{P}{Lx}$ equation is used for the PINN model, the PySINDy equation should be used as a governing equation instead of the RANS derived x momentum equation.

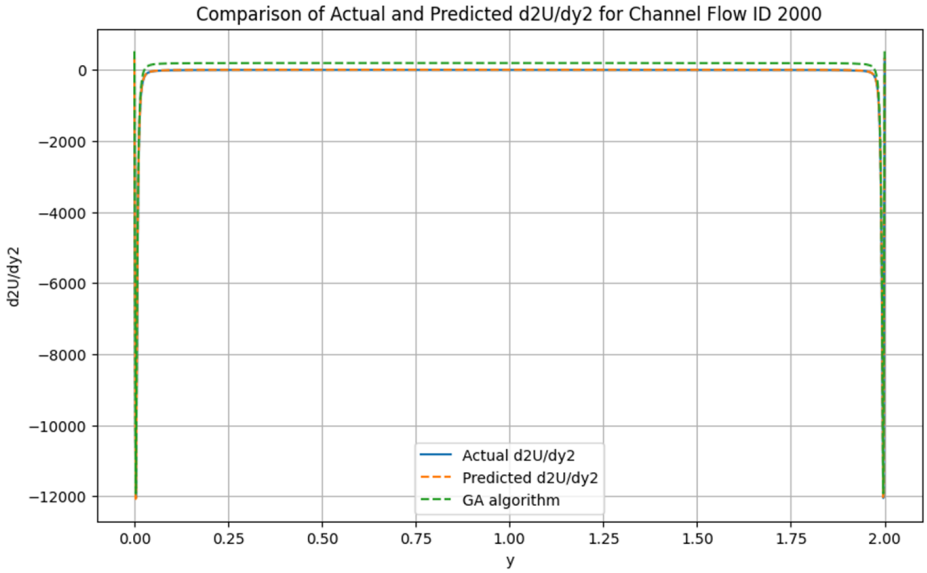

Figure 51: Results from the Genetic Algorithm on a PySINDy Derived Equation

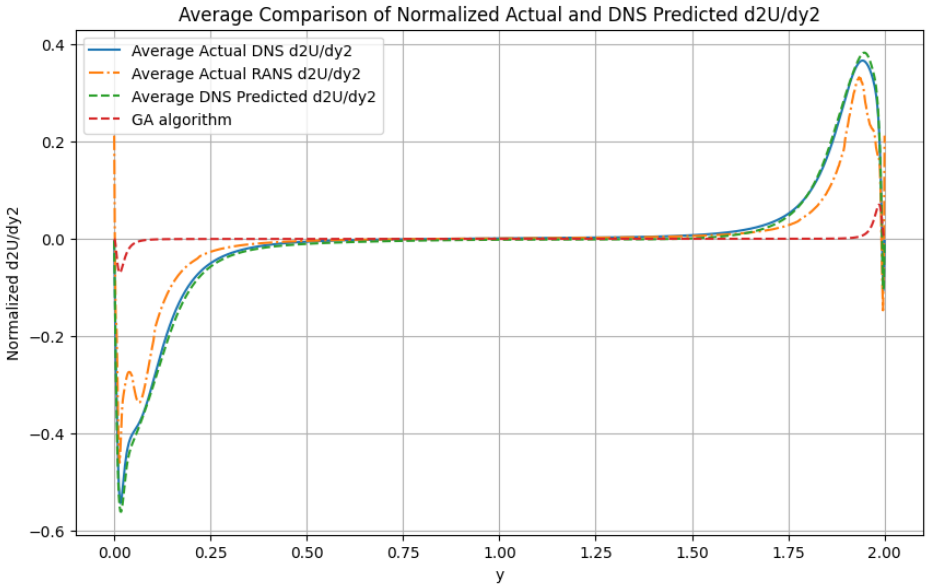

Figure 52: Results for Normalised and Averaged d2U$_{dy}$2*withtheGeneticAlgorithm*

Figure 51 shows in green a curve resulting from a PySINDy equation that went through a coefficient improvement process by using the Genetic Algorithm. The behaviour and general shape of the curve match the DNS data very well. However, there is a slight vertical offset that hasn't been solved.

Figure 52 gives us a more general view. it has been observed that the equation provides a satisfactory fit for the interaction at the bottom wall, while the fit for the interaction at the top wall is slightly weaker. It also looks like our equation does not capture the neutrality of the middle part perfectly: the curve sits slightly under 0. Looking back at figure 48 for the Channel flow with Re_tau = 550 plots, the PySINDy

curve shows oscillatory behaviour in the middle part as opposed to the DNS and RANS which shows a linear behaviour of the type $\frac{d^2U}{dy^2} = 0$.

Looking at the Couette flow results still in figure 48, it has been also observed that the impact of PI = 20 and PI = 100, where on both arcs at the extremities of the linear middle part and for a same setup, the PySINDy curve will be slightly under the DNS curve at the left arc and perfectly matching at the right arc for PI = 20. However, it is the opposite for PI = 100, the PySINDy curve will be well fitting at the left arc but under the DNS curve at the right arc. When considering the overall results, a combination of well-fitted and slightly below the DNS curve can be observed at both arcs near the center for each configuration that includes both 20PI and 100PI outcomes. This results, as shown in figure **??**, to a normalised and averaged PySINDy curve that has a slight vertical offset at the middle part.

4.1.3.2 Y Momentum Equation

The governing equation that has been found by using PySINDy that gives good results for the y momentum is equation (65):

$$-\frac{dv'v'}{dy} - \frac{dP}{dy} = 0 \tag{65}$$

The following testing and validation plots will show the results from this equation.

Table 12: Y Momentum PySINDy Equation Results

Equation	**R2 Train**	**RMSE Train**	**R2 Test**	**RMSE Test**
$\frac{d(v'v')}{dy} = -1.000\frac{dP}{dy}$	1.0000	2.202e-06	1.0000	7.185e-07
$\frac{d(v'v')}{dy} =$ $-2451.492\frac{dP}{dy}^3 + 0.294\frac{dP}{dy}^3\mathrm{Re}_\tau + 1257020.483\frac{dP}{dy}^5$	0.9224	0.001616	0.9386	0.002454
$\frac{d(v'v')}{dy} =$ $-2665.127\frac{dP}{dy}^3 + 0.131\frac{dP}{dy}^3\mathrm{Re}_\tau + 1442007.391\frac{dP}{dy}^5$	0.9113	0.001727	0.9293	0.002633
$\frac{d(v'v')}{dy} = -972.561\frac{dP}{dy}^3 + 0.339\frac{dP}{dy}^3\mathrm{Re}_\tau$	0.8428	0.002299	0.8561	0.003757
$\frac{d(v'v')}{dy} = -0.001\frac{dP}{dy}\mathrm{Re}_\tau + -670.648\frac{dP}{dy}^3$	0.8406	0.002315	0.8541	0.003782
$\frac{d(v'v')}{dy} = -0.002\frac{dP}{dy}\mathrm{Re}_\tau$	0.6836	0.003262	0.4345	0.007446
$\frac{d(v'v')}{dy} = -0.002\frac{dP}{dy}\mathrm{Re}_\tau + 0.009\frac{dP}{dy}^3\mathrm{Re}_\tau$	0.6837	0.003262	0.4344	0.007447
$\frac{d(v'v')}{dy} = -1.517\frac{dP}{dy}^3\mathrm{Re}_\tau$	0.5465	0.003905	0.3721	0.007846
$\frac{d(v'v')}{dy} = 0.083\frac{dP}{dy}^3\mathrm{Re}_\tau$	0.2952	0.004869	0.1756	0.008990
$\frac{d(v'v')}{dy} = -0.289\frac{dP}{dy}^3\mathrm{Re}_\tau$	0.2219	0.005116	0.1376	0.009195

For the y momentum equation, Table 12 shows different uncovered equations sorted by descending r squared value for the testing dataset. Upon analysing the first equation, it becomes apparent that it is entirely equivalent to the RANS equation that was mathematically derived from the problem configuration. Based on the $\mathbf{r^2}$ results, it can be concluded that the equation matches the DNS data extremely well.

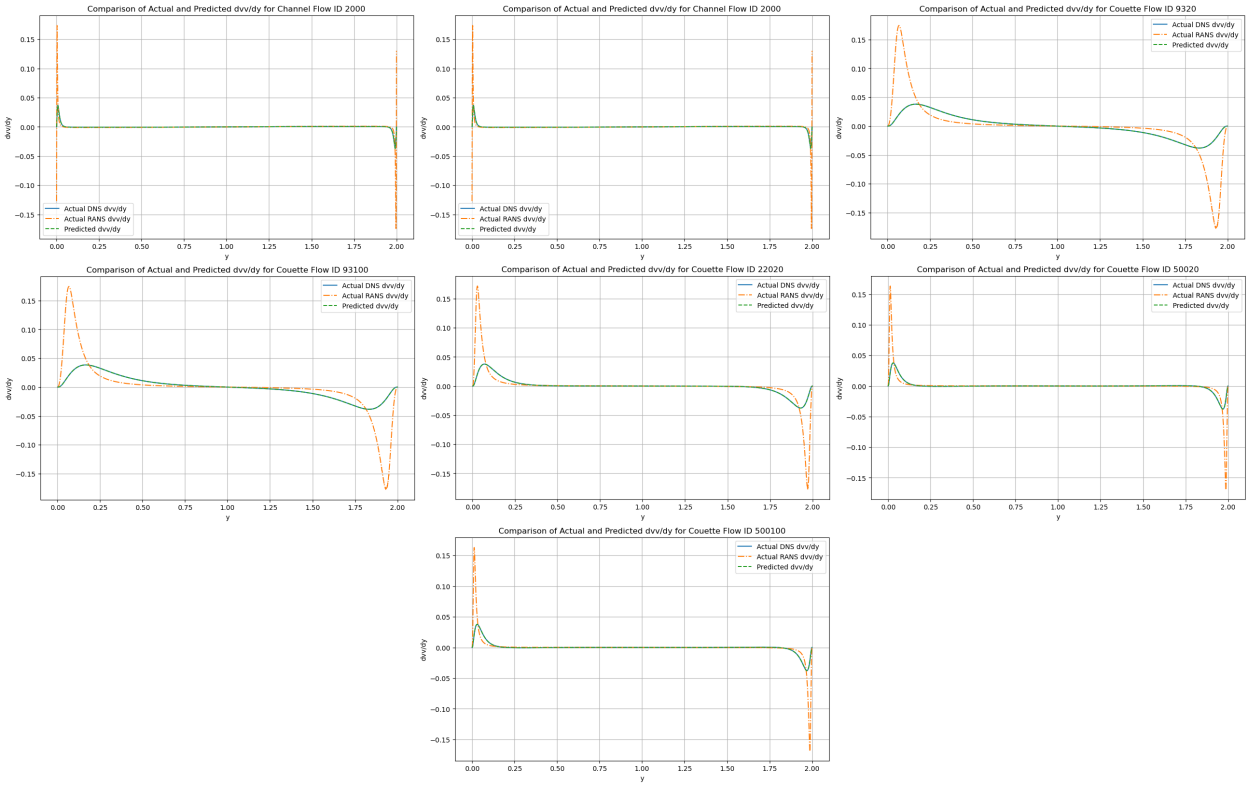

Figure 53: Results for $\frac{dv'v'}{dy}$

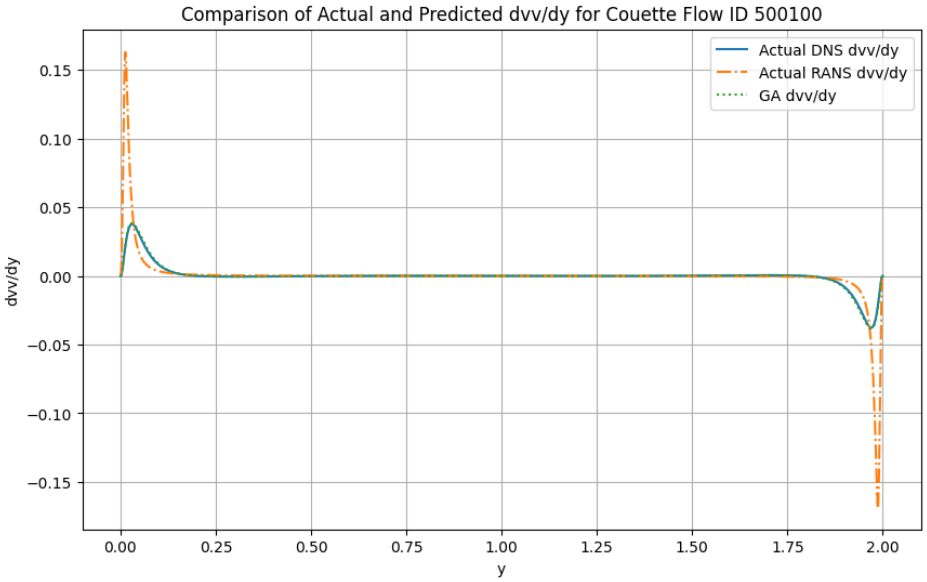

Figure 54: Results for $\frac{dv'v'}{dy}$ with the Genetic Algorithm

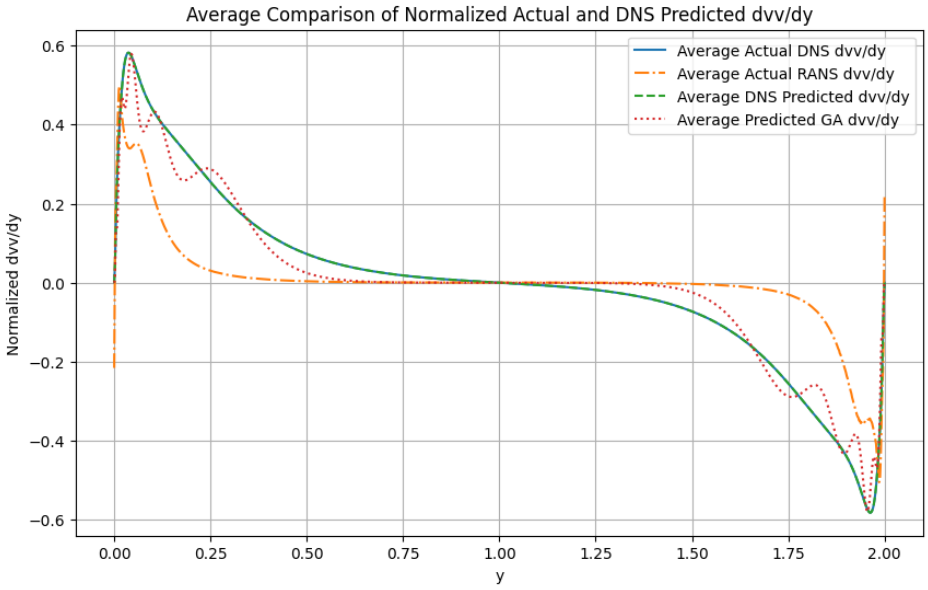

Figure 55: Results for Normalised and Averaged dvv$_d$y

Figure 53 helps in the visual validation of the newfound equation by plotting $\frac{dv'v'}{dy}$ calculated from the other terms of the PySINDy y momentum equation versus y as the green curve, and comparing it against DNS data and CED produced RANS data as the blue and orange curves respectively for each different setup in the testing dataset.

Figure 53 confirms the high r squared score as it has been observed that the green and blue curves are superimposed for each testing dataset results. Figure 54 shows the best fit among different Reynolds numbers. Despite employing channel data as the initial parent, the model fits the data well, with the exception of the near-wall area which shows less accuracy.

Figure 55 also helps in having a more general view of the results by normalising and averaging all the testing dataset results and plotting them in a single graph. There again, the PySINDy and DNS curves are superimposed, confirming the reliability of our equation and its generalisability for at least Channel and Couette flow configurations. After processing validation using a genetic algorithm, the resulting graph displays increased complexity and more pronounced curves.

4.1.3.3 Final Approach Results Analysis

Two equations have been identified to replace the RANS equations in the PINN model. The DNS data demonstrate that both of these equations offer good correlation. The y momentum equation is a perfect match. The x momentum equation also shows good correlation, however, there are some slight differences for lower Reynolds numbers, and symmetrical inconsistencies for Couette flow.

4.2 Machine Learning Results

4.2.1 Simple Neural Network

4.2.1.1 Predicting the Reynolds Stress Tensors

This part of the results for the machine learning aspect of the project deals with the RST outputs for the simple neural network. The test file is the Reynolds Number 93 with 20 data-points, one of the chosen files at random from the neural network to be a part of the 20% test split. The average MSE comparison of the RSTs across all Reynolds values, ie: 93, 220 and 500 for Channel flow and Reynolds numbers 180, 550, 1000, 2000 and 5200 for Couette flow for all 6 stress tensors is further displayed for all tested epoch numbers (10, 50, 100). Training results are outlined in Appendix C1.

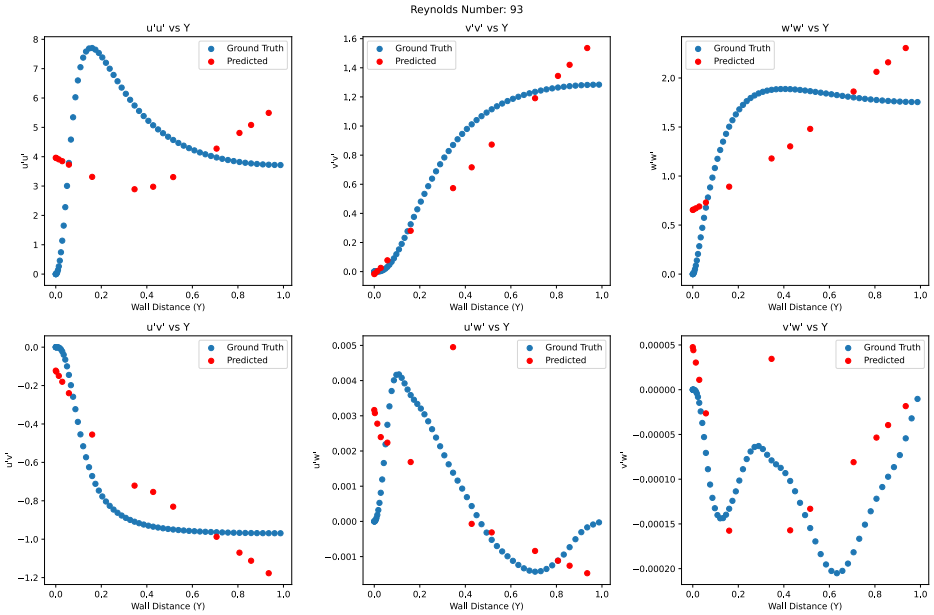

Figure 56: Simple Neural Network Training for Channel Flow - Re 180 - Epoch Number 10

As illustrated in Figure 56, the simple NN model demonstrates initial challenges in accurately predicting the RSTs during Epoch 10. The ground truth values throughout both the Channel and Couette flows are considerably underestimated by the model. For example, in the $u'u'$ tensor, the predictions of the simple NN model only reaches about 4, whereas the ground truth reveals values peaking at 7. This considerable deviation, particularly noticeable around the peak at wall distance $Y = 0.4$, highlights the model's early stage of underfitting where it has not yet adapted to the dataset's complexity. The model's inability to accurately capture the complicated dynamics of the DNS data suggests that the model was not fine-tuned enough during the initial training phase to fully comprehend the complexities of the flow dynamics.

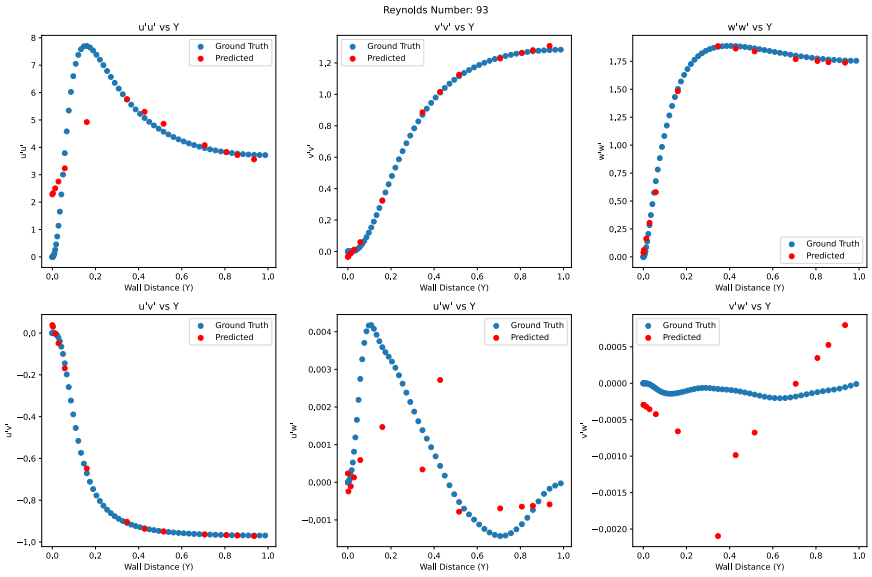

Figure 57: Simple Neural Network Training for Channel Flow - Re 180 - Epoch Number 50

As depicted in Figure 57, noticeable improvements are evident in the predictions made by the simple NN model at Epoch 50. Although still facing challenges, particularly in accurately predicting intermediately complex tensor interactions like $u'v'$, the model begins to approach the ground truth values more closely. For instance, the predicted values in the $v'v'$ tensor now reach up to approximately 0.8, showing enhanced adaptation to the dataset's dynamics compared to Epoch 10. This moderate stage of training exhibits the model's capacity to better learn from the flow data, albeit not fully optimized yet. It underscores a transitional phase in learning, balancing between underfitting and optimal fitting, as the model refines its parameters to more accurately capture the complexities of the flow dynamics.

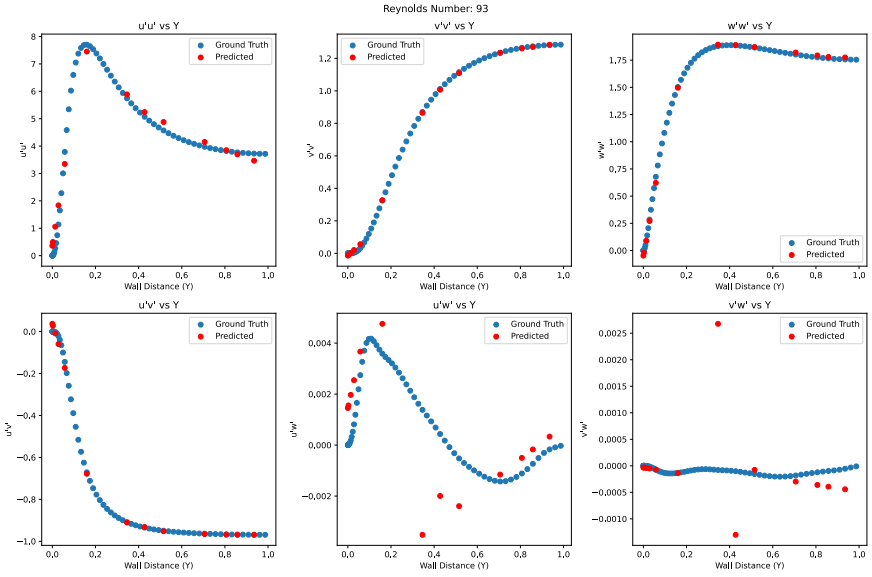

Figure 58: Simple Neural Network Training for Channel Flow - Re 180 - Epoch Number 100

Illustrated in Figure 58, by Epoch 100, the simple NN model demonstrates a significant advancement in its prediction accuracy. The model not only captures the correct trends across all tensors but also closely approximates the peak values, such as in the $w'w'$ tensor, where predictions almost mirror the ground truth peaking at about 1.75. Despite these improvements, slight overpredictions, particularly at the boundaries of the domain, suggest an onset of overfitting. This epoch indicates a high proficiency level, where the model has effectively learned the detailed nuances of the DNS data, although at the risk of incorporating some noise into the predictions. The advanced training phase shown here offers a glimpse into the model's capabilities to generalize well across diverse flow types, highlighting the delicate balance required to maintain accuracy without overfitting.

- **Comparative Discussion:** Comparing figures 56 through to 58, it is observed that the higher the epoch number, the better and more accurately that the simple neural network predicts the DNS data given the unseen test file. An Epoch number of 100, as seen in figure 58 seems to overfit the data for some of the RSTs such as $v'v'$, $w'w'$ and $u'v'$. The predictions made by the model are not bad in any sense however and a model architecture consisting of a different number of neurons and hidden layers, or even a different optimiser could have further enhanced thees results further still. It is observed that figures 56, 57 and 58 the prediction made by the model was not all so accurate for $u'w'$ and $v'w'$ and the behaviours of the model was often a bit chaotic for these stress tensors.

- **Analysis of Mean Squared Error Across Epochs:** The following figures present the progression of Mean Squared Error (MSE) for each component of the Reynolds Stress Tensor (RST) across three key epochs during the training of our simple neural network model. These epochs represent distinct phases in the model's training regimen, each providing insight into the model's capability to adapt and improve its predictive accuracy over time. The MSE values are crucial indicators of the model's performance, with lower values denoting more accurate predictions of the turbulent stresses in fluid dynamics simulations. This analysis aims to highlight the improvements and challenges encountered by the model as it learns from the DNS data.

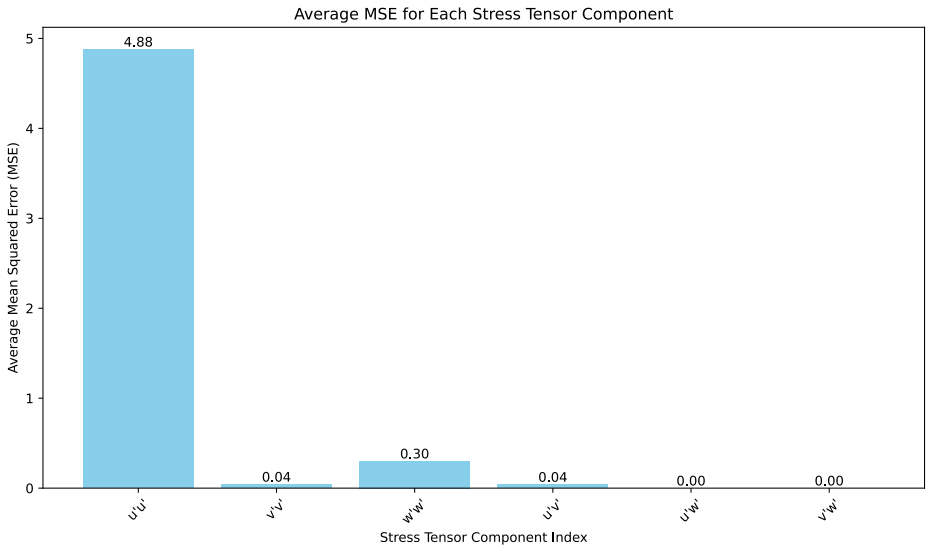

Figure 59: RST Average MSE - Epoch Number 10

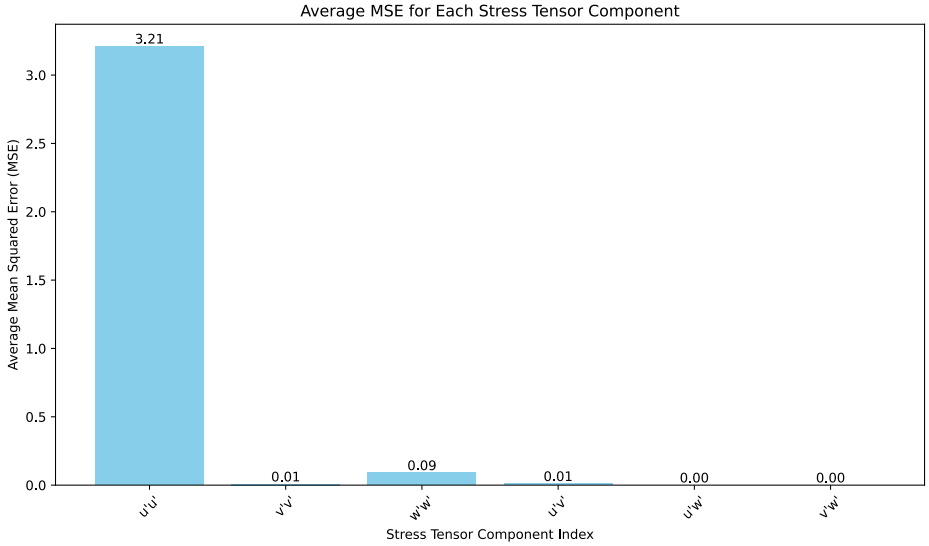

Figure 60: RST Average MSE - Epoch Number 50

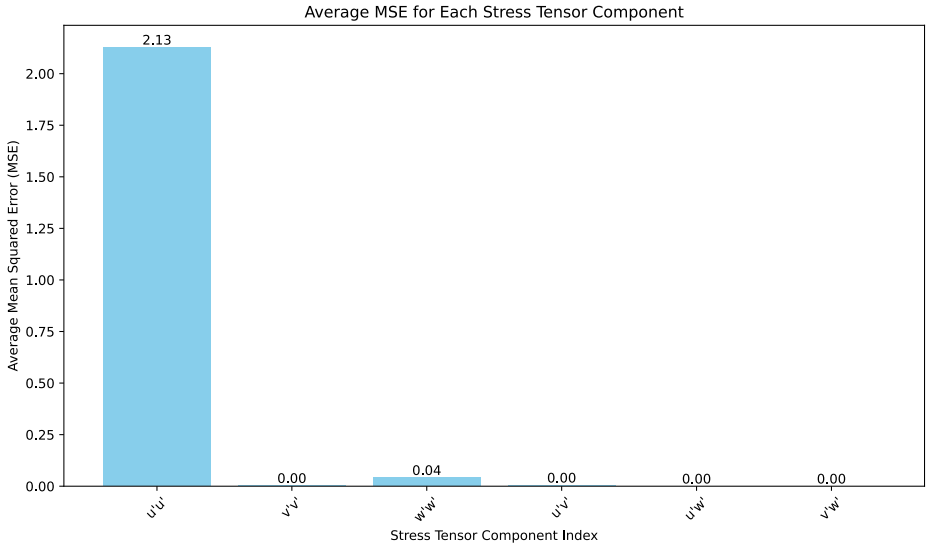

Figure 61: RST Average MSE - Epoch Number 100

The progression of mean squared error (MSE) for each Reynolds Stress Tensor (RST) component across Epochs 10, 50, and 100 reveals significant insights into the model's learning and adaptation over time, as illustrated in Figures 59, 60, and 61. At Epoch 10 (Figure 59), the MSE is highest for the $u'u'$ tensor at 4.88, indicating initial struggles in accurately capturing this component's dynamics, whereas other tensors like $v'v'$, $w'w'$, $u'v'$, $u'w'$, and $v'w'$ display considerably lower errors, suggesting either better initial model alignment or less complex dynamics in these components. By Epoch 50 (Figure 60), the MSE for $u'u'$ improves to 3.21, reflecting the model's enhanced capability to learn from the data, with other components maintaining low error rates. Continuing this trend, by Epoch 100 (Figure 61), the MSE for $u'u'$ further reduces to 2.13, while almost negligible errors in the other components affirm the model's high accuracy over the simpler dynamics of those tensors. This trajectory demonstrates how well the model is learning the physics underlying the flow dynamics and how well it can drastically improve its predictions over time, especially for the most difficult tensor components.

4.2.2 PINN

4.2.2.1 Predicting the Reynolds Stress Tensors

Environmental Setup

The various PINN models were implemented using the Jupyter Notebook environment, selected for its interactive and intuitive development interface. Additionally, leveraging widely used libraries such as Numpy and Torch for their pre-built functions used for creating the models. Alongside, Matplotlib played an essential role in visualising both the data and results for the creation of the entire PINN's pipeline.

Table 13 illustrates how the training part has been established for both RANS based PINN and PySindy-based PINN.

Version	Model	Flow Type	Rows (training)	rows (testing)
Splitted	RANS based PINN	Channel	4342	511
Splitted	RANS based PINN	Couette	891	255
Unified	RANS based PINN	Both	2811	766
Splitted	PySindy based PINN	Channel	130000	40000
Splitted	PySindy based PINN	Couette	50000	10000
Unified	PySindy based PINN	Both	120000	40000

Table 13: Illustration of the training setup for RANS-based PINN and PySindy-based PINN.

RANS and PySindy Comparison

Unified Flow Model The unified model aims to effectively generalise the characteristics of both the flow Channel and Couette configurations. To do so, this model is trained on datasets from both flow types. Besides, a series of different configurations were tested and evaluated using the MSE as a metric to identify the optimal result as demonstrated in Table 14 and Table 15.

$LayersNeurons$	10	20	30
2	4.87e-1	8.13e-1	12.21e-1
4	2.00e-1	6.41e-1	3.22e-1
8	1.88e-1	13.71e-1	3.61e-1

Table 14: Different (Unififed) RANS based PINN Model's Configuration

$LayersNeurons$	10	20	30
2	2.97e-1	5.18e-1	9.71e-2
4	2.05e-1	1.17e-1	12.13e-1
8	4.03e-1	4.33e-1	12.49e-1

Table 15: Different (Unified) PySindy based PINN Model's Configuration

From these tables, several observations emerge. Firstly, it is evident that increasing the number of neurons generally improves the results, even though, here the results do not align well with the DNS curves. However, caution is warranted as higher neuron counts can lead to overfitting. Secondly, comparing the PySindy-based PINN model to the RANS-based PINN model reveals slighlty worse performance in the latter. This contrast becomes even more apparent when examining Figures 62 and 63.

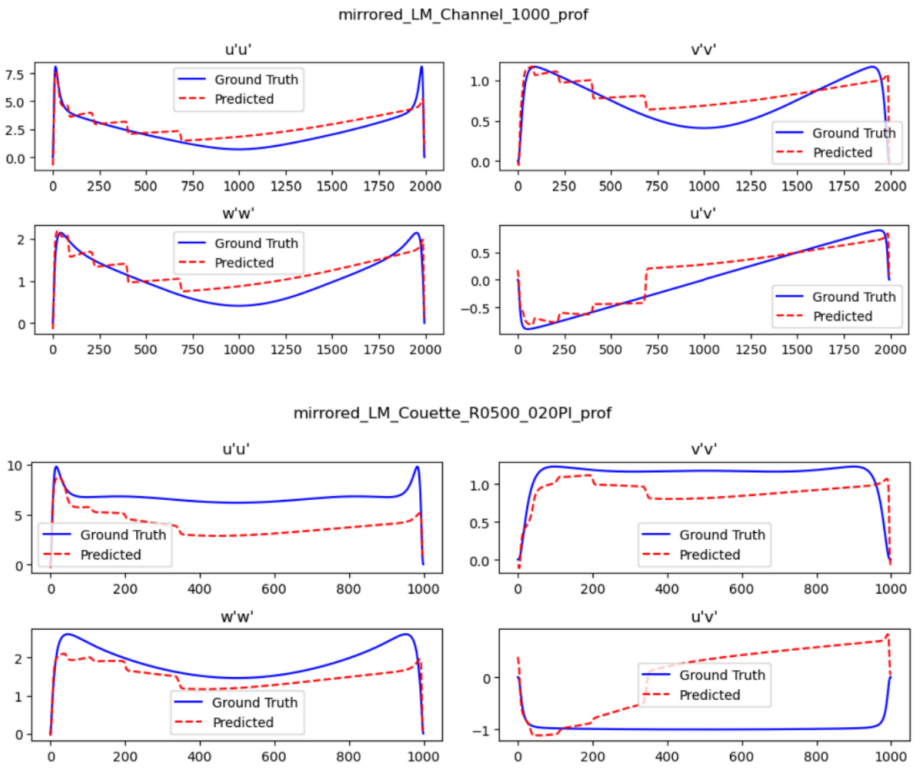

Figure 62: Unified RANS result (20 neurons & 4 hidden layers).

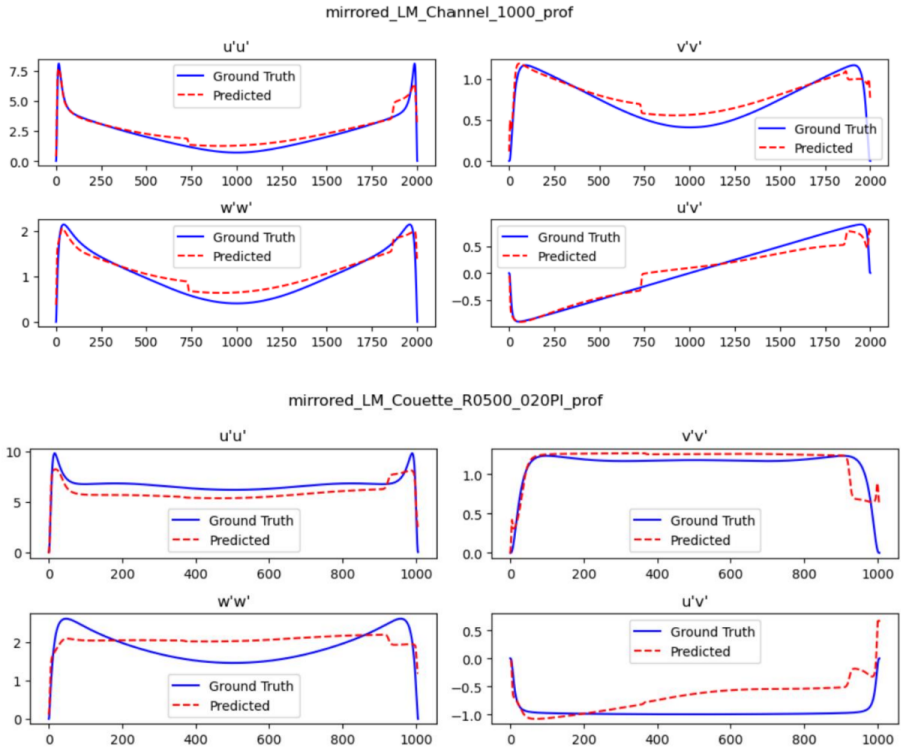

Figure 63: Unified PySindy result (20 neurons & 4 hidden layers).

As depicted in Figures 62 and 63, it is noticeable that both models demonstrate poor results as shown by the oscillations and sorts of noises which appear in the curves, which sometimes do not follow the DNS curves correctly. Although both models seem to give better results in predicting the Reynolds Stress Tensor for the Channel flow they struggle with the Couette flow (Appendix). This problem could arise due to several factors, including the lack of data from the Couette dataset and disparities in physical properties between the two flow configurations.

Consequently, a new approach has been chosen to address the challenges contributing to these outcomes. Rather than employing a single model to predict the Reynolds Stress Tensor for both flow types, two distinct models with identical structures were developed. One model was exclusively trained on Channel flow data, while the other on Couette flow data.

Split Flow Model This split model strategy aims to better capture the unique characteristics and behaviour of each flow type which might lead to an improved predictive performance. The Table 16 illustrates the configuration chosen for both RANS and Pysindy models.

Parameter	Value
Input	4
Hidden layer	10
Neurons	64
Output	6
activation function	Tanh
learning rate	2.5e-4
Optimiser	Adamax
epoch	2000

Table 16: Model's Configuration for both RANS & PySindy based PINN.

The subsequent results presented in this section will be derived from this configured approach.

Figures 64 and 65 demonstrate the results obtained with this new approach. It is noticeable that the results for both RANS based PINN model and PySindy based model are giving good results, even giving slightly better result on channel flow than the ones from unified model. However, a significant improvement is observed in the prediction for the Couette flow. Both models were able to successfully capture the complexity of the flows.

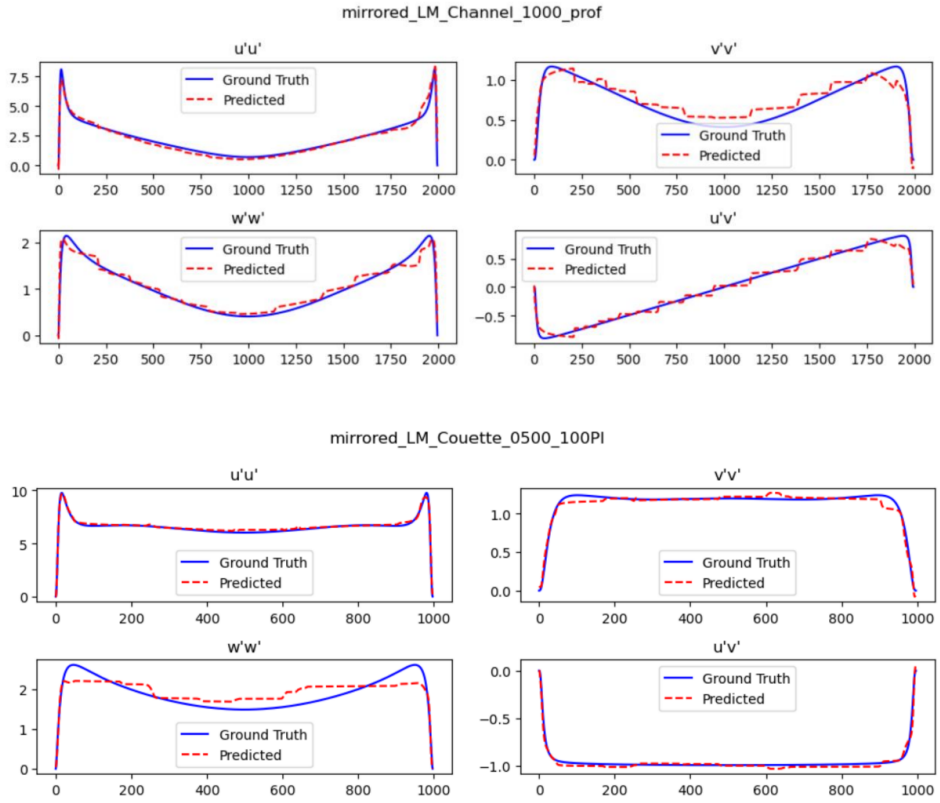

Figure 64: Split RANS Result

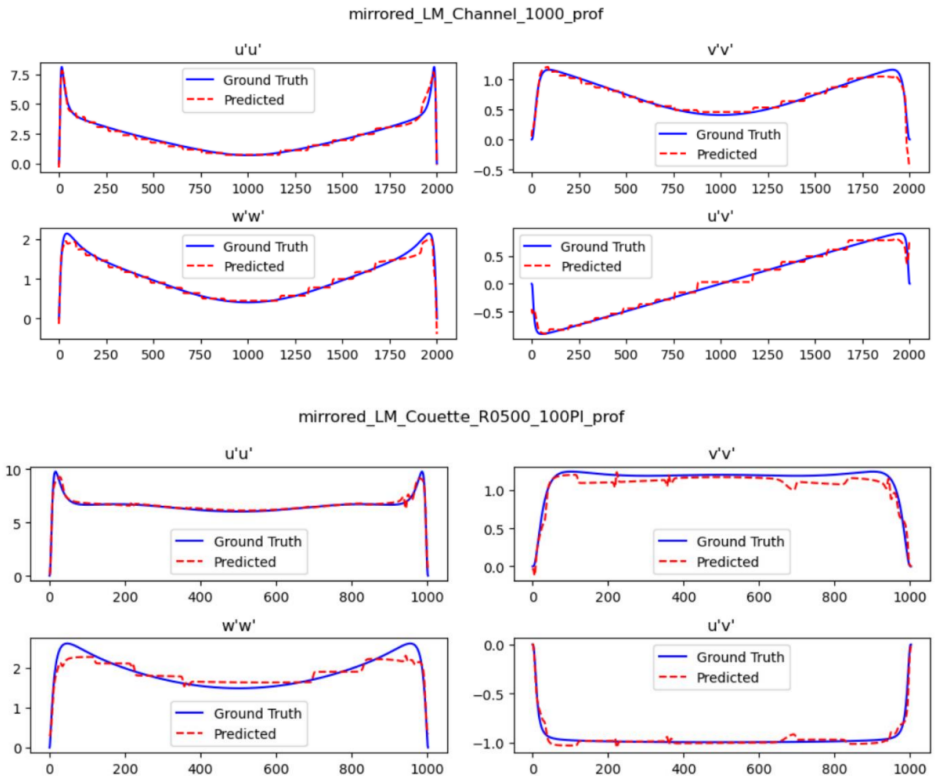

Figure 65: Split PySindy Result

To support these plots, table 17 illustrates the MSE when testing the model on unseen data

Model	Flow Type	MSE
RANS	Channel	5.27e-2
RANS	Couette	1.43e-2
PySindy	Channel	9.60e-3
PySindy	Couette	1.16e-2

Table 17: Model's Results

Based on both quantitative metrics as well as qualitative analysis, a conclusion analogous to the unified model can be drawn. Indeed, the PySindy based PINN seems to give a slightly better performance, suggesting that the equations derived by the PySindy are well suited for the problem at hand.

Now that the model is able to provide with efficient result for predicting the Reynolds Stress Tensor, it is possible to fetch from that prediction the turbulent kinetic energy using the relationship mentioned previously. As Figure 128 depicts, the result fits almost correctly the DNS curves demonstrating once again the efficiency of the split model version.

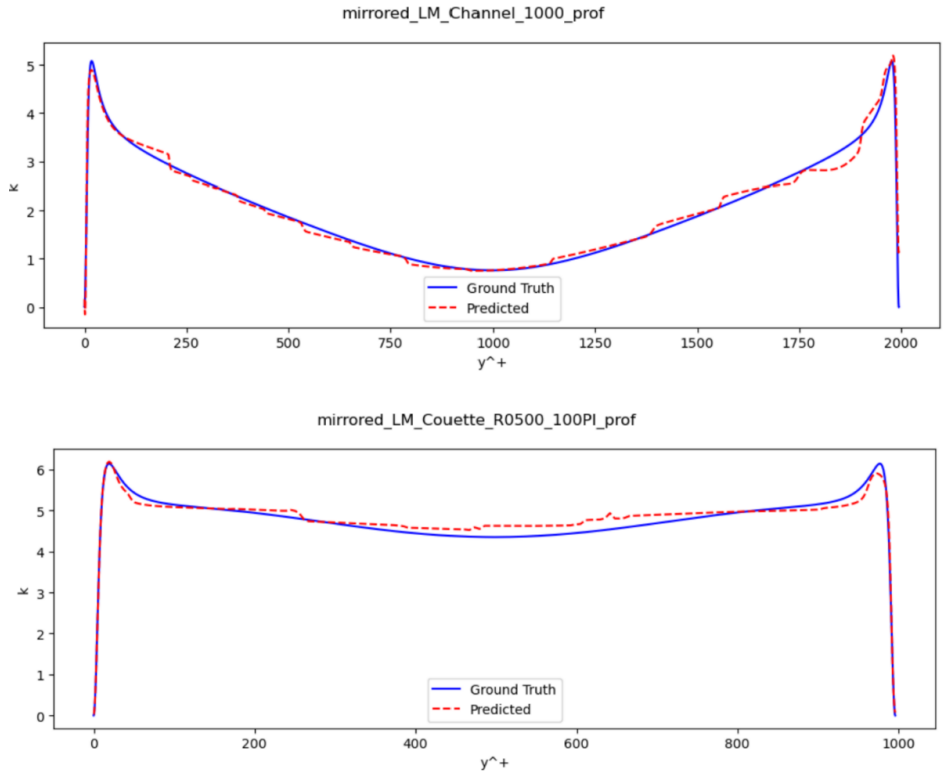

Figure 66: Kinetic turbulent energy result (RANS based PINN model).

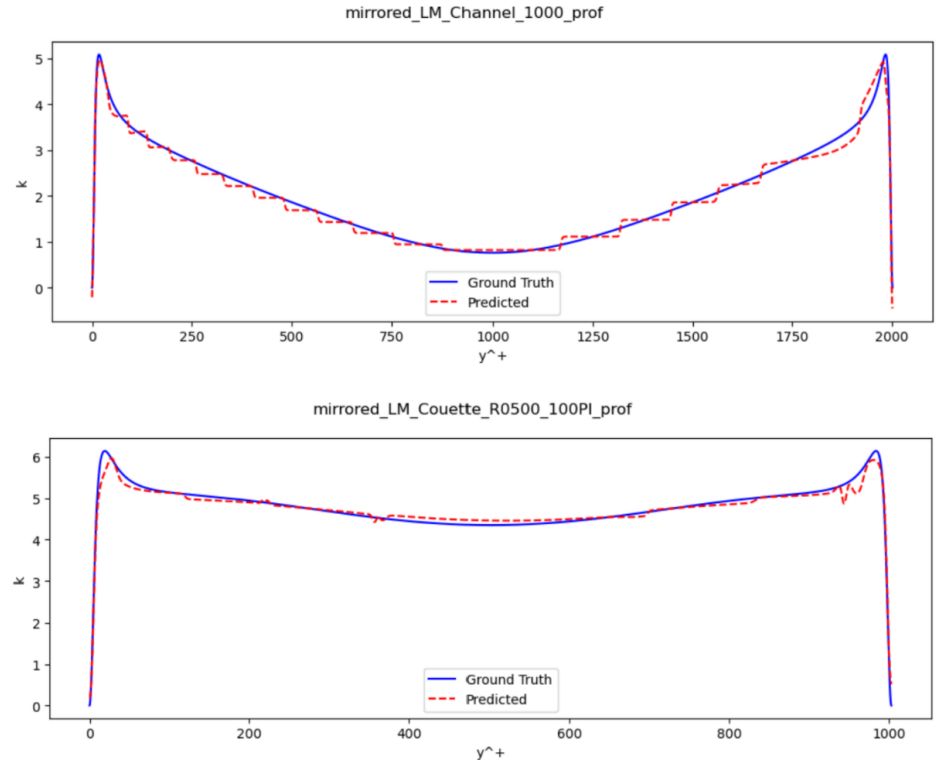

Figure 67: Kinetic turbulent energy result (PySindy based PINN model).

5 Final Model

After obtaining good results in all three previous sections, efforts were made to combine all of the models to obtain the Final Framework, which executed the last section of the Pipeline displayed in Figure 1. Up to this point, only the first section of the pipeline had been developed, which constituted the Equation Discovery section of the PySIndy and the training of the PINN model, with DNS inputs. This last section was made of three main parts, the RANS automated code, the Neural-Network Enhancement and the Comparison. The framework was compiled using Python, 3.11.4 version for several reasons. Initially, the objective was to ensure that the software was accessible to all users. However, it still required obtaining the RANS results, which was made possible by using a model that automatically obtained the RANS Reynolds Stresses and PINN inputs via Ansys ICEM and Fluent. Moreover, all of the code scripting and model generation was carried out in Python, except for the RANS automatisation, which was performed using Matlab.

To overcome this issue, the RANS code for both Channel and Couette flows was migrated to Python's environment inside the Visual Studio Code editor, as well as some modifications were made to ensure proper integration with the rest of the model and the scripting language, such as translating Matlab's built-in functions (*replace_between*); using Python's os library and subprocess functions for executing commands through the Operative System and terminal, and modifications in the data exportation to output a file with the Normalised Reynolds Stress Tensor and a file with the PINN input. For the PINN prediction part of the code, the code had to be modified from a Jupyter Notebook file to a script code, which did not involve an additional challenge as they were written using the same interpreter and language.

The last part of the code consisted of a comparative plotting tool, to enable a qualitative analysis of the Neural Network (Predicted) and SST k-w (RANS) Reynold Stresses. Additionally, all the DNS datasets with the Reynolds Stresses were stored in a folder and a piece of code was included that checked if the existing data corresponded to the Reynolds Number inputted, and in case it existed it also plotted the DNS results, labelled as ground truth.
Due to time constraints, it was not possible to implement some more features. The plan was to create a Graphic User Interface (GUI) that allowed the user to select the inputs such as Reynolds Number, Kinematic Viscosity, and Iterations. This would enable the user to obtain plots and files containing the Reynold Stress Tensor. Additionally, there was an aim to incorporate a quantitative comparison of all data. Although the codes for comparing the data were already created, they were not implemented into the current code.

In the technical work, three files were submitted:

- FINAL_MODEL_Channel
- FINAL_MODEL_Couette
- FINAL_MODEL_Channel_Extra_Cases

The first two models are individual predictors for Channel and Couette. Initially, it had been thought to combine the two models and have an if clause that asked for the model you wanted to predict. However, as it was an initial prototype, it was opted to present two models, one for each case. Also, all the inputs were predefined to match the properties of the 5 and 3 DNS cases. The final model provided a feature that

allows running different values of the Reynolds Number, and subsequently kinematic viscosity and friction velocity, with the use of a relation explained in the results section.

5.1 Code Requirements and Execution

This code has been tested within Cranfield's Virtual machine environment (*VMWork*) and managed to compile and run. However, in order to be able to run the code some steps must be followed.

1. The editor and terminal used was Visual Studio Code. This editor was selected as it allowed the installation of libraries by using its own terminal interface.
2. The language interpreter was conda 3.11.4
3. The *torch* library must be installed to be able to run the Neural Network part of the code. To achieve this the following inputs must be inputted in the terminal, figure 68.

```
1        pip install torch
2
```

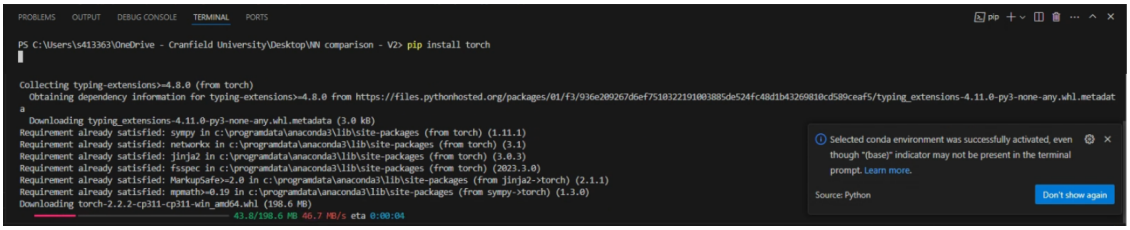

Figure 68: Installing PyTorch

4. In both the channel.py and couette.py functions, change the location of the executable for Ansys ICEM (icemcfd.bat) and Ansys Fluent (fluent.exe), in the ansys_executable and fluent_executable variables. The locations were defined as the ones in the Virtual Machine, figure 69.

```
# ANSYS ICEM executable
ansys_executable = 'C:\\Program Files\\ANSYS Inc\\v232\\icemcfd\\win64_amd\\bin\\icemcfd.bat'
macro_script = 'Mesh_New.rpl'
command = [ansys_executable, '-batch', '-script', macro_script]
subprocess.run(command, shell=True)
print('ICEM DONE----------------------------------------')

# ANSYS Fluent executable
fluent_executable = 'C:\\Program Files\\ANSYS Inc\\v232\\fluent\\ntbin\\win64\\fluent.exe'
fluent_script = 'Case_new.jou'
command_fluent = [fluent_executable, '3d', '-g', '-i', fluent_script]
subprocess.run(command_fluent, shell=True)
print('FLuent DONE----------------------------------------')
```

Figure 69: Location Change

5. Open the main function and run the code.

5.1.0.1 Modifications

Inside the codes, two options were created to test the code. Firstly, in option one the code automatically executes Ansys and generates the "_PINN.csv" files which are directly inputted into the predict function which executes the neural network. This is the main intention of the code, however, obtaining fully converged simulations can take a considerable amount of time. On the other hand, option two relies on previously converged simulations so there is no necessity of using RANS. To use this option, only the "_PINN.csv" files from the RANS_Datasets Folder have to be copied into the main directory.

5.2 Results

The model was tested with 11 cases divided in three groups:

1. 5 Channel datasets with known DNS Results (Reynolds 182, 501, 1001, 1996 and 5186)
2. 3 Couette datasets with known DNS Results (Reynolds 93, 221, 543)
3. 3 Channel datasets with unknown DNS Results (Reynolds 1500, 4000 and 6000)

To generate this last set of properties (Friction Reynolds Number, Kinematic viscosity and friction velocity) a relationship between the three had to be defined.

Firstly, by using the friction Reynolds Number definition, equation (66) becomes apparent:

$$Re_{tau} = \frac{u_{tau} * h}{\nu} \tag{66}$$

Being, h the channel half width, the following relation, equation (67) can be obtained:

$$\frac{Re_{tau} * \nu}{u_{tau}} = 1 \tag{67}$$

Therefore by figuring out the values of the Reynolds Number and kinematic viscosity, the friction velocity can be obtained. However, we did not manage to obtain a demonstration of the relationship between these two variables, however, it was seen that the Relation between the Reynolds Number and Kinematic Viscosity could be approximated as an exponential function, and with the use of Excel, the following relationship seen in figure 70 was obtained.

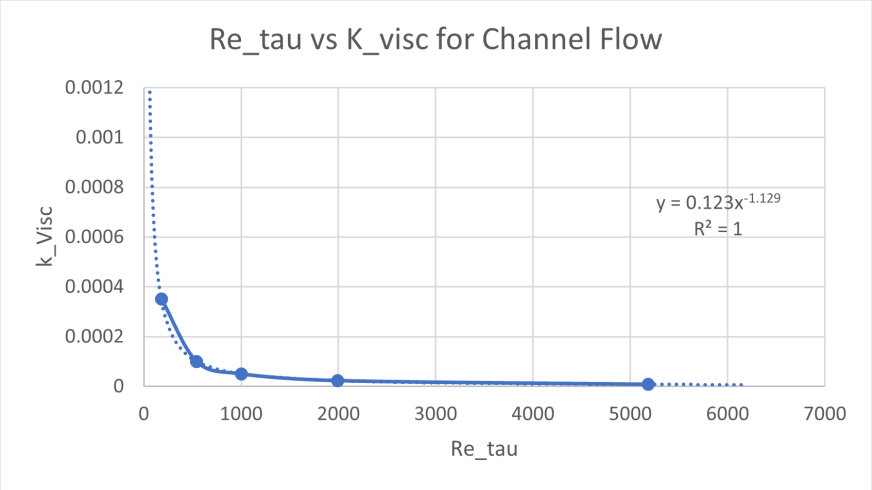

Figure 70: Reynolds Number and Kinematic Viscosity Relationship

This can be translated into equation (**??**):

$$\mu = 0.123 * Re_{tau}^{-1.129} \tag{68}$$

Obtaining the following predictions for this Reynolds Numbers: 1500, 4000 and 6000 in figure 71.

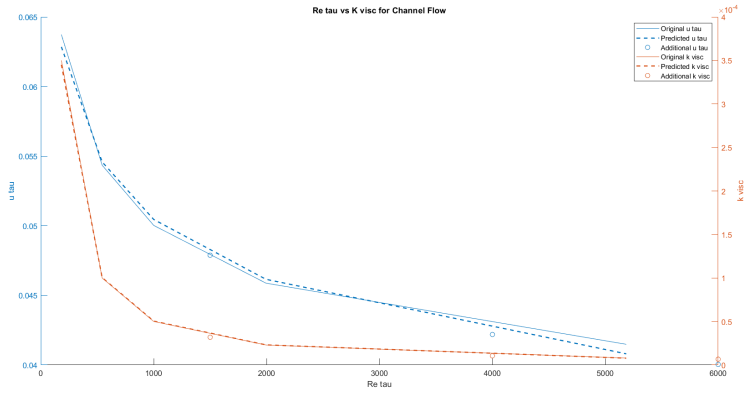

Figure 71: A Residual Connection

5.2.1 Channel Datasets with Known DNS Results

In this comparison, only the Reynolds 182 and Reynolds 5182, seen in figure 72 will be taken into account. All the results can be found in the Appendix D. By conducting a qualitative analysis of the plots, it can be observed that there has been an overall improvement in the normal stresses. However, there is still a considerable gap between the predicted and DNS values. When focusing on individual variables, it can be seen that the normal stress in the x-direction, $u'u'$, which was a major issue for RANS, has been partially resolved for all cases. The RANS results are now more accurate for the more turbulent cases, achieving almost one-to-one matches for the 5186 case along the middle section. The Y and Z-direction components have also shown noticeable improvements, despite the RANS results being close to the DNS results already,

especially for the $w'w'$ value. However, the $'u'v'$ component has provided worse results than the RANS stresses. This is because the RANS prediction for these components was accurate, so instead of leaving the results unmodified, it ended up predicting worse results.

Although the results improved, there were two major issues with this model. Firstly, it can be seen some fluctuations and oscillations in the prediction, did not occur when DNS was used as input. This could be attributed to a mismatch in the meshes between the DNS and RANS, as the model was trained with different grid points than the input.

Secondly, and more importantly, there are mismatches in the wall values, as the values of the stresses should be 0 in all components of the Reynolds Stress Tensor, as the gradients and turbulent kinetic are 0 in the Channel Flow. This could be appreciated in the right wall value for the $w'w'$ component of the Reynolds 1001 and the right wall of the $u'v'$ for all the cases.

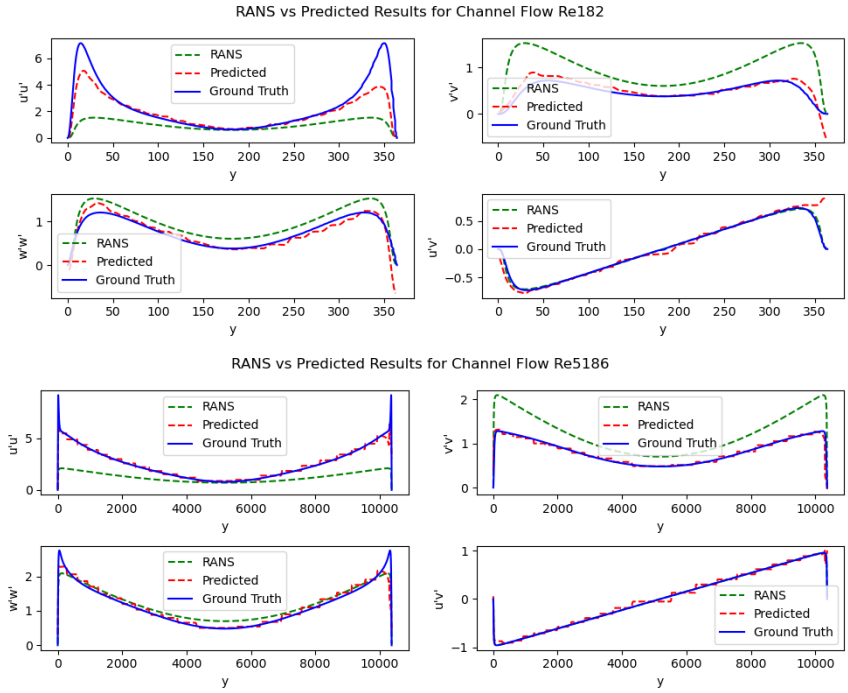

Figure 72: RANS and PySindy based PINN with RANS Input Comparison for Channel Flow Re 182 and 5200

Looking at the 1001 case, figure 73, which was unseen by the PINN model, it can be seen that the results were greatly improved, indicating that the model is capable of predicting new results and does not suffer from overfitting. Additionally, similar behaviours can be seen from the prediction of seen data, and even it has been able to provide better results than the Reynolds 182 Case.

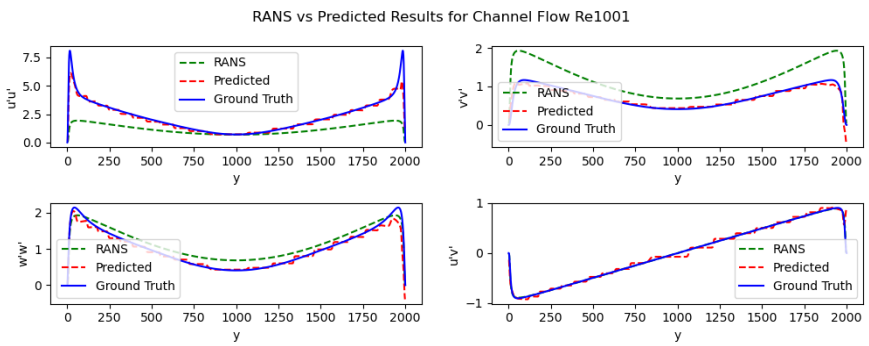

Figure 73: RANS and PySindy based PINN with RANS Input Comparison for Channel Flow Re 1000

Having established a relation to predict values of the kinematic viscosity based on equation (68) the cases for Reynolds Number 1500, 4000 and 6000 were launched. As there is no DNS dataset to compare, only the RANS and predicted values were plotted. From the RANS vs DNS comparison, the RANS results underestimated the values for $u'u'$, $v'v'$ and $w'w'$, as well as failed to capture the steep gradient in the near-the-wall regions. Although there is no DNS dataset to compare the predictions, it ban be observed that the prediction corrects those two aspects, so it could be expected that the prediction is closer to the ground truth than the RANS. Furthermore, in this plots, the mismatches in the wall values can be seen more clearly in figure 74.

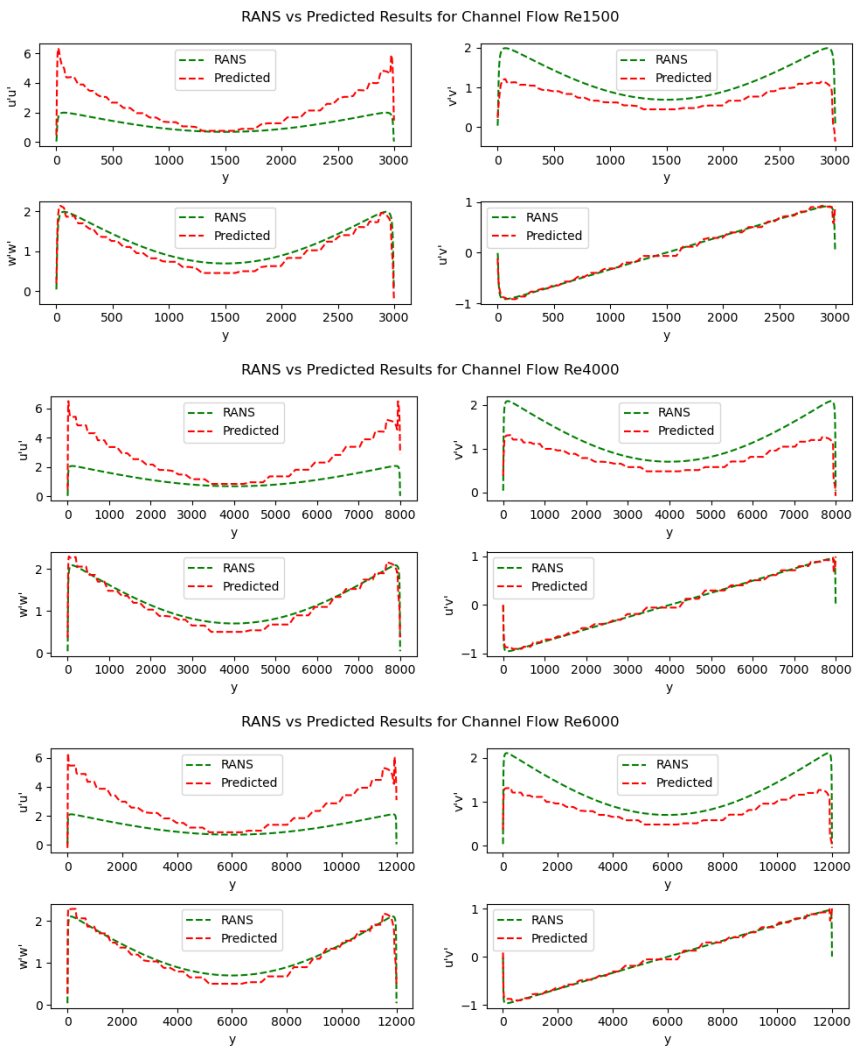

Figure 74: RANS and PySindy based PINN with RANS input comparison for Channel Flow Re 1500, 4000 and 6000

5.2.2 Couette Results

The last model executed was the PySindy-based PINN for Couette flows for the three test cases. The results, captured in figure 75 align with the ones observed for the Channel flow, characterized by an overall improvement in the normal stress component but inaccuracies in the $u'v'$. However, some differences compared to the Channel arise when the near-the-wall region prediction is analysed. Firstly for the Reynolds 93 case, the predicted curve for $u'u'$ managed to predict the correct behaviour, but the $w'w'$ and $u'v'$ curves had exaggerated features, such as the steep positive gradient for the forer and steep decreasing gradients for the latter, which could be attributed to the lack of training data.

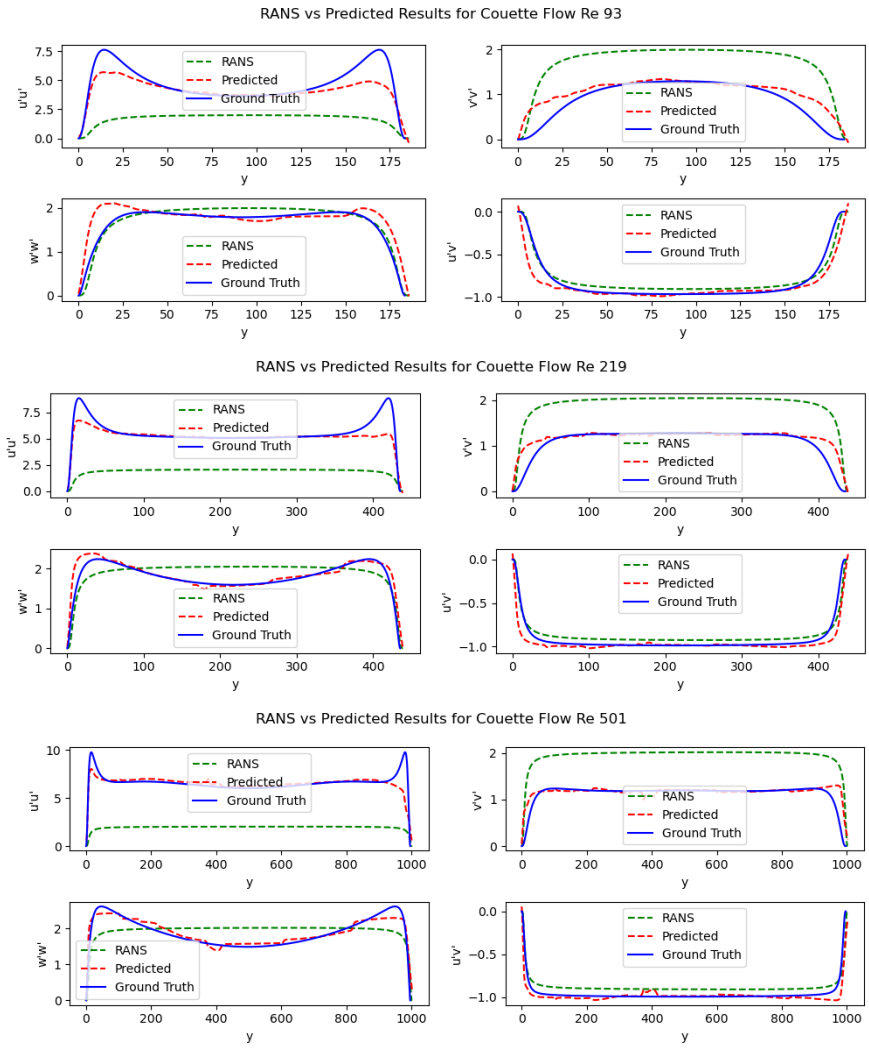

Figure 75: RANS and PySindy based PINN with RANS input comparison for Couette Flow Re 93, 220 and 5200

Finally, after computing the RMSE of both PINN and RANS predictions against the DNS or ground truth and compared against each other in the histograms at figures 76 and 77. The overall improvement in the prediction of $u'u'$, $v'v'$ and $w'w'$ as well as the detriment in the prediction $u'v'$ can be quantified and compared. As the $'u'u$ presented the most error for both Channel and Couette, it was expected that a major improvement could be made there. Additionally, $v'v'$ and $w'w'$ predictions have also noticeable improvements, but the quantification of the error shows how inaccurate the $u'v'$ component prediction is. Also, it was noticeable that both RANS and PINN had similar behaviours for both Channel and Couette.

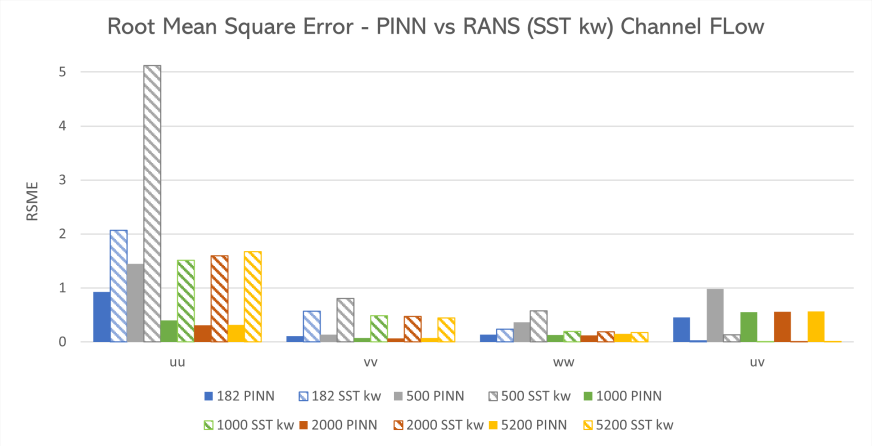

Figure 76: RANS and PySindy based PINN RSME Comparison for all Channel and Couette Flows

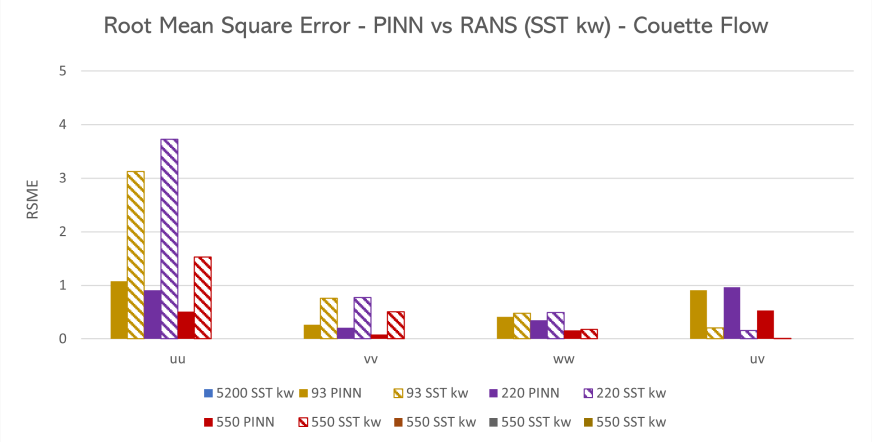

Figure 77: RANS and PySindy based PINN RSME Comparison for all Channel and Couette Flows

Finally, to conclude this section, table 18 showcasing the improvement in the reduction of the errors for all cases has been computed, highlighting the behaviours showcased in the previous paragraphs.

Table 18: RSME improvements by PINN against RANS for all Channel and Couette Test Case

	Channel					Couette		
	Re 182	Re 500	Re 1000	Re 2000	Re 5200	Re 93	Re 220	Re 500
uu	55.2%	71.8%	73.5%	80.7%	81.2%	65.7%	75.6%	66.8%
vv	80.8%	83.2%	84.9%	86.3%	83.4%	65.3%	73.5%	83.9%
ww	42.7%	37.2%	33.5%	35.7%	11.6%	14.0%	29.7%	13.1%
uv	-1994.1%	-655.0%	-5620.5%	-7978.5%	-8690.0%	-345.8%	-518.8%	-4115.2%

6 Discussion and Conclusions

The project was significantly impacted by constant shifts in the main goals and approaches taken over time and this led to the creation of several models that were not utilised in the final model; despite this, they still provided insight and knowledge to achieve the final solution. In the end, a PINN-based neural network was developed constrained with pySINDy governing equations with the capability to accurately predict the Reynolds Stresses. These contributions have already formed the basis and provided insight for furthering and improving this methodology separately. However, the main achievement of this project was the integration of three different techniques and models into a single framework, capable of predicting and enhancing the Reynolds stresses for a test case with two different boundary conditions. This initial approach has already successfully predicted values with RSME of 0.0096 error for unseen Channel Flow data and 0.016 for Unseen Couette data when given DNS input and compared to (DNS). However, when tested with RANS inputs, predicted values with 0.201 and 0.486 (Couette) error compared to the values of RSME 2.1 (Channel) and 1.6 (Couette) of RANS compared to DNS, achieving an improvement of 63.9% and 54.6%, for unseen Channel and Couette test cases respectively. Although the results appear promising, several improvements can be made to the model. Since this was just the first iteration and due to time constraints, there was no time left for refinements. Some of these improvements include having more data to train, as datasets typically involve thousands of points, as well as further refining the PINN constraints to respect the boundary conditions.

Regarding the SINDy approach, the PySINDy with FROLS optimisers was utilised to capture turbulence characteristics for both low and high Reynolds numbers. However, a few challenges were encountered, and several methods were adapted to overcome them. PySINDy assumes that the data is clean; high levels of noise can affect its ability to identify accurate models. Therefore, clean and robust preprocessing data was provided with smoothing and mirroring processing. PySINDy works best with time-series data that exhibit clear dependencies. However, the data used was time-averaged data, so delta y was used as delta time to ensure proper functionality. There is a risk of overfitting when PySINDy identifies too many terms, which can lead to models that are too complex. To prevent this, the equation was validated with genetic algorithms. In terms of $\frac{dv'v'}{dy}$, it showed the potential to capture more nonlinear characteristics than sindy, However, the method failed to exhibit any improvement, due to insufficient iterations resulting from time constraints.

In conclusion, the obtained governing equations suggest that the PySINDy approach is adaptable and capable of uncovering various governing equations with differing complexities. However, predicting turbulence models remains a significant challenge.

Regarding the software, improvements such as a GUI and full integration of both cases in the same code were considered and have been taken into account for future development. Additionally, one of the main issues of the code is the time demand required to fully converge a simulation. Currently, the code only allows for changing the iterations of the first-order and second-order upwind schemes and not the convergence. To do that, one needs to manually change the values in the Case.txt file. Also, although the code was expected to run on Cranfield's High-Performance Computing Facility, there were issues with how to execute submission commands from a script. However, the code generates the journal file and mesh, as well as a submission file necessary to run the Fluent simulation, and an option to read those results.

Regarding team management, although the final integration of the code was done last week, the work has been consistent throughout the whole extent of the project. The initial grouping of members into teams

and subdivision of tasks was both a risk factor and a safety barrier. Not having one part of the code would negatively impact the output of the project but would not directly impact the other teams' work. In the end, all three teams worked collaboratively and at the same pace, resulting in a final model that integrates the work of everyone involved.

Individually, from the CED part, this study has involved two major sections. Firstly, defining the viscous models has been challenging, but the results obtained for both Channel Flow 182 and Channel Flow 5200, although out of the scope of this project, have been insightful in understanding how the different models thrive and fail in predicting different regions of the wall and different turbulent flows. On the other hand, the automation of the CFD simulations has been useful in launching all the simulations with all the models. By changing values in the code and templates, the simulations could be launched both on the virtual machine and HPC. Furthermore, it allows for implementation in the pipeline, allowing future users who have access to ANSYS ICEM and Fluent not to depend on additional effort to provide PINN inputs, as currently, CFD is essential to create those inputs.

Finally, future research involves refining the code and providing larger datasets. Once the code has been refined, additional work could be done in generating the necessary inputs with the use of only scalars such as the Reynolds Numbers and kinematic viscosity, either by using Neural Networks or SINDy to provide the PINN inputs, completely removing the usage of CFD software. Finally, the last step is to expand this model for bi-directional and tri-directional flows, using test cases such as the hill data.

7 References

7.1 Bibliography

[1] Smits AJ. Lectures in Fluid Mechanics, Viscous Flows and Turbulence. Princeton, NJ: Princeton University. (MAE 552, Viscous Flows and Boundary Layers; MAE 553, Turbulent Flow).

[2] NASA Langley Research Center. Spalart-Allmaras Turbulence Model. Turbulence Modeling Resource. https://turbmodels.larc.nasa.gov/spalart.html.

[3] Fluent Inc. Fluent Guide 5.1. Chapter 10: Modelling Turbulence. November 28, 2001

[4] Wilcox DC. Turbulence Modeling for CFD. DCW Industries; 1994. Wilcox, D. C. (2006), Turbulence Modeling for CFD (3rd ed.), DCW Industries, ISBN 0963605100

[5] Two-equation eddy-viscosity turbulence models for engineering applications.F. R. Menter. AIAA Journal 1994 32:8, 1598-1605. Available at https://arc.aiaa.org/action/showCitFormats?doi=10.2514%2F3.12149

[6] Kolmogorov, A. N. "Dissipation of Energy in the Locally Isotropic Turbulence." Proceedings: Mathematical and Physical Sciences, vol. 434, no. 1890, 1991, pp. 15–17. JSTOR, http://www.jstor.org/stable/51981. Accessed 1 May 2024.

[7] P.R Spalart, Strategies for turbulence modelling and simulations, International Journal of Heat and Fluid Flow, Volume 21, Issue 3, 2000, Pages 252-263, ISSN 0142-727X,

[8] Lee M, Moser RD. Extreme-scale motions in turbulent plane Couette flows. Journal of Fluid Mechanics. 2018;842:128-145. doi:10.1017/jfm.2018.131

[9] Lee, M., Moser, R. D. (2015). Direct numerical simulation of turbulent channel flow up to Re 5200. Journal of Fluid Mechanics, 774, 395-415. doi:10.1017/jfm.2015.268

[10] S. L. Brunton, J. L. Proctor, J. N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*. Proceedings of the National Academy of Sciences, 113(15):3932–3937, 2016. https://browse.arxiv.org/pdf/2004.08424v1.

[11] B. M. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. N. Kutz, S. L. Brunton, *PySINDy: A Python package for the sparse identification of nonlinear dynamical systems from data*. Journal of Open Source Software.https://arxiv.org/pdf/2111.08481.

[12] A. A. Kaptanoglu, B. M. de Silva, U. Fasel, et al., *PySINDy: A comprehensive Python package for robust sparse system identification*. Journal of Open Source Software, 2022. https://www.semanticscholar.org/reader/3e89a9f1301a46d011ff04d7c05ae5134f05961a.

[13] R. Rubini *Sparsification Techniques for Reduced Order Models of Turbulent Flows* https://eprints.soton.ac.uk/467602/1/Final_Thesis.pdf.

[14] Kaptanoglu AA, Zhang L, Nicolaou ZG, Fasel U, Brunton SL. *Benchmarking sparse system identification with low-dimensional chaos*. arXiv. 2023 Feb 4. Report No.: 2302.10787. Available from: https://arxiv.org/abs/2302.10787]

[15] Raissi M, Perdikaris P, Karniadakis GE. *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*. Journal of Computational Physics. 2018;378:686-707.

[16] Raissi M, Yazdani A, Karniadakis GE. *Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations*. Science. 2020;367(6481):1026-1030.

[17] Mao Z, Jagtap AD, Karniadakis GE. *Physics-informed neural networks for high-speed flows*. Computer Methods in Applied Mechanics and Engineering. 2020;360:112789.

[18] Guastoni L, Güemes A, Ianiro A, Discetti S, Schlatter P, Azizpour H, Vinuesa R. *Prediction of wall-bounded turbulent flows via sparse modelling*. Physical Review Fluids. 2021;6(3):034602.

[19] Zhu L, Zhang W, Sun X, Liu Y, Yuan X. *Turbulence closure for high Reynolds number airfoil flows by deep neural networks*. Aerospace Science and Technology. 2021;110:106452.

[20] Pandey S, Schumacher J, Sreenivasan KR. *A perspective on machine learning in turbulent flows*. International Journal of Heat and Fluid Flow. 2020;81:108544.

[21] Iyer KP, Schumacher J, Yeung PK. *Fractal iso-level sets in high-Reynolds-number scalar turbulence*. Physical Review Fluids. 2020;5(4):044605.

[22] Majchrzak M, Marciniak-Lukasiak K, Lukasiak P. *A Survey on the Application of Machine Learning in Turbulent Flow Simulations*. Energies. 2023;16(4):1755.

[23] Fang R, Sondak D, Protopapas P, Duraisamy K. *Neural network models for the anisotropic Reynolds stress tensor in turbulent channel flow*. Journal of Turbulence. 2019;20(1):1-19.

[24] Vignon A, Guastoni L, Schlatter P, Azizpour H, Vinuesa R. *Deep reinforcement learning for flow control in Rayleigh-Bénard convection*. Physical Review Fluids. 2021;6(6):064602.

[25] Guastoni L, Güemes A, Ianiro A, Discetti S, Schlatter P, Azizpour H, Vinuesa R. *Perspectives on predicting and controlling turbulent flows through machine learning*. Physics of Fluids. 2021;33(6):061401.

[26] Obiols-Sales O, Vishnu A, Malaya N, Chandramouli Sharan A. *CFDNet: A deep learning-based accelerator for fluid simulations*. In: Proceedings of the 34th ACM International Conference on Supercomputing. 2020:1-12.

[27] Maulik R, San O, Jacob JD, Crick C. *Sub-grid scale model classification and blending through deep learning*. Journal of Fluid Mechanics. 2019;870:784-812.

[28] Brunton SL, Noack BR, Koumoutsakos P. *Machine learning for fluid mechanics*. Annual Review of Fluid Mechanics. 2020;52:477-508.

[29] Wang R, Kashinath K, Mustafa M, Albert A, Yu R. *Towards physics-informed deep learning for turbulent flow prediction*. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining. 2020:1457-1466.

[30] Alpaydin, E. (2020). *Introduction to Machine Learning* (4th ed.). MIT Press.

[31] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[32] Haykin, S. (2009). *Neural Networks and Learning Machines (3rd ed.)*. Pearson.

[33] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533-536.

[34] University of Ohio *Polynomial and Spline Interpolation* http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/lecture19.pdf

[35] Wikipedia contributors *Spline interpolation — Wikipedia, The Free Encyclopedia* https://en.wikipedia.org/w/index.php?title=Spline_interpolation&oldid=1195463638

[36] M. Raissi, P. Perdikaris, G.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, Volume 378,2019,Pages 686-707,ISSN 0021-9991, Available at: (https://www.sciencedirect.com/science/article/pii/S0021999118307125)

[37] Raissi M, Perdikaris P, Karniadakis GE. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. Division of Applied Mathematics, Brown University, Providence, RI, USA; Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, USA. 2017. Available from: https://arxiv.org/pdf/1711.10561

7.2 Images

[9] Osborne Reynolds (Scan from book) [Public domain], via Wikimedia Commons (File: Reynolds observations turbulence 1883.svg). Retrieved from: https://commons.wikimedia.org/wiki/File:Reynolds_observations_turbulence_1883.svg (Accessed on 30 April 2024)

[10] Left: Comparison of a DNS (a), LES (b) and RANS (c) simulation of a jet flow (Italian Agency For New Energy Technologies 2006). Right: Schematic differentiation between RANS, LES and DNS modelling (Deck et al. 2014). Retrieved from: https://www.researchgate.net/figure/Left-Comparison-of-a-DNS-a-LES-b-and-RANS-c-simulation-of-a-jet-flow-Italian_fig1_330765625 (Accessed on 30 April 2024)

[11] Banihani E. and El Haj Assad M. (2018). "Boundary-Layer Theory of Fluid Flow past a Flat-Plate: Numerical Solution using MATLAB." Retrieved from: https://www.researchgate.net/figure/Velocity-boundary-layer-development-on-a-flat-plate-2_fig1_323218834(Accessed on 30 April 2024)

[12] Subdivisions of the Near-Wall Region. ANSYS FLUENT 12.0 Theory Guide. Fluent Inc. Retrived from https://www.afs.enea.it/project/neptunius/docs/fluent/html/th/node98.htm (Accessed on 30 April 2024)

8 Appendix A: Derivation of Navier-Stokes Equation

8.1 Appendix A1: Continuity Equation for compressible fluids

This demonstration of the continuity equation for compressible fluids has been obtained from Professor Alexander Smits's Viscous Flows and Turbulence Lectures Notes in Fluid Mechanics pages 15 and 16 [1].

This demonstration starts from the conservation of mass principle where:

$$\frac{d}{dt}\int_v \rho \, dv = -\int_A n \cdot \rho \mathbf{V} \, d\mathbf{A} \tag{A1.1}$$

Since the volume is fixed, the partial derivative term can be brought inside the integral (A1.2) and by using the divergence theorem the integral can be transformed into a volume integral (Eq. A1.3).

$$\int \frac{d\rho}{dt} \, dv + \int_A n \cdot \rho \mathbf{V} \, d\mathbf{A} = 0 \tag{A1.2}$$

$$\int \left(\frac{d\rho}{dt} + \nabla \rho \cdot \mathbf{V} \right) dv = 0 \tag{A1.3}$$

The integral can be reduced to 0 due to the volume being arbitrary, therefore obtaining equations a-4.

$$\rho \nabla V = -\frac{D\rho}{Dt} \tag{A1.4}$$

By applying the nabla operator from the right term of the equation (Eq. A1.5) and expanding the terms obtained using the chain rule (Eq. A1.6) the equation can be divided into two components: the divergence of the velocity field in the cartesian form and the rate of variance in the density respect of the cartesian coordinates in space.

$$\nabla \rho \cdot \mathbf{V} = \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} = -\frac{d\rho}{dt} \tag{A1.5}$$

$$\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} = u\frac{\partial \rho}{\partial x} + \rho\frac{\partial(u)}{\partial x} + v\frac{\partial \rho}{\partial y} + \rho\frac{\partial(v)}{\partial y} + w\frac{\partial \rho}{\partial z} + \rho\frac{\partial(w)}{\partial z} + = -\frac{d\rho}{dt} \tag{A1.6}$$

Divergence of the Velocity Field

$$\rho \left(\frac{\partial(u)}{\partial x} + \frac{\partial(v)}{\partial y} + \frac{\partial(w)}{\partial z} \right) = \rho \nabla V \tag{A1.7}$$

Rate of Variance in the Density

$$u\frac{\partial \rho}{\partial x} + v\frac{\partial \rho}{\partial y} + w\frac{\partial \rho}{\partial z} \tag{A1.8}$$

This last term can be moved to the left-hand side and combined with the partial derivative of the density in time and can be simplified into the material derivative $\frac{D}{Dt}$, which describes how the total quantity changes through time.

$$-u\frac{\partial(\rho)}{\partial x} - v\frac{\partial(\rho)}{\partial y} - w\frac{\partial(\rho)}{\partial z} - \frac{d\rho}{dt} = -\frac{D\rho}{Dt} \tag{A1.9}$$

By combining the material derivative (Eq. A1.8) and the divergence of the velocity field (Eq. A1.9), the final expression for the continuity equation can be obtained:

$$\rho \nabla V = -\frac{D\rho}{Dt} \tag{A1.10}$$

8.2 Appendix A2: Euler and Navier-Stokes Momentum Equation for compressible fluids

This demonstration of the momentum equation or Euler's Equation for compressible fluids has been obtained from Professor Alexander Smits's Viscous Flows and Turbulence Lectures Notes in Fluid Mechanics pages 19 and 20 [1].

This derivation starts by applying Newton's Second law for an inviscid fluid, therefore obtaining the following Equation:

$$F_{\text{ext}} - \int \nabla \cdot \rho \, dA + \int \rho \mathbf{g} \, dv = \frac{\partial}{\partial t} \int \rho \mathbf{V} \, dv + \int (n \cdot \rho \mathbf{V}) \mathbf{V} \, dV \tag{A2.1}$$

Where F_{ext} are the external forces acting on the fluid, $\int \nabla \cdot \mathbf{p} \, dA$ represents the force due to the pressure, $\int \rho \mathbf{g} \, dv$ the forces acting on the volume due to gravity, $(\partial/\partial t) \int \rho \mathbf{v} \, dV$ the variation in momentum and $\frac{\partial}{\partial t} \int \rho \mathbf{v} \, dV$ accounts for the convective momentum term of the equation.

Assuming a fixed volume, non-existing external forces, and moving the derivative inside the integral is obtained in Eq. A2.2.

$$- \int \nabla \cdot \rho \, dA + \int \rho \mathbf{g} \, dv = \int \frac{\partial \rho \mathbf{V}}{\partial t} dv + \int \rho \mathbf{V} \, dv + \int (\nabla \cdot \rho \mathbf{V}) \mathbf{V} \, dV \tag{A2.2}$$

By using the product rule of vector calculus [1] to the term ρV and transform the right and left-hand side of the equation into volume integrals (Eq. A2.3) and simplifying, it leads to equation A2.4.

$$- \int (\nabla \rho + \rho \mathbf{g}) \, dv = \int \left(\frac{\partial \rho \mathbf{V}}{\partial t} + (\nabla \cdot \rho \mathbf{V}) \mathbf{V} + (\rho \mathbf{V} \cdot \nabla) \mathbf{V} \right) dv \tag{A2.3}$$

$$\int \left(\rho \frac{\partial \mathbf{V}}{\partial t} + \rho (\mathbf{V} \cdot \nabla) \mathbf{V} + \nabla \rho - \rho \mathbf{g} \right) dv = 0 \tag{A2.4}$$

Finally, because this analysis deals with an arbitrary volume, the integrand can be equalised to 0 (Eq. A2.5), obtaining the final equation A2.5

$$\rho \left(\frac{\partial \mathbf{V}}{\partial t} + (\mathbf{V} \cdot \nabla) \mathbf{V} \right) + \nabla \rho - \rho \mathbf{g} = 0 \tag{A2.5}$$

According to the definition of the material derivative, the expression $[\partial \mathbf{V}/\partial t + (\mathbf{V} \cdot \nabla)\mathbf{V}]$ can be replaced with the material derivative of $\mathbf{V}$, leading to the final Euler Equation:

$$\rho \frac{D\mathbf{V}}{dt} = -\nabla \rho + \rho \mathbf{g} \tag{A2.6}$$

Navier-Stokes Momentum Equation includes a term that takes into account the forces applied by viscous forces, described by the term $\mu \nabla^2 \mathbf{V}$, whose derivation can be found in Professor Alexander Smits's Viscous Flows and Turbulence Lectures Notes in Fluid Mechanics pages 22 and 23 [1]. By implementing, this term in Eulers Equation (Eq A2.6), the final form of the Navier-Stokes Momentum Equation can be obtained :

$$\rho \frac{D\mathbf{V}}{dt} = -\nabla \rho + \rho \mathbf{g} + \mu \nabla^2 \mathbf{V} \tag{A2.7}$$

$$\nabla \cdot v = 0 \tag{A2.8}$$

[1] Product Rule: $\nabla \cdot (\phi \mathbf{A}) = \phi (\nabla \cdot \mathbf{A}) + \mathbf{A} \cdot (\nabla \phi)$

8.3 Appendix A3: RANS Equations Derivation

This demonstration of Reynold's Average Navier Stokes Equations for compressible fluids has been obtained from Professor Alexander Smits's Viscous Flows and Turbulence Lectures Notes in Fluid Mechanics Chapter 7 pages 211 to 215 [1].

The derivation of the final set of RANS equations begins with the previously derived Navier-Stokes equations which are the continuity equation (Equation (A2.8)) and the momentum equation (Equation (**??**)) for incompressible flows.

$$\nabla \mathbf{V} = 0; \tag{A3.1}$$

$$\frac{D\mathbf{V}}{Dt} = -\frac{1}{\rho}\nabla p + \mathbf{g} + \mu\nabla^2\mathbf{V} \tag{A3.2}$$

As was mentioned in the Report, the RANS set of equations was obtained by applying the Reynolds decomposition, displayed in Equation (6) where the instantaneous velocity (u) was defined by the addition of the averaged velocity ($\bar{U}$) and the fluctuating term (u').

$$u = \bar{U} + u' \tag{A3.2}$$

Additionally, the mathematical rules that are applied to time-averaged terms and will be used in this explanation are:

$$\overline{a'} = 0 \tag{Rule 1}$$

$$\overline{a + b} = \overline{a} + \overline{b} \tag{Rule 2}$$

$$\overline{\overline{a}} = \overline{a} \tag{Rule 3}$$

$$\overline{\overline{a} \cdot b} = \overline{a} \cdot \overline{b} \tag{Rule 4}$$

$$\bar{a'} = 0 \tag{Rule 5}$$

$$\bar{a'} = 0 \tag{Rule 1}$$

Firstly, we will start by deriving the RANS continuity equation by expanding Equation (A3.1), by using the definition of the divergence theorem.

$$\nabla \mathbf{V} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0; \tag{A3.4}$$

Then, by applying the Reynolds decomposition to all components of the velocity, we can obtain the following expression:

$$\nabla \mathbf{V} = \frac{\partial \overline{U}}{\partial x} + \frac{\partial \overline{V}}{\partial y} + \frac{\partial \overline{W}}{\partial z} + \frac{\partial u'}{\partial x} + \frac{\partial v'}{\partial y} + \frac{\partial w'}{\partial z} = 0; \tag{A3.4}$$

This expression can be divided into 2 terms, the time average velocity terms and the fluctuating velocity terms.

If we take the average velocity terms $\overline{U}$, $\overline{V}$, and $\overline{W}$, the following equation represents the continuity equation for fluid flow, signifying that in steady-state conditions, the divergence of the average velocity field is zero. Which can be simplified into the equation (A3.6):

$$\frac{\partial \overline{U}}{\partial x} + \frac{\partial \overline{V}}{\partial y} + \frac{\partial \overline{W}}{\partial z} = 0; \tag{A3.5}$$

$$\nabla \overline{\mathbf{V}} = 0; \tag{A3.6}$$

Finally, by substituting the divergence of the average flow by 0, into equation (A3.4), we can derive that the divergence of the fluctuation term of the velocity is also 0, and after simplifying we obtain equation (A3.8).

$$0 + \frac{\partial u'}{\partial x} + \frac{\partial v'}{\partial y} + \frac{\partial w'}{\partial z} = 0 \tag{A3.7}$$

$$\nabla \mathbf{v}' = 0; \tag{A3.8}$$

To derive the RANS momentum equation, we will particularise the Momentum equation (A3.2) to the X momentum equations. Additionally, we will apply the divergence definition to the velocity and pressure terms and the Reynolds decomposition will be applied to those same terms, therefore expanding the equation.

$$\begin{aligned}&\frac{\partial \overline{U}}{\partial t} + \frac{\partial u'}{\partial t} + \overline{U}\frac{\partial \overline{U}}{\partial x} + u'\frac{\partial u'}{\partial x} + u'\frac{\partial \overline{U}}{\partial x} + \overline{U}\frac{\partial u'}{\partial x} + \overline{V}\frac{\partial \overline{U}}{\partial y} + v'\frac{\partial u'}{\partial y} + v'\frac{\partial \overline{U}}{\partial y} + \overline{W}\frac{\partial \overline{U}}{\partial z} + w'\frac{\partial u'}{\partial z} + w'\frac{\partial \overline{U}}{\partial z} \\ &= -\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x} - \frac{1}{\rho}\frac{\partial p'}{\partial x} + \nu\nabla^2\overline{U} + \nu\nabla^2 u'\end{aligned} \tag{A3.9}$$

This equation can be simplified by using the rules defined at the start of the section.

- By definition, averaged terms do not vary in time: $\partial \overline{U}/\partial t = 0$
- By applying (Rule 1) the time derivative of a fluctuating term is 0: $\partial u'/\partial t = 0$

The next step consists on time-averaging all the terms left of the equations, which is represented with the overbar.

$$\begin{aligned}&\overline{\overline{U}\frac{\partial \overline{U}}{\partial x} + u'\frac{\partial u'}{\partial x} + u'\frac{\partial \overline{U}}{\partial x} + \overline{U}\frac{\partial u'}{\partial x} + \overline{V}\frac{\partial \overline{U}}{\partial y} + v'\frac{\partial u'}{\partial y} + v'\frac{\partial \overline{U}}{\partial y} + \overline{W}\frac{\partial \overline{U}}{\partial z} + w'\frac{\partial u'}{\partial z} + w'\frac{\partial \overline{U}}{\partial z}} \\ &= \overline{-\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x} - \frac{1}{\rho}\frac{\partial p'}{\partial x} + \nu\nabla^2\overline{U} + \nu\nabla^2 u'}\end{aligned} \tag{A3.9}$$

By applying (Rule 2), the time average of the sum of the components is equal to the sum of the individual components time-averaged.

$$\begin{aligned}&\overline{\overline{U}\frac{\partial \overline{U}}{\partial x}} + \overline{u'\frac{\partial u'}{\partial x}} + \overline{u'\frac{\partial \overline{U}}{\partial x}} + \overline{\overline{U}\frac{\partial u'}{\partial x}} + \overline{\overline{U}\frac{\partial \overline{U}}{\partial x}} + \overline{\overline{V}\frac{\partial \overline{U}}{\partial y}} + \overline{v'\frac{\partial u'}{\partial y}} + \overline{\frac{\partial \overline{U}}{\partial y}} + \overline{\overline{W}\frac{\partial \overline{U}}{\partial z}} + \overline{w'\frac{\partial u'}{\partial z}} + \overline{w'\frac{\partial \overline{U}}{\partial z}} \\ &= \overline{-\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x}} - \overline{-\frac{1}{\rho}\frac{\partial p'}{\partial x}} + \overline{\nu\nabla^2\overline{U}} + \overline{\nu\nabla^2 u'}\end{aligned} \tag{A3.9}$$

By crossing all zero terms in equation (A3.9), the following expression is obtained.

$$\overline{\overline{U}\frac{\partial \overline{U}}{\partial x}} + \overline{\overline{U}\frac{\partial u'}{\partial x}} + \overline{\overline{U}\frac{\partial \overline{U}}{\partial x}} + \overline{\frac{\partial \overline{U}}{\partial y}} + \overline{\overline{W}\frac{\partial \overline{U}}{\partial z}} + \overline{w'\frac{\partial u'}{\partial z}} + \overline{w'\frac{\partial \overline{U}}{\partial z}}$$
$$= \overline{-\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x}} - \overline{-\frac{1}{\rho}\frac{\partial p'}{\partial x}} + \overline{\nu\nabla^2\overline{U}} + \overline{\nu\nabla^2 u'} \tag{A3.9}$$

9 Appendix B: Utilities Function

9.1 Appendix B1: Interpolation Function Test

```
from src.utils.DataUtils import DataUtils
import pytest


@pytest.mark.parametrize("function", [fsquare, fCubic, fQuartic, fQuintic, fSextic])
def test_interpolateSquare(function):
    resolution = 10000
    for num in [10000,5000,4000,3000,2000]:
        x = np.linspace(-8.65,10.548,num)
        y = np.array([])

        for k in range(len(x)):
            y = np.append(y,function(x[k]))

        x_new,y_new = DataUtils.interpolate(x,y,resolution)
        print(num)
        for k in range(len(x_new)):
            assert abs(function(x_new[k]) - y_new[k] ) < 10e-4
```

9.2 Appendix B2: Mirroring Function

```
def mirroring(x : np.array, y : np.array, ymirroring : bool = False) -> tuple[np.array, np.array]:
        lasty = y[-1]
        #flip the originaly array
        ymirror = np.array(np.flip(y))
        #remove the first element to not duplicate last y of the original array
        ymirror = np.delete(ymirror, 0)

        if(ymirroring):
            ymirror = lasty - ymirror + lasty

        y = np.append(y, ymirror)

        # process the mirroring of x array
        if len(x)!=len(y):
            xmirror = np.array([])
            lastx = x[-1]
            for i in range(1,len(x)):
                # x mirror is built by adding the differences between
                # consecutive elements of the x array to the last x element
                xmirror = np.append(xmirror,(lastx + np.abs(lastx-x[len(x) - i - 1])))

            x = np.append(x, xmirror)

        return (x,y)
```

10 Appendix C: Machine Learning

10.1 Appendix C1: Simple Neural Network Reynolds Stresses Training Plots

10.1.1 Appendix C1.1: 10 Epochs

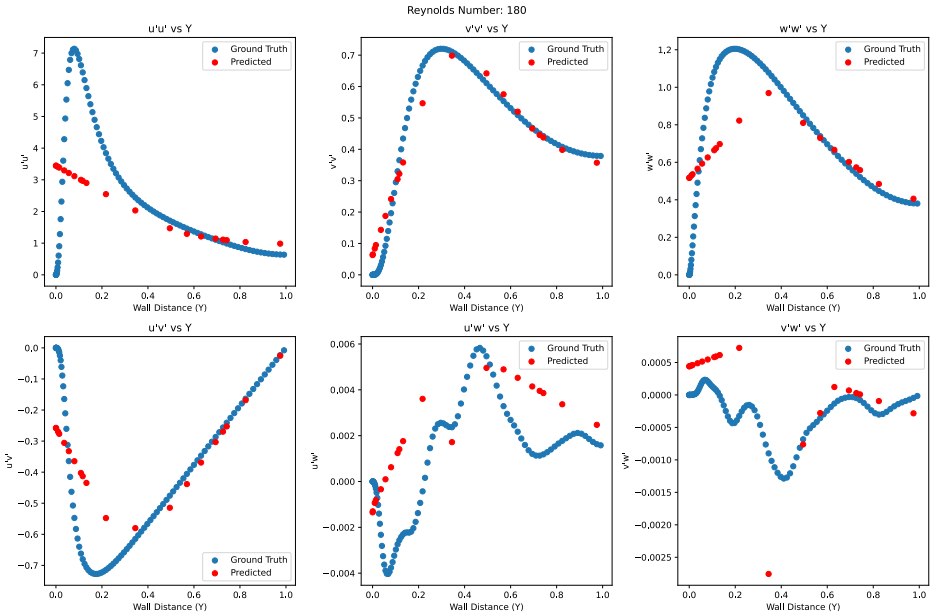

Figure 78: Simple Neural Network Training for Channel Flow - Re 180 - Epoch Number 10

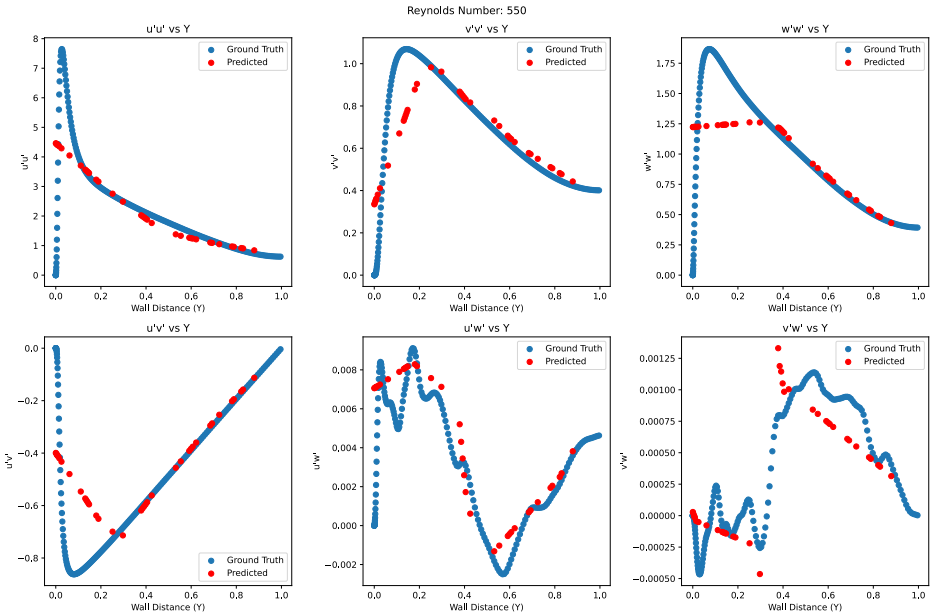

Figure 79: Simple Neural Network Training for Channel Flow - Re 550 - Epoch Number 10

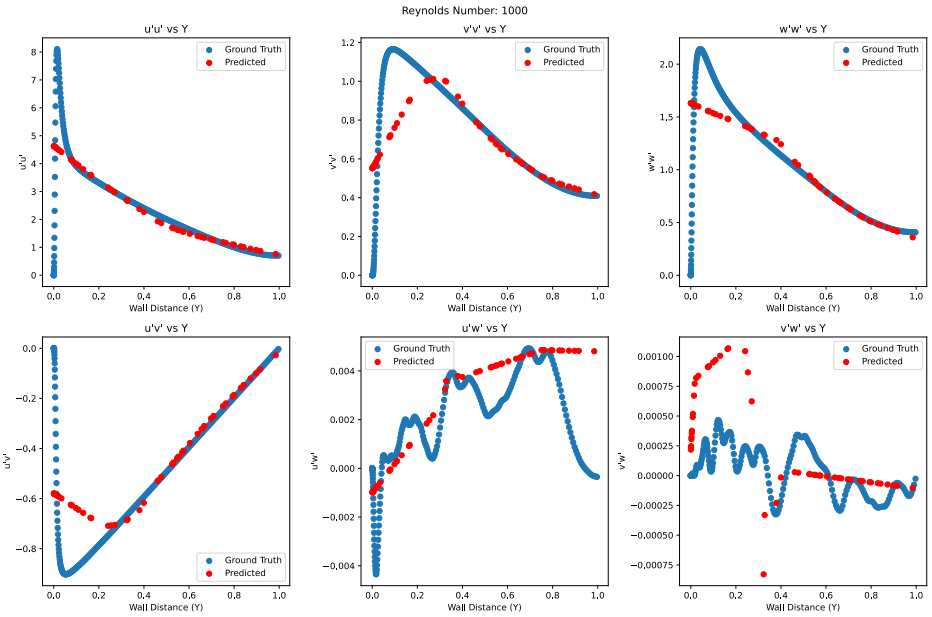

Figure 80: Simple Neural Network Training for Channel Flow - Re 1000 - Epoch Number 10

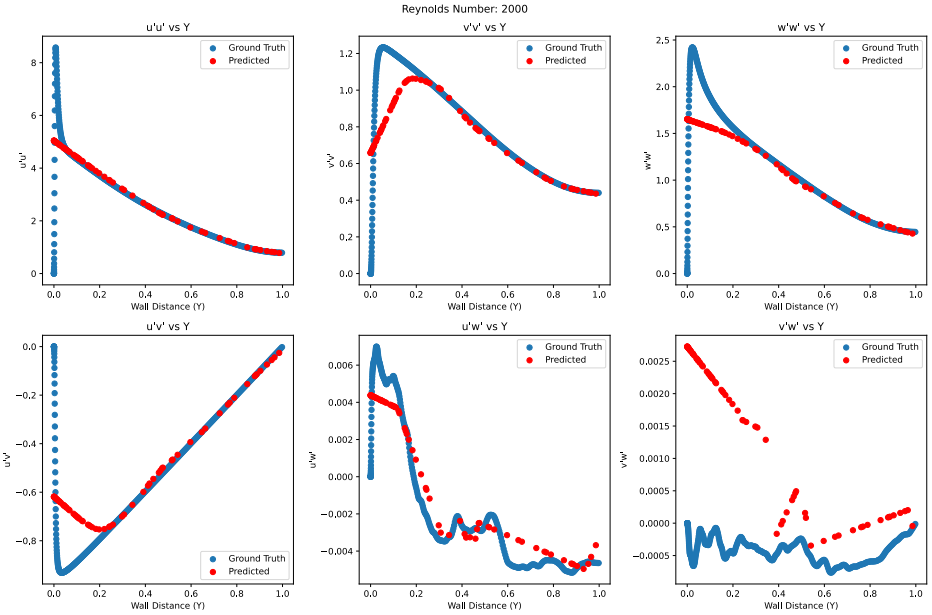

Figure 81: Simple Neural Network Training for Channel Flow - Re 2000 - Epoch Number 10

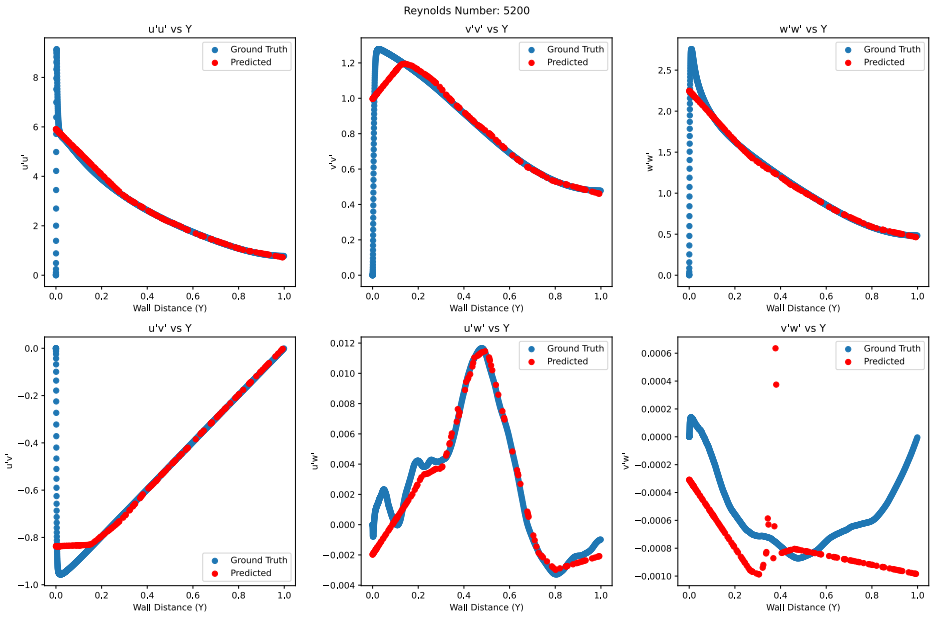

Figure 82: Simple Neural Network Training for Channel Flow - Re 5200 - Epoch Number 10

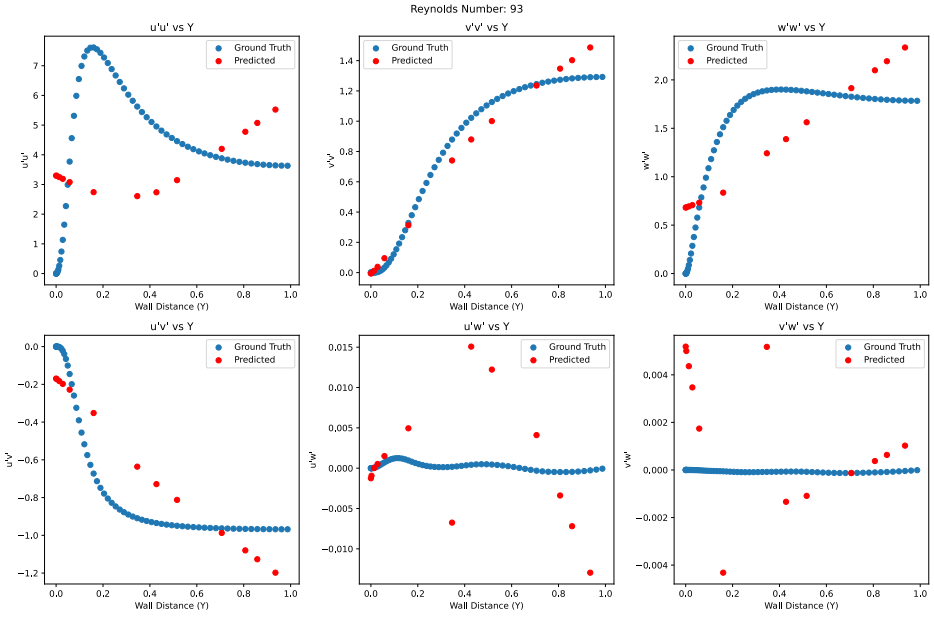

Figure 83: Simple Neural Network Training for Couette Flow - Re 93 - Data Points 100 - Epoch Number 10

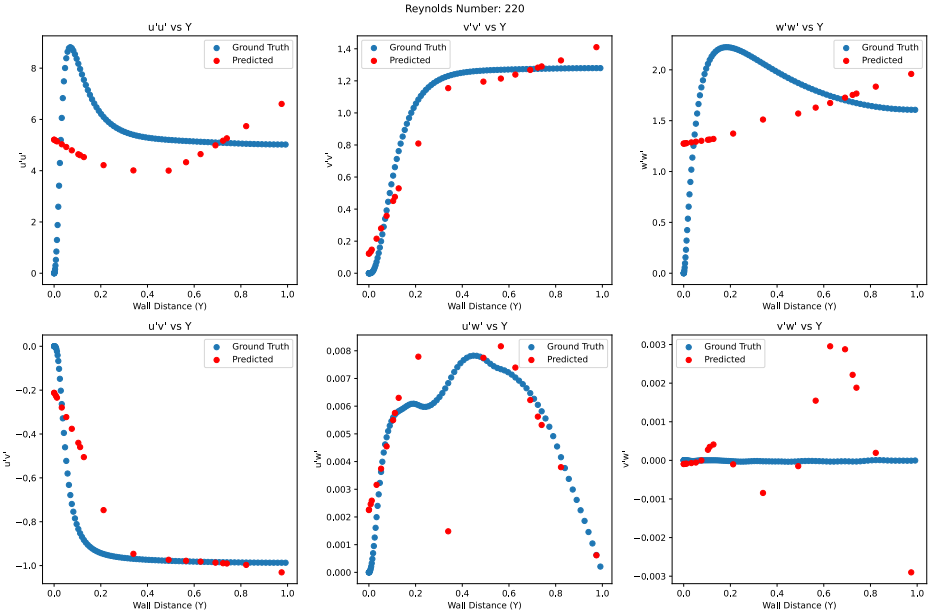

Figure 84: Simple Neural Network Training for Couette Flow - Re 220 - Data Points 20 - Epoch Number 10

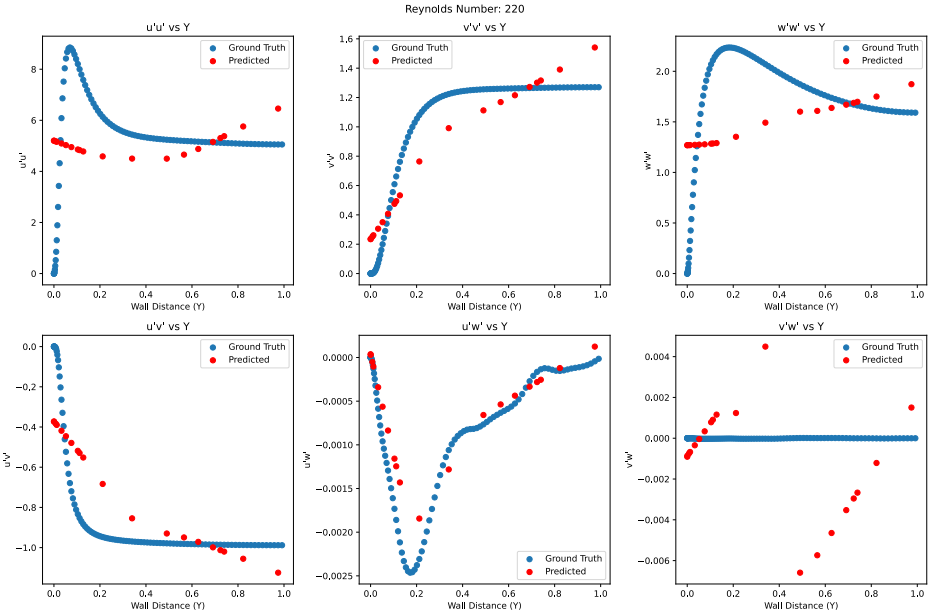

Figure 85: Simple Neural Network Training for Couette Flow - Re 220 - Data Points 100 - Epoch Number 10

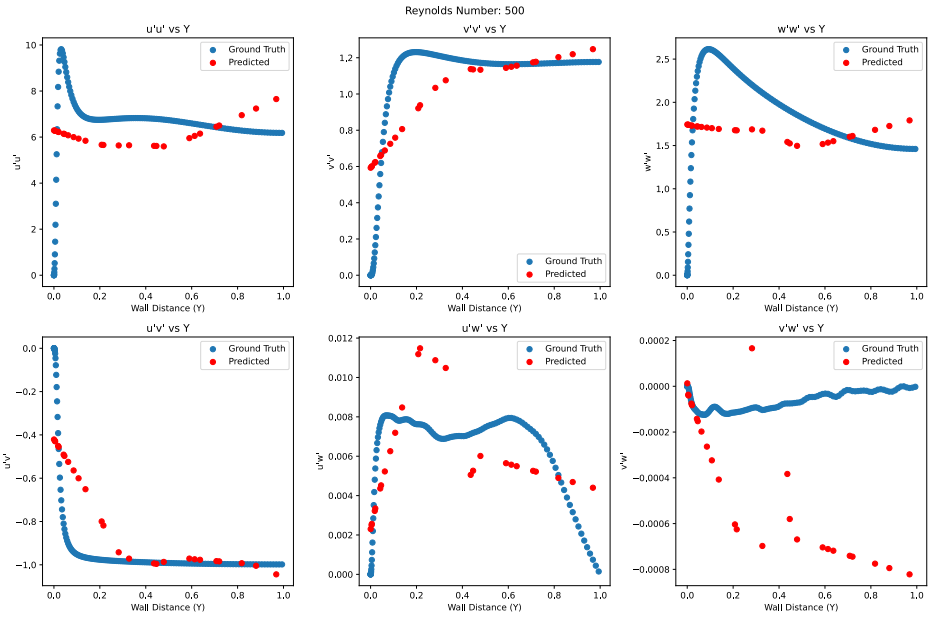

Figure 86: Simple Neural Network Training for Couette Flow - Re 500 - Data Points 20 - Epoch Number 10

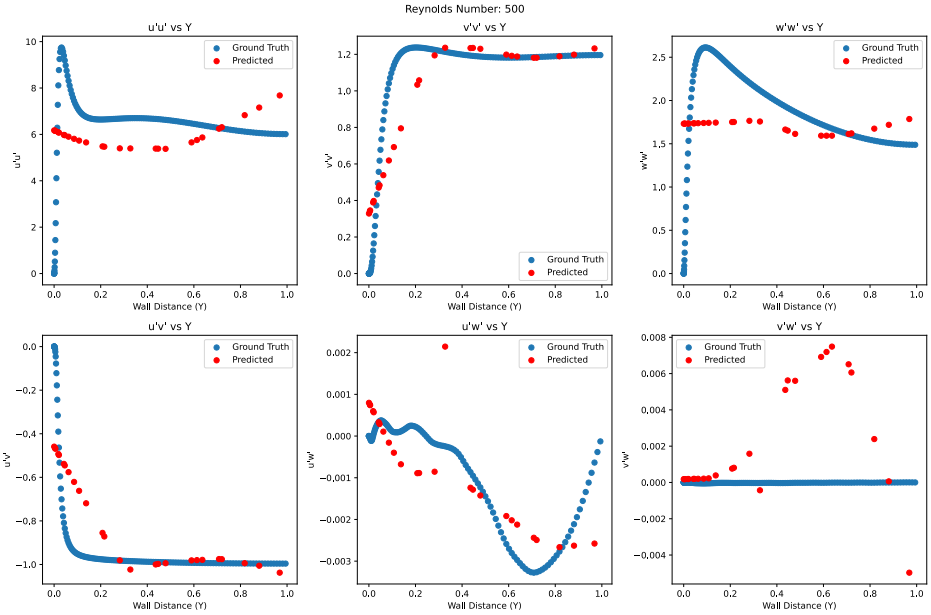

Figure 87: Simple Neural Network Training for Couette Flow - Re 500 - Data Points 100 - Epoch Number 10

10.1.2 Appendix C1.1: 50 Epochs

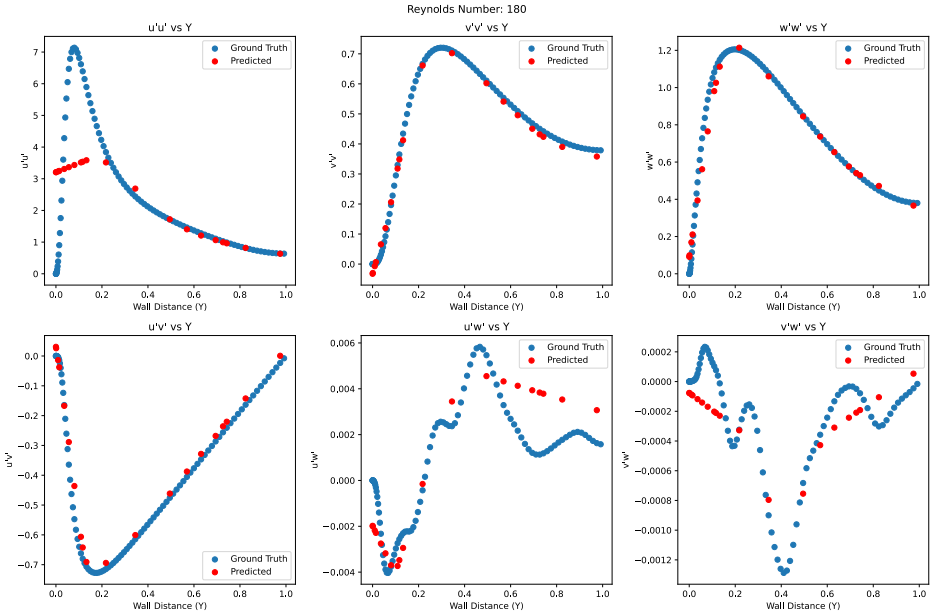

Figure 88: Simple Neural Network Training for Channel Flow - Re 180 - Epoch Number 50

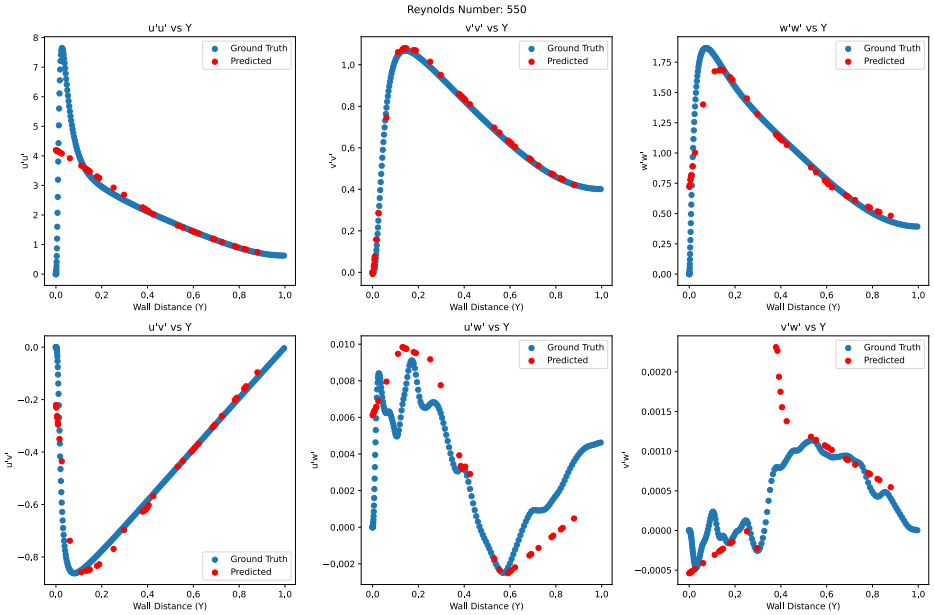

Figure 89: Simple Neural Network Training for Channel Flow - Re 550 - Epoch Number 50

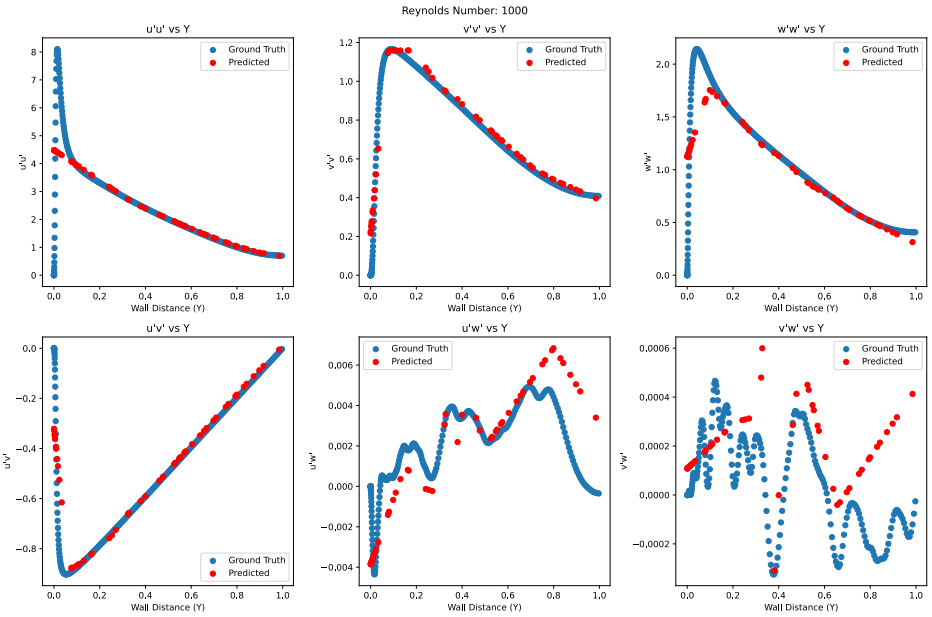

Figure 90: Simple Neural Network Training for Channel Flow - Re 1000 - Epoch Number 50

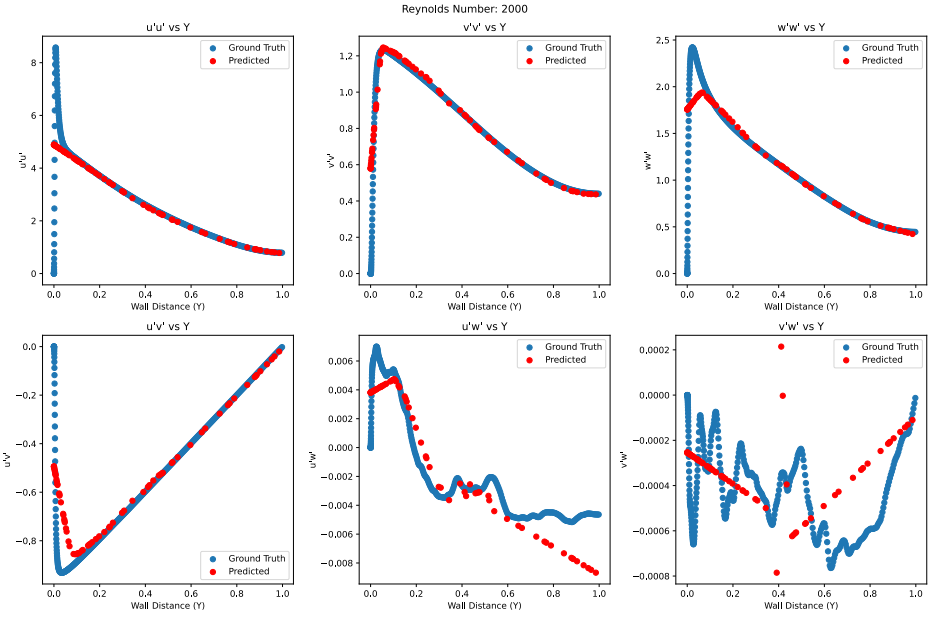

Figure 91: Simple Neural Network Training for Channel Flow - Re 2000 - Epoch Number 50

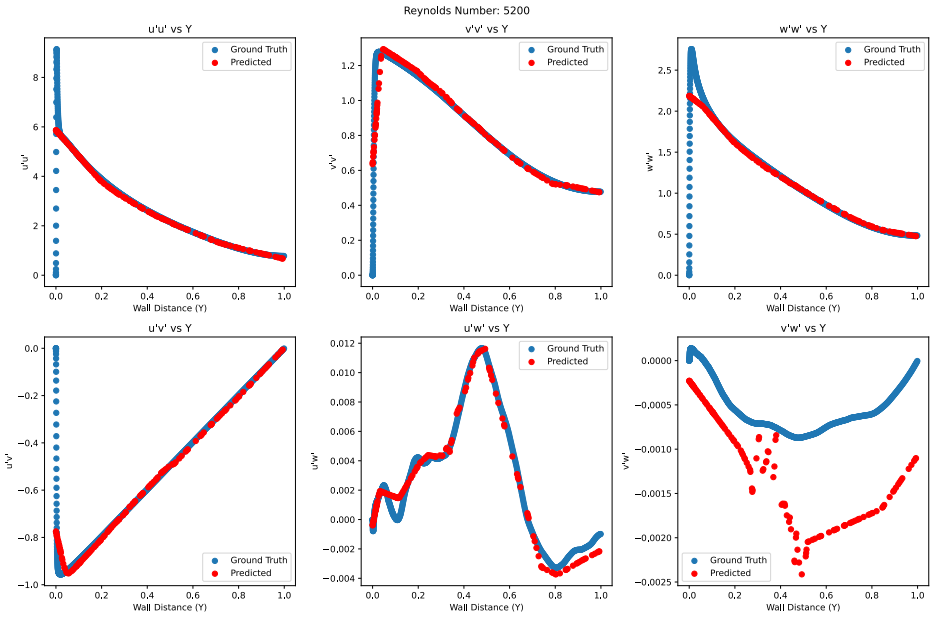

Figure 92: Simple Neural Network Training for Channel Flow - Re 5200 - Epoch Number 50

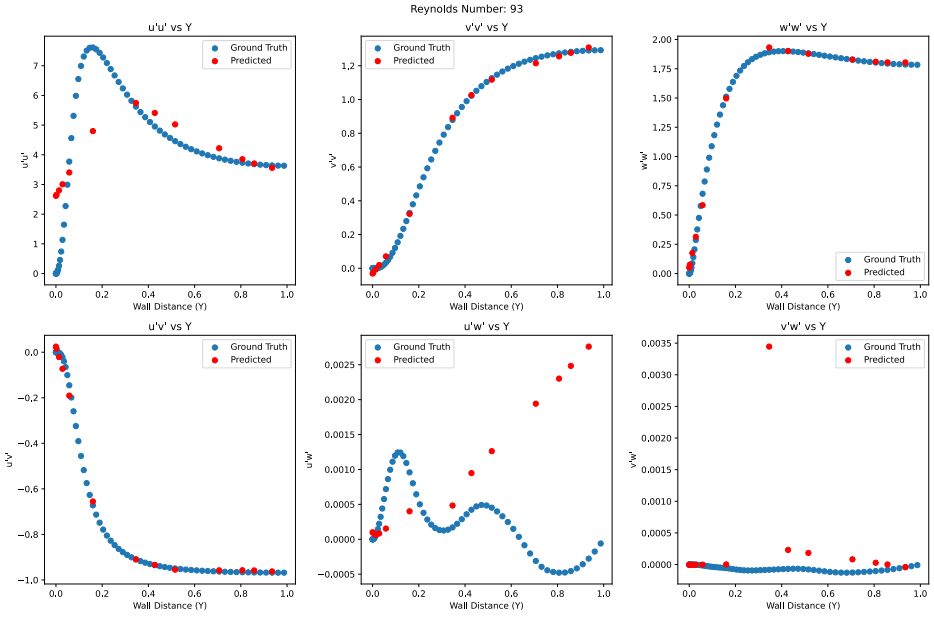

Figure 93: Simple Neural Network Training for Couette Flow - Re 93 - Data Points 100 - Epoch Number 50

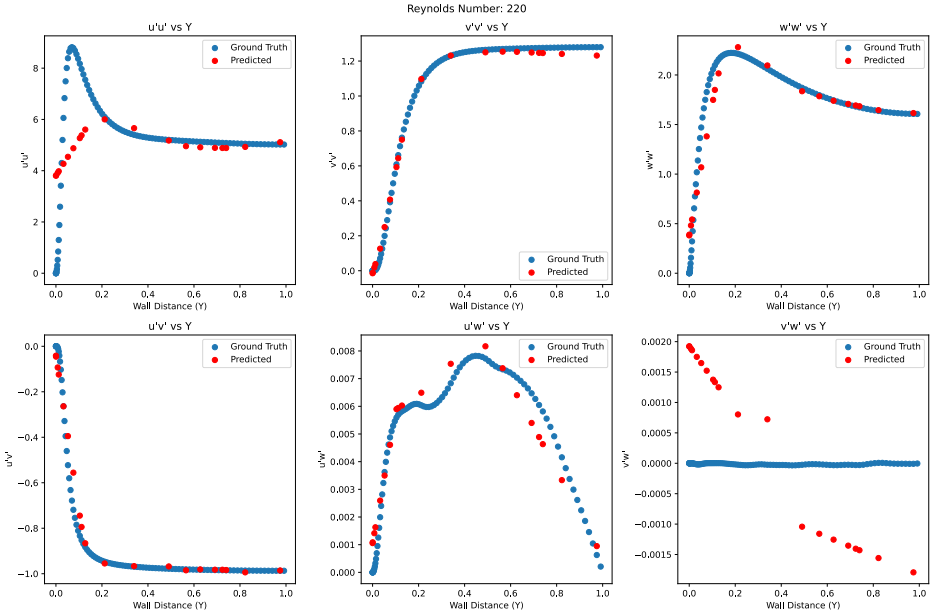

Figure 94: Simple Neural Network Training for Couette Flow - Re 220 - Data Points 20 - Epoch Number 50

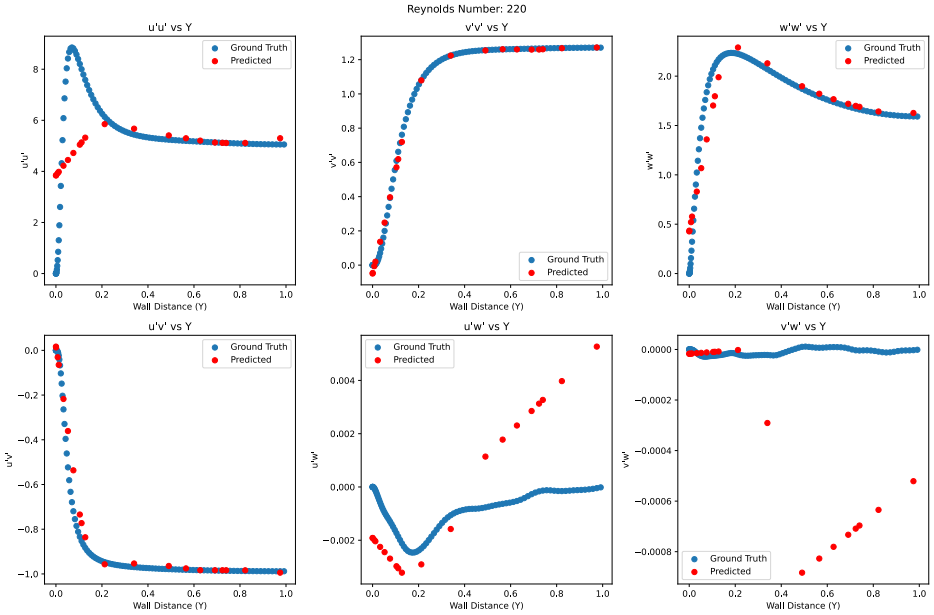

Figure 95: Simple Neural Network Training for Couette Flow - Re 220 - Data Points 100 - Epoch Number 50

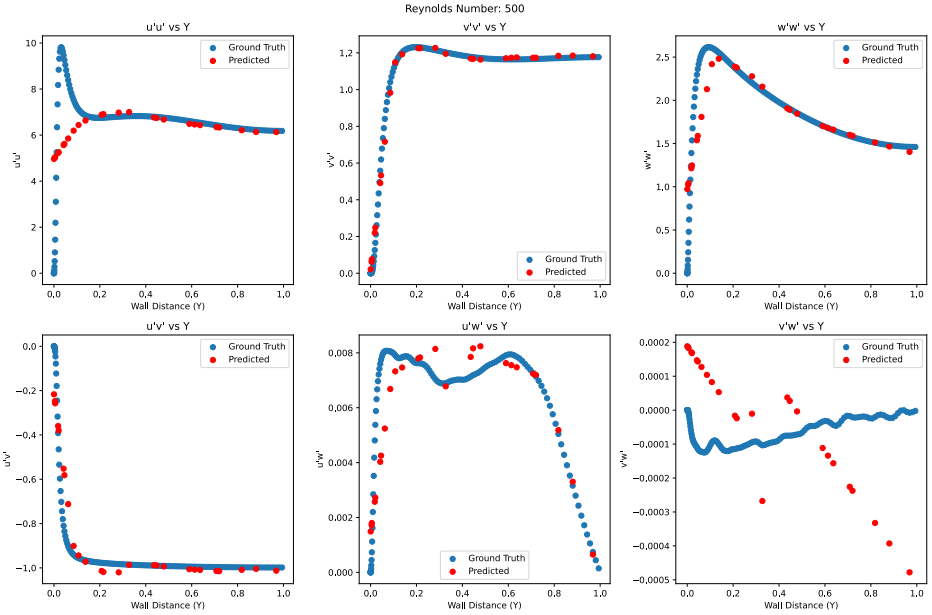

Figure 96: Simple Neural Network Training for Couette Flow - Re 500 - Data Points 20 - Epoch Number 50

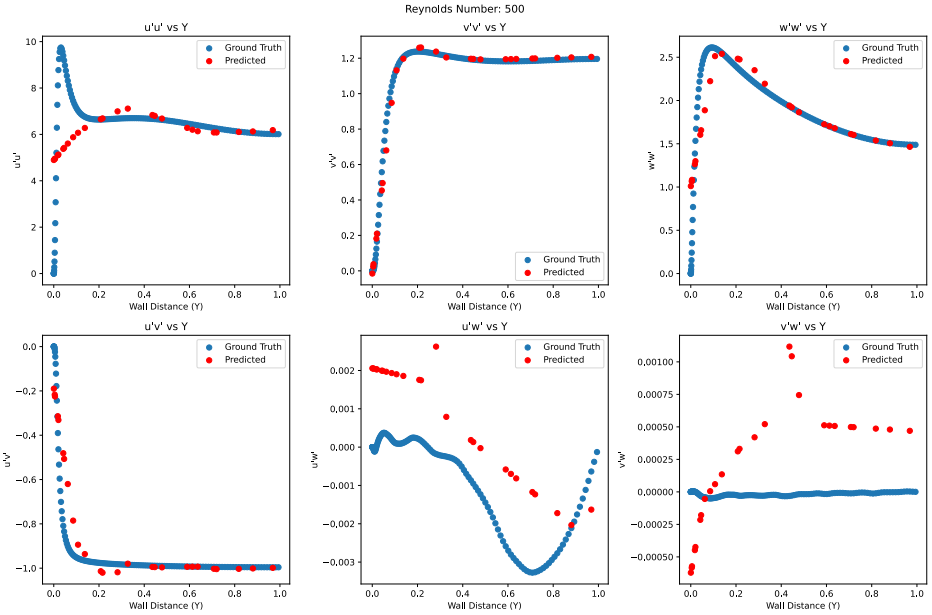

Figure 97: Simple Neural Network Training for Couette Flow - Re 500 - Data Points 100 - Epoch Number 50

10.1.3 Appendix C1.1: 100 Epochs

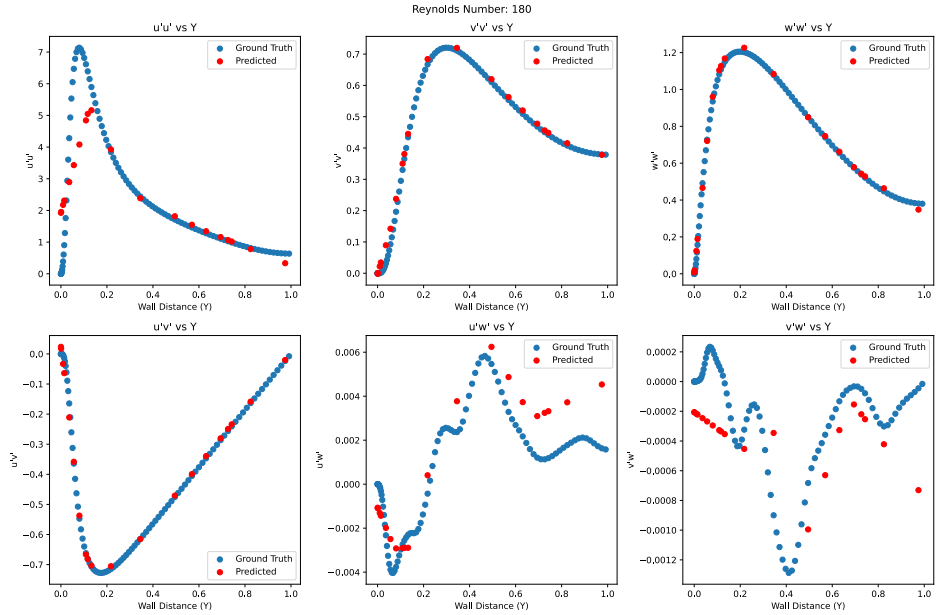

Figure 98: Simple Neural Network Training for Channel Flow - Re 180 - Epoch Number 100

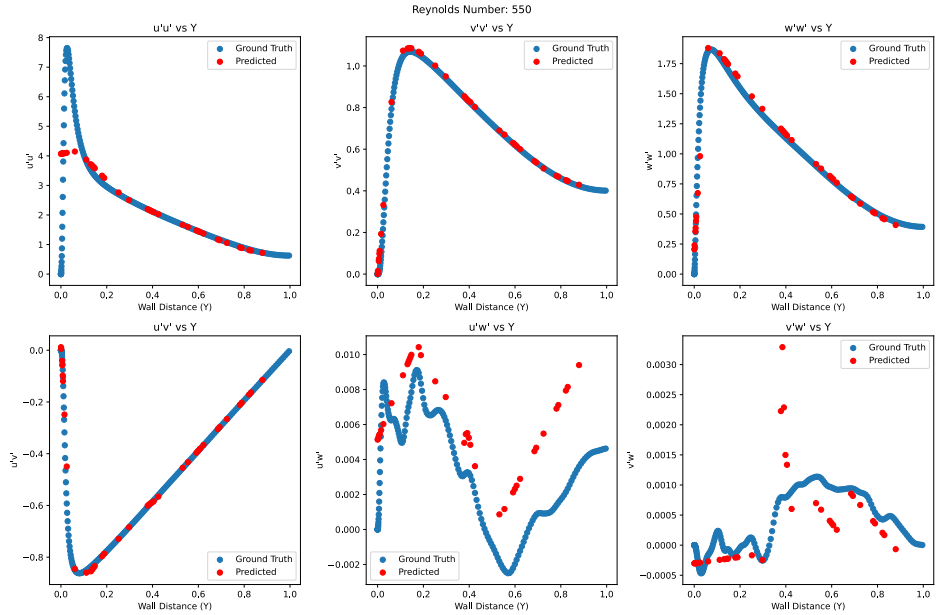

Figure 99: Simple Neural Network Training for Channel Flow - Re 550 - Epoch Number 100

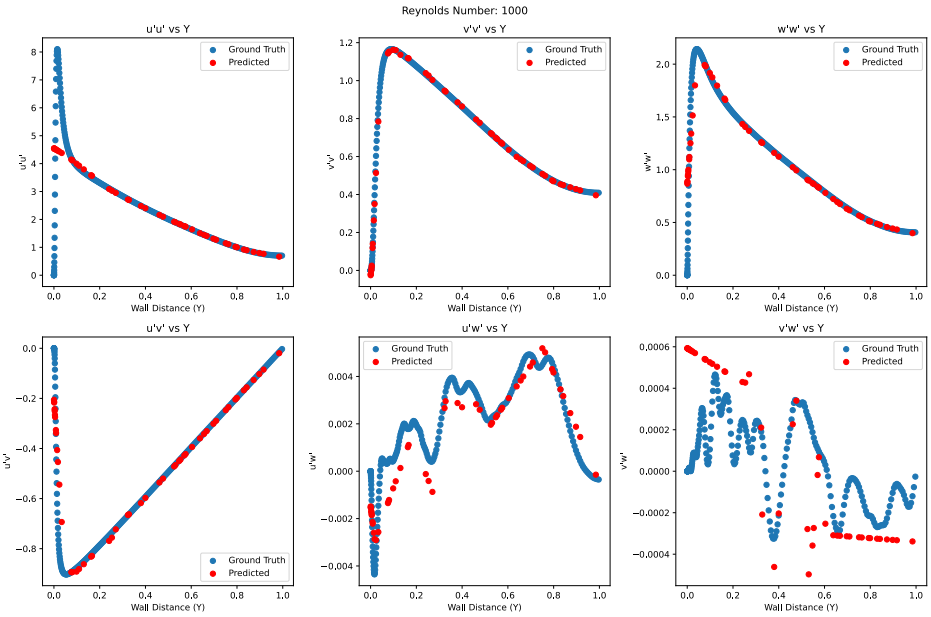

Figure 100: Simple Neural Network Training for Channel Flow - Re 1000 - Epoch Number 100

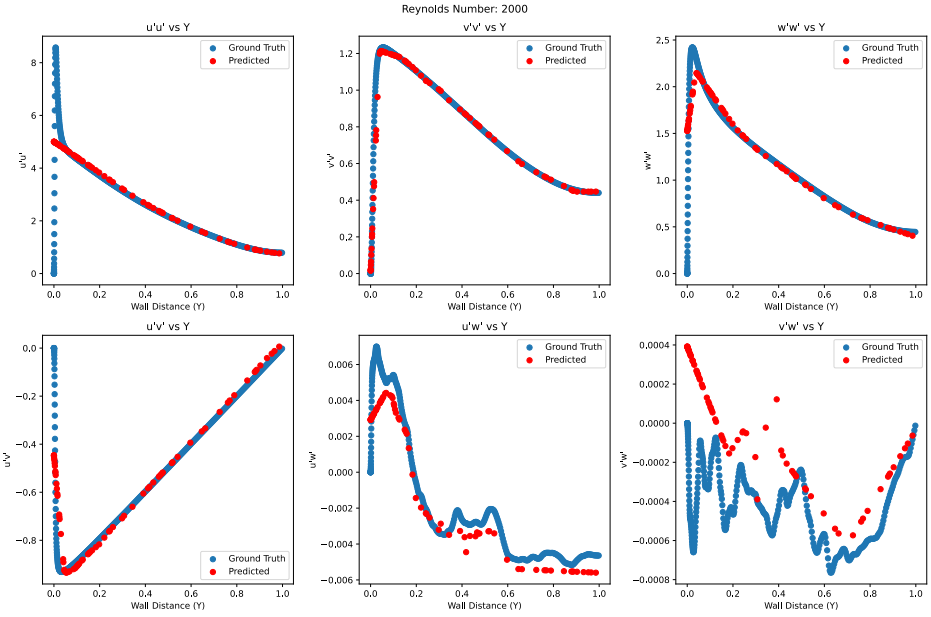

Figure 101: Simple Neural Network Training for Channel Flow - Re 2000 - Epoch Number 100

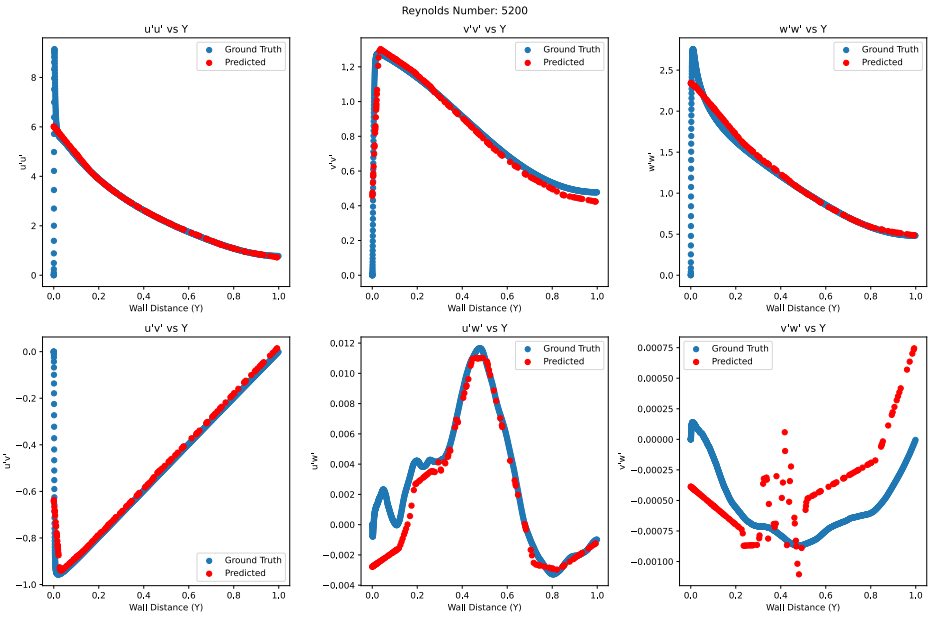

Figure 102: Simple Neural Network Training for Channel Flow - Re 5200 - Epoch Number 100

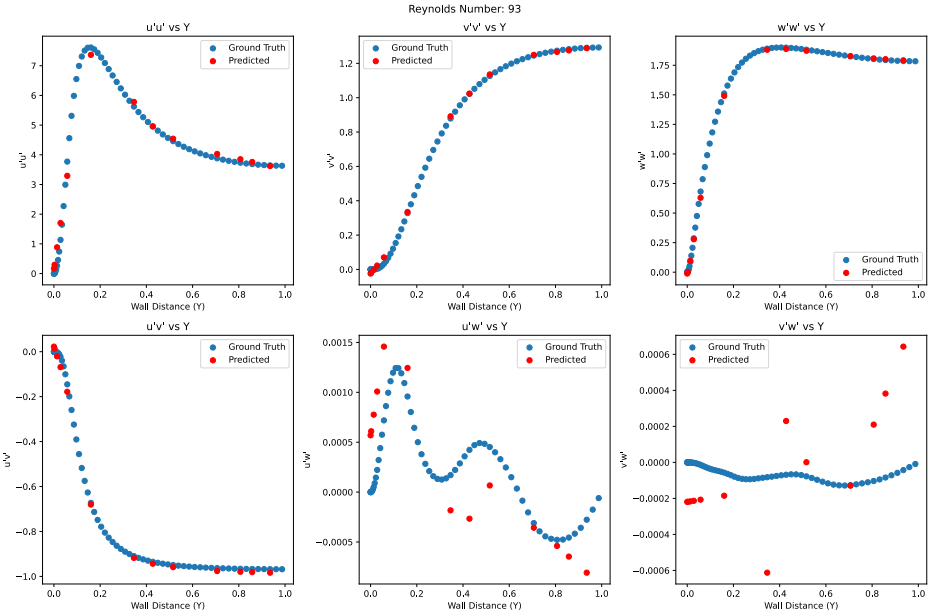

Figure 103: Simple Neural Network Training for Couette Flow - Re 93 - Data Points 100 - Epoch Number 100

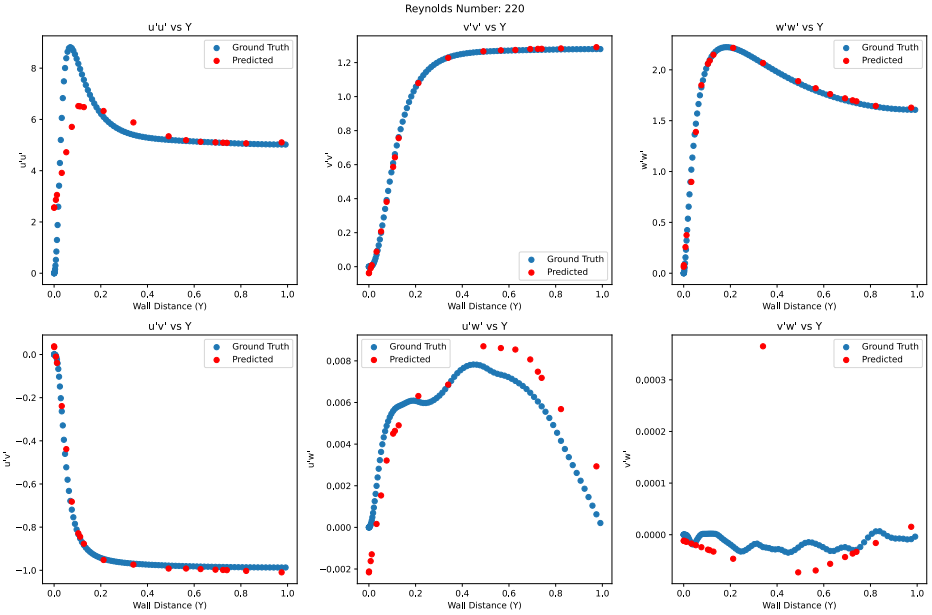

Figure 104: Simple Neural Network Training for Couette Flow - Re 220 - Data Points 20 - Epoch Number 100

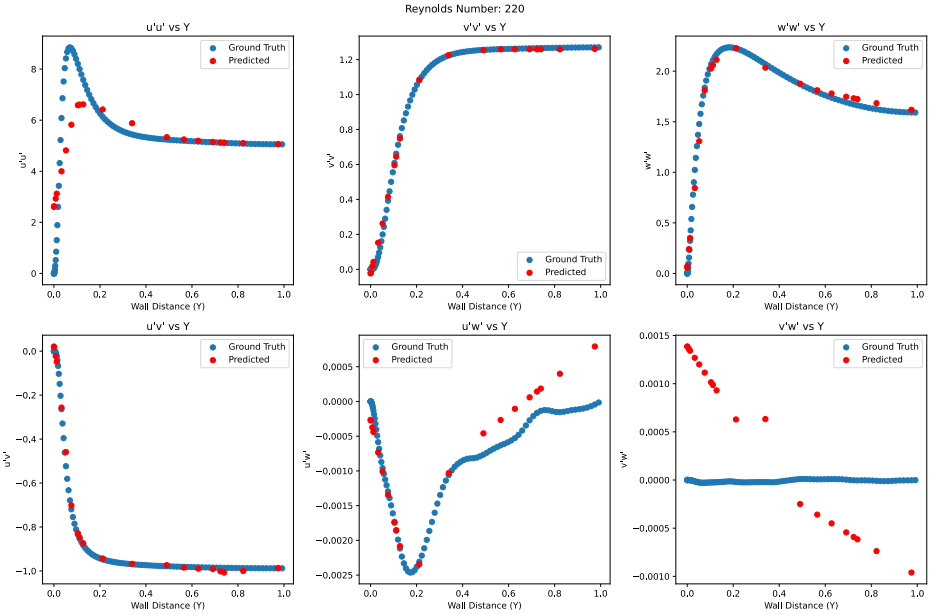

Figure 105: Simple Neural Network Training for Couette Flow - Re 220 - Data Points 100 - Epoch Number 100

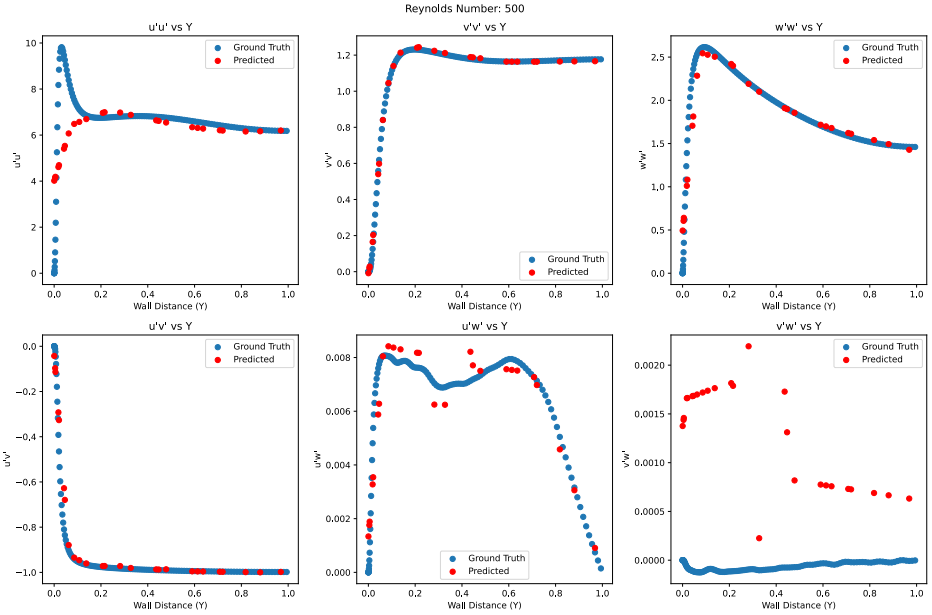

Figure 106: Simple Neural Network Training for Couette Flow - Re 500 - Data Points 20 - Epoch Number 100

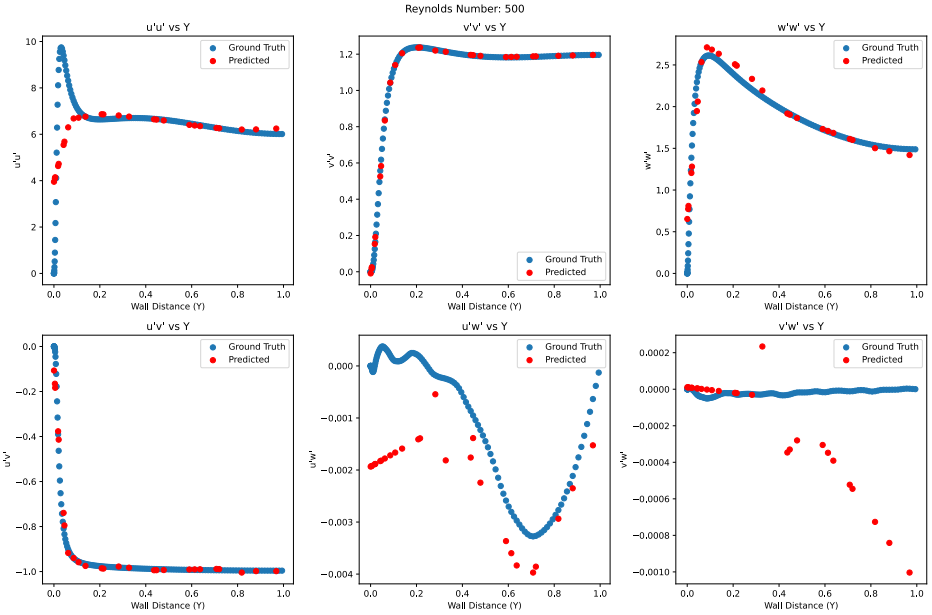

Figure 107: Simple Neural Network Training for Couette Flow - Re 500 - Data Points 100 - Epoch Number 100

10.2 Appendix C2: K Optimisation Experimental Plots

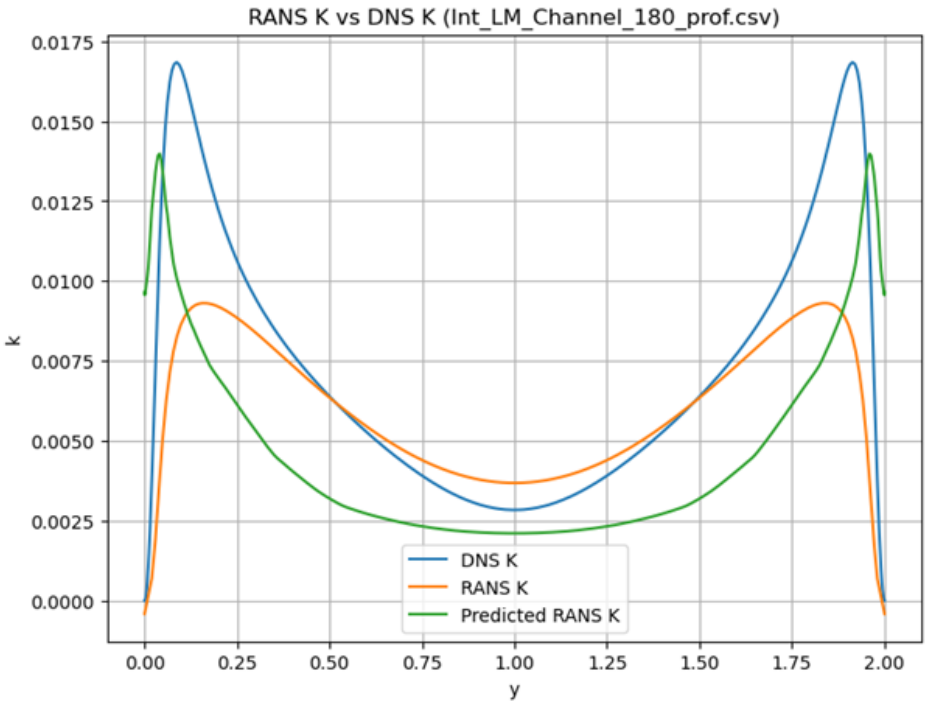

Figure 108: K Optimisation for Re 180 Channel Flow

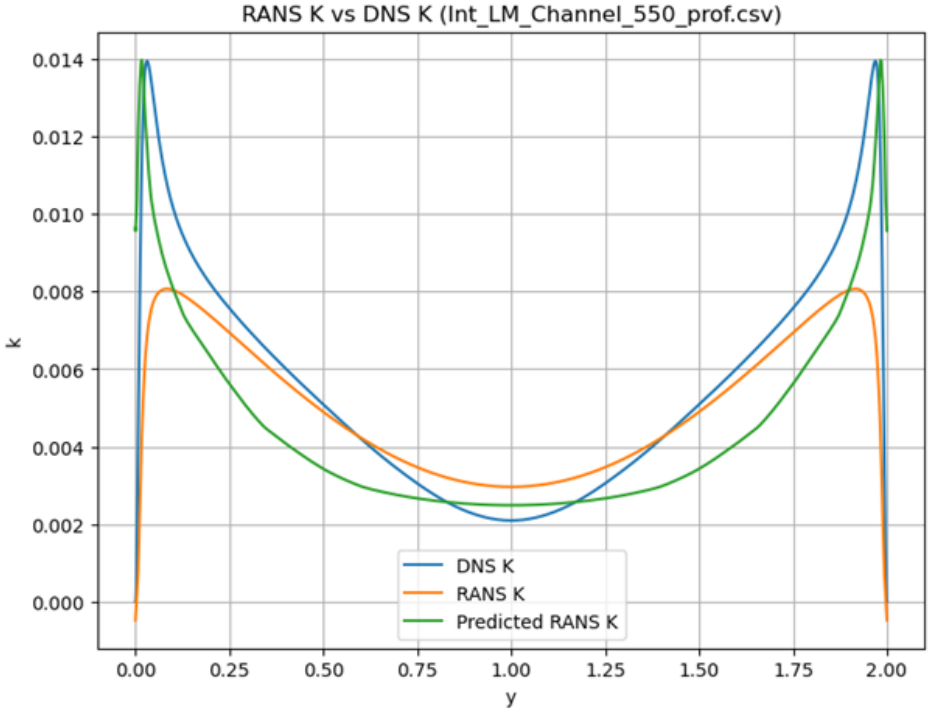

Figure 109: K Optimisation for Re 550 Channel Flow

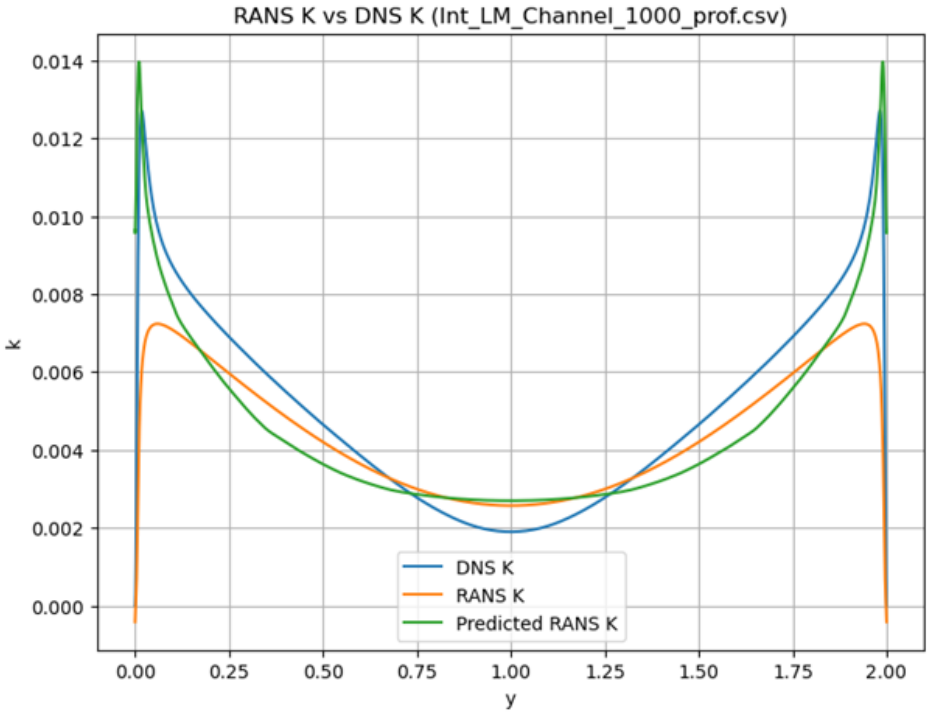

Figure 110: K Optimisation for Re 1000 Channel Flow

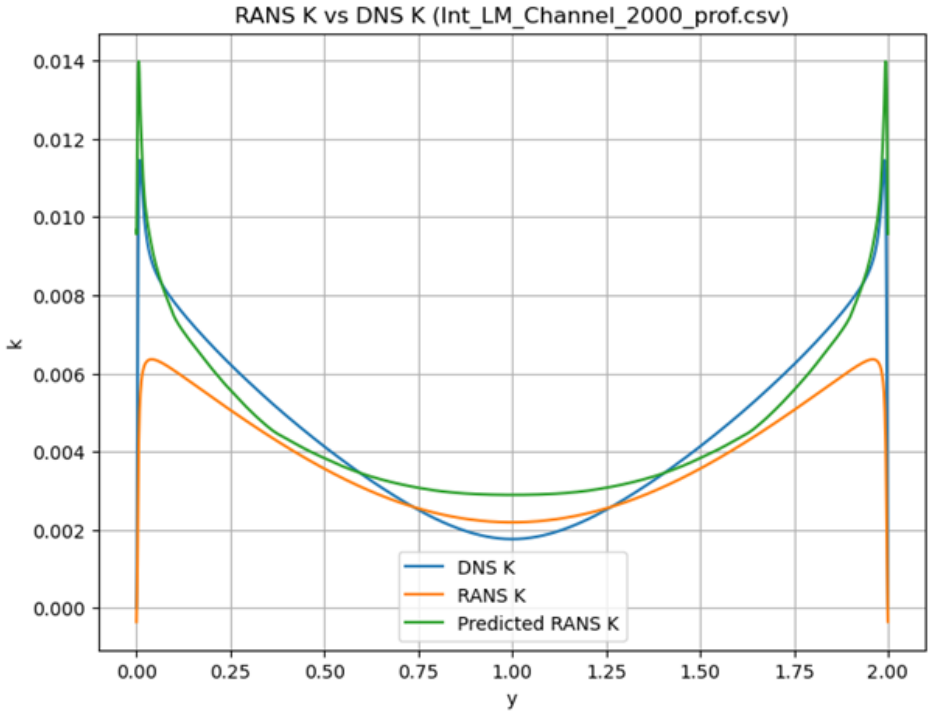

Figure 111: K Optimisation for Re 2000 Channel Flow

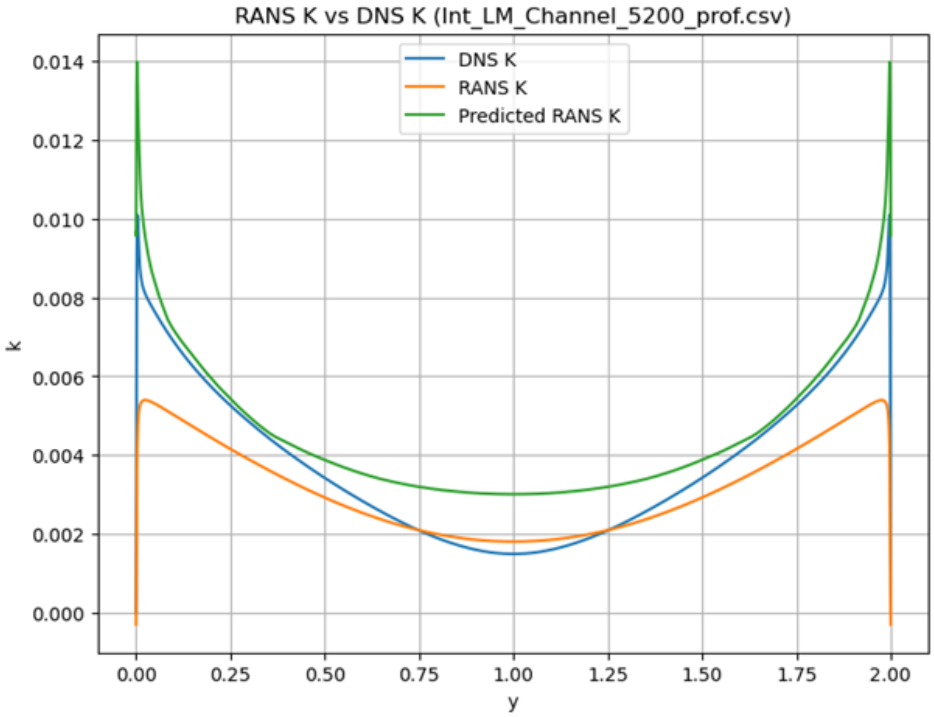

Figure 112: K Optimisation for Re 5200 Channel Flow

10.3 Appendix C3: PINN Prediction Plots

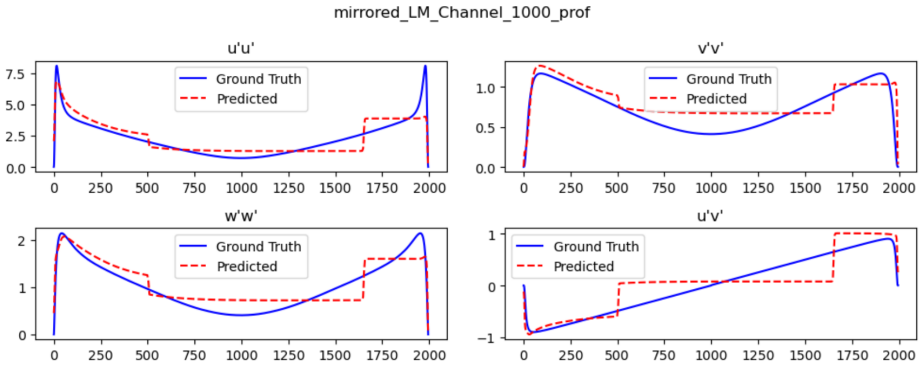

Figure 113: Unified RANS Based PINN Channel 10 Neurons & 2 Hidden Layers

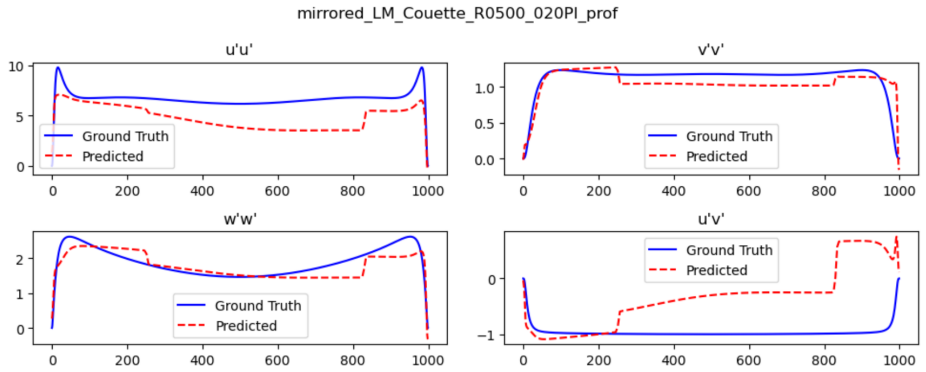

Figure 114: Unified RANS Based PINN Couette 10 Neurons & 2 Hidden Layers

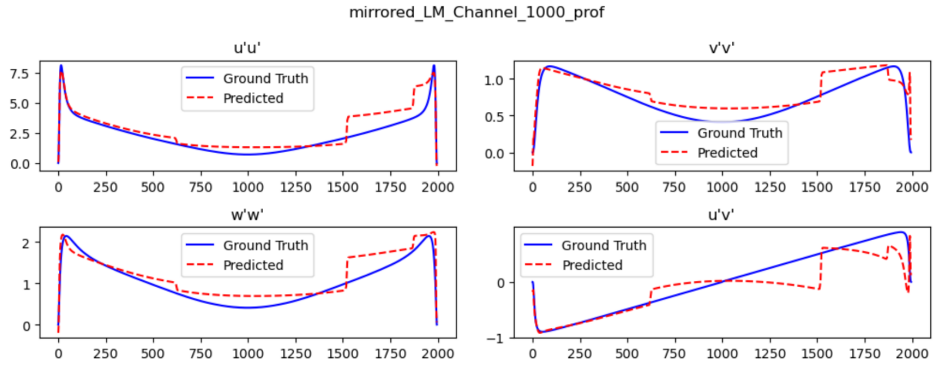

Figure 115: Unified RANS Based PINN Channel 10 Neurons & 8 Hidden Layers

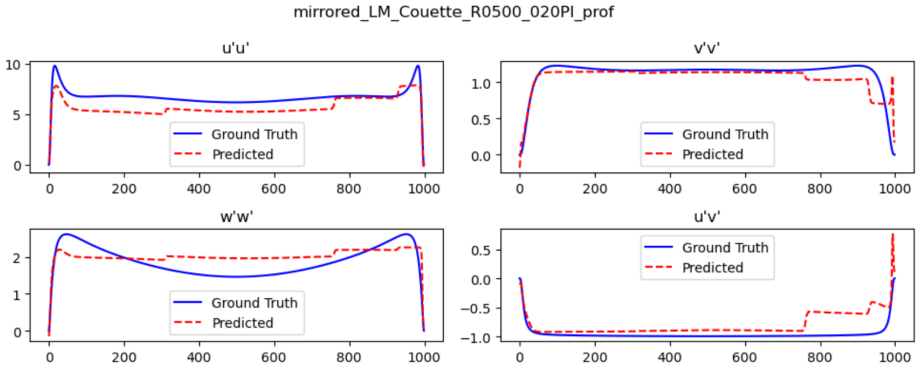

Figure 116: Unified RANS Based PINN Couette 10 Neurons & 8 Hidden Layers

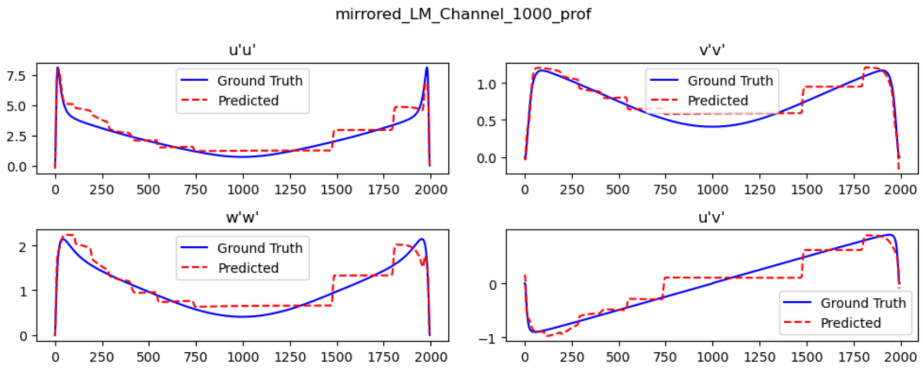

Figure 117: Unified RANS Based PINN Channel 20 Neurons & 2 Hidden Layers

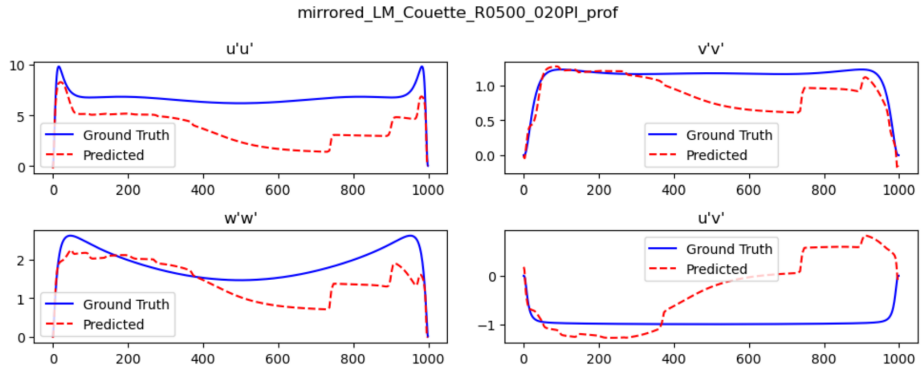

Figure 118: Unified RANS Based PINN Couette 20 Neurons & 2 Hidden Layers

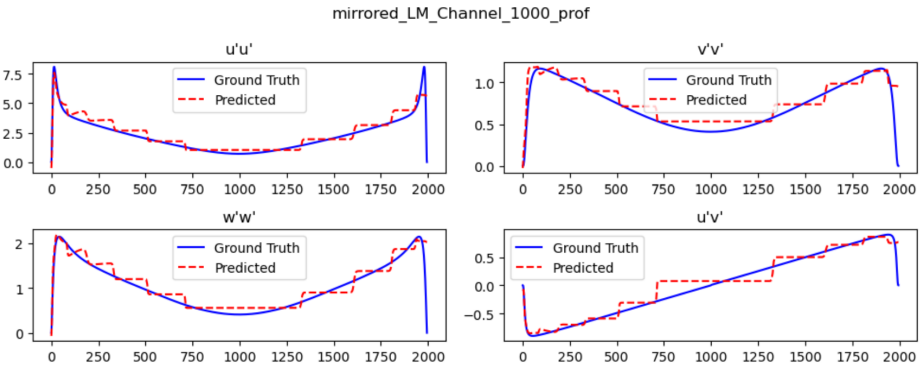

Figure 119: Unified RANS Based PINN Channel 20 Neurons & 8 Hidden Layers

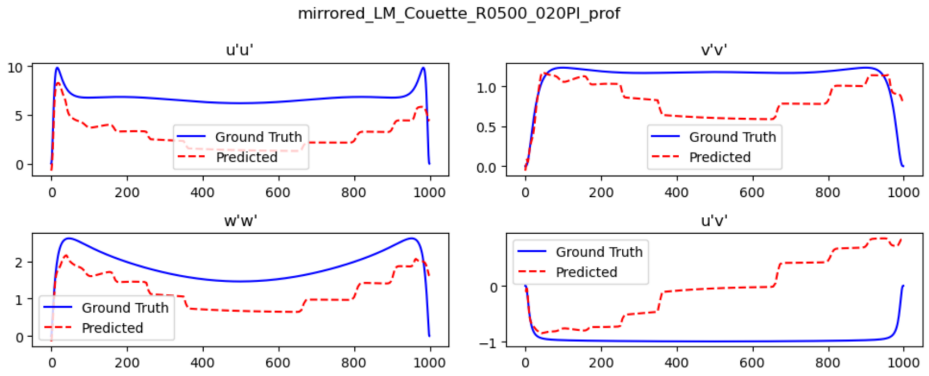

Figure 120: Unified RANS Based PINN Couette 20 Neurons & 8 Hidden Layers

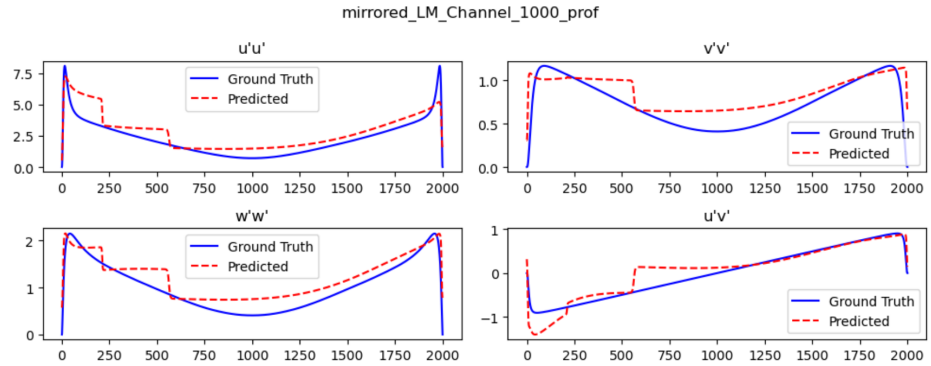

Figure 121: Unified PySindy Based PINN channel 10 Neurons & 2 Hidden Layers

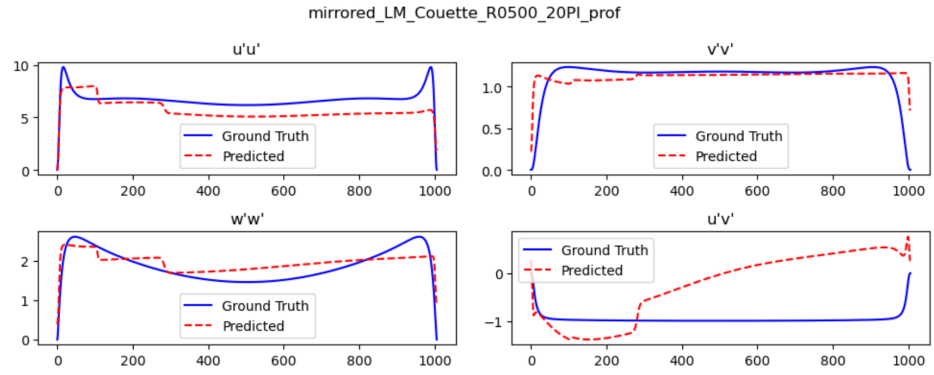

Figure 122: Unified PySindy Based PINN Couette 10 Neurons & 2 Hidden Layers

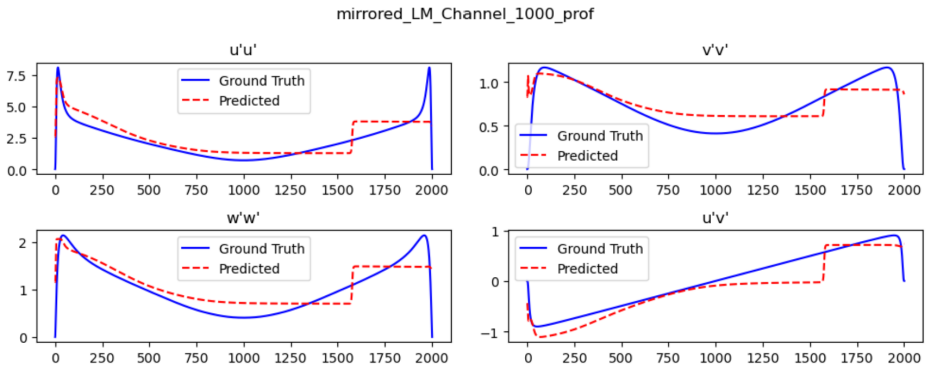

Figure 123: Unified PySindy Based PINN Channel 10 Neurons & 8 Hidden Layers

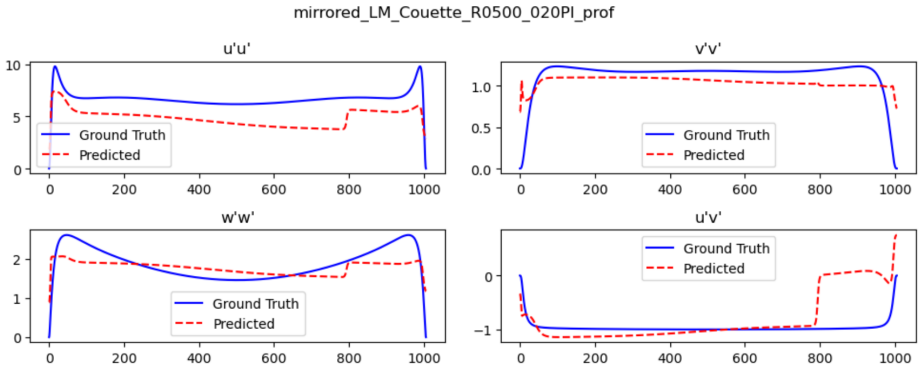

Figure 124: Unified PySindy Based PINN Couette 10 Neurons & 8 Hidden Layers

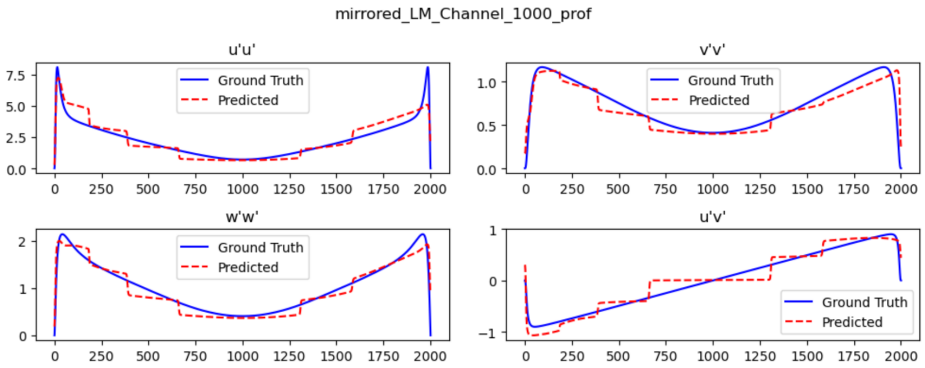

Figure 125: Unified PySindy Based PINN Channel 20 Neurons & 2 Hidden Layers

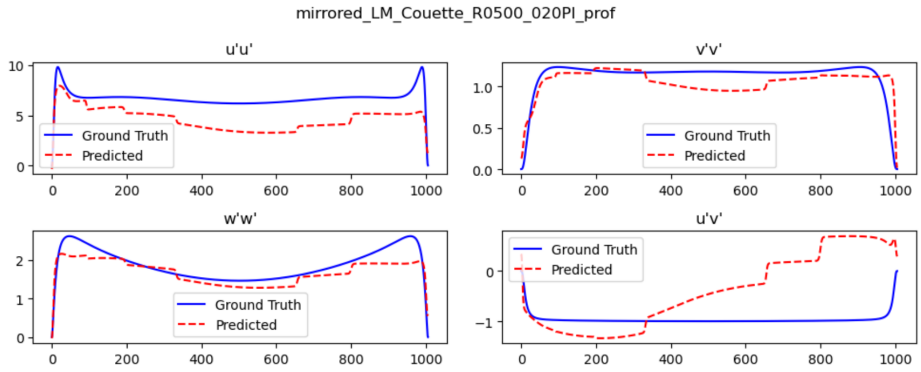

Figure 126: Unified PySindy Based PINN Couette 20 Neurons & 2 Hidden Layers

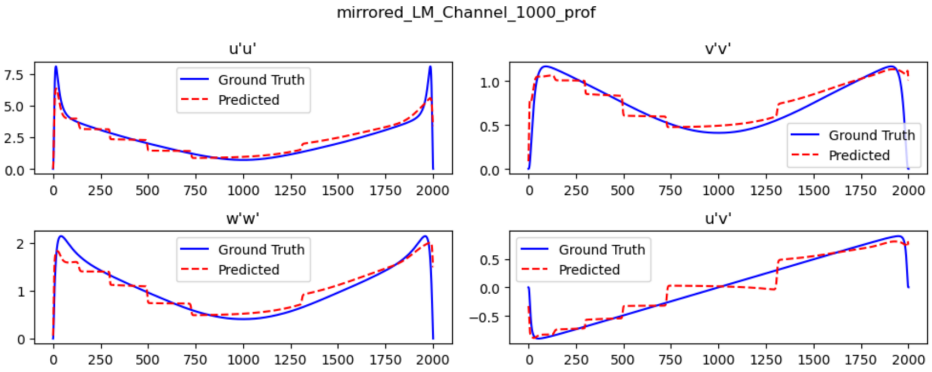

Figure 127: Unified PySindy Based PINN Channel 20 Neurons & 8 Hidden Layers

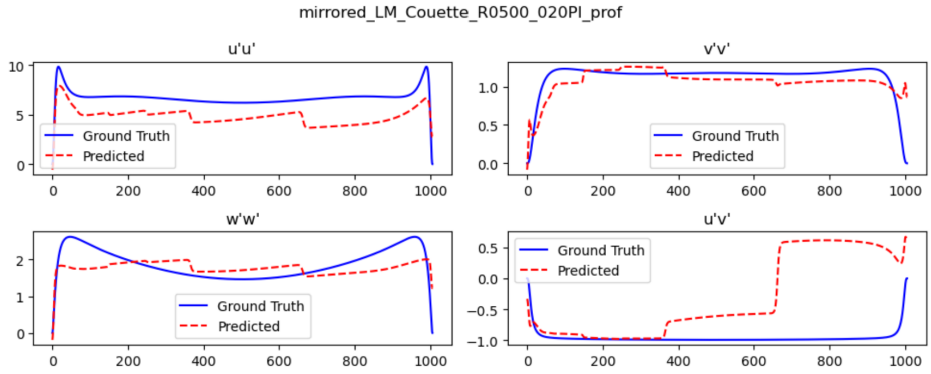

Figure 128: Unified PySindy Based PINN Couette 20 Neurons & 8 Hidden Layers

11 Appendix D: Final Model Results

11.1 Appendix D1: Channel Flow Results

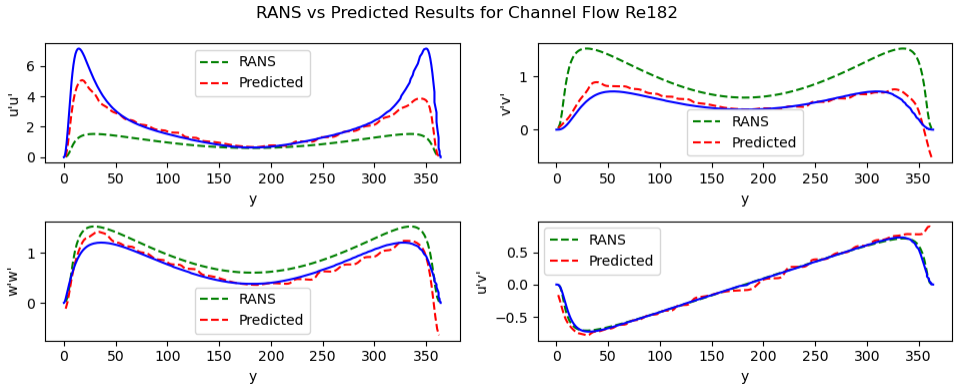

Figure 129: RANS, PINN and DNS predictions for the Reynolds Stresses for Channel Flow 180

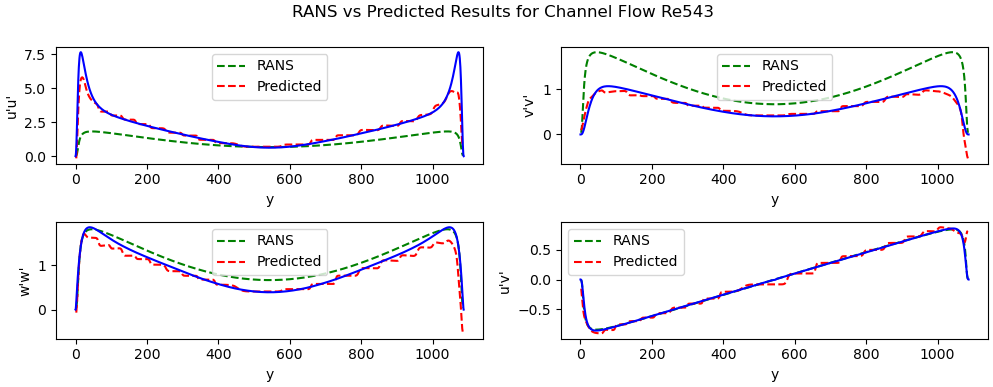

Figure 130: RANS, PINN and DNS predictions for the Reynolds Stresses for Channel Flow 500

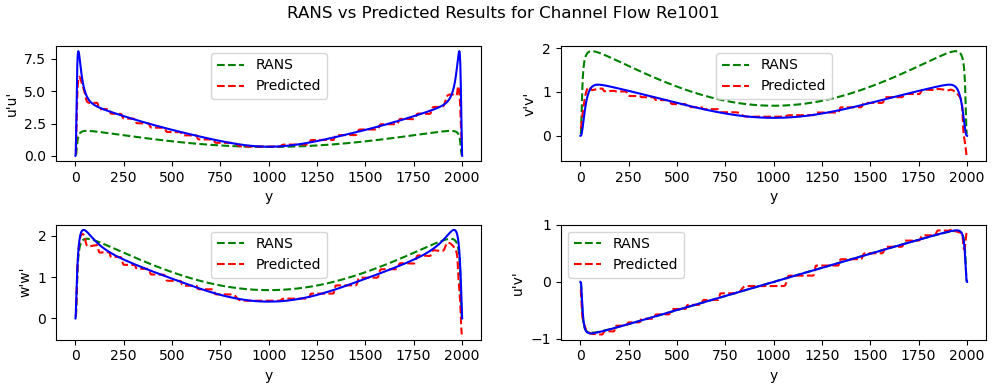

Figure 131: RANS, PINN and DNS predictions for the Reynolds Stresses for Channel Flow 1000

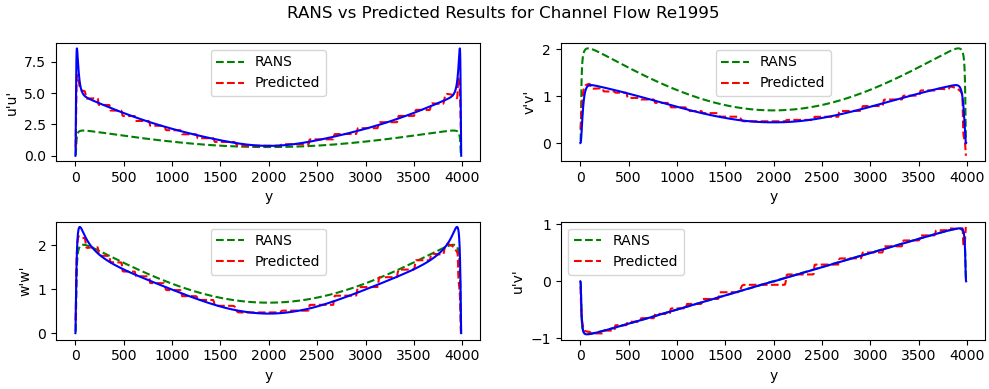

Figure 132: RANS, PINN and DNS predictions for the Reynolds Stresses for Channel Flow 2000

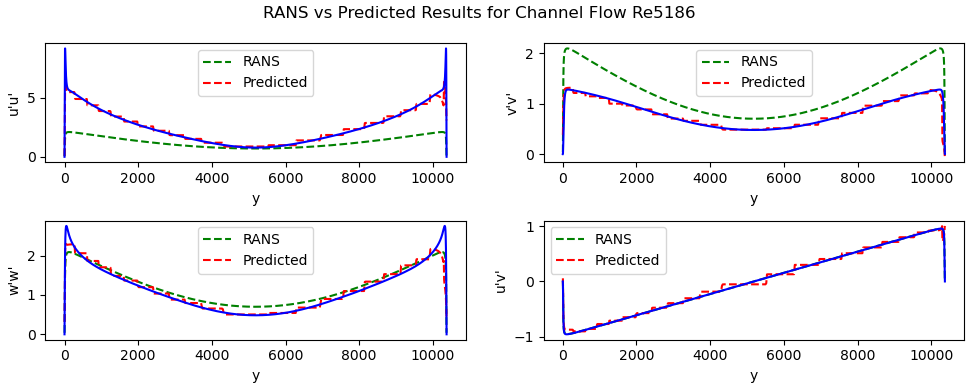

Figure 133: RANS, PINN and DNS predictions for the Reynolds Stresses for Channel Flow 5200

11.2 Appendix D2: Couette Flow Results

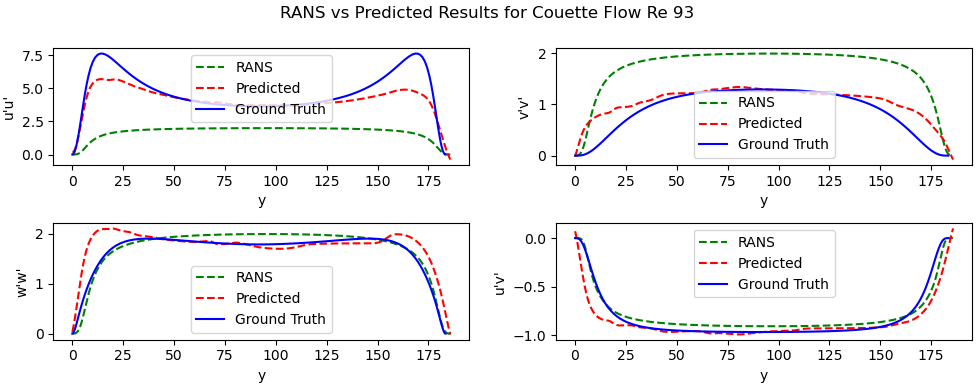

Figure 134: RANS, PINN and DNS predictions for the Reynolds Stresses for Couette Flow 93

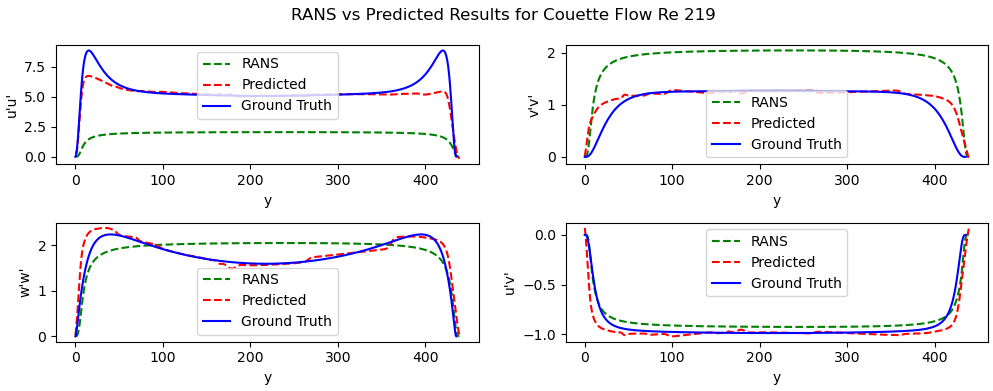

Figure 135: RANS, PINN and DNS predictions for the Reynolds Stresses for Couette Flow 220

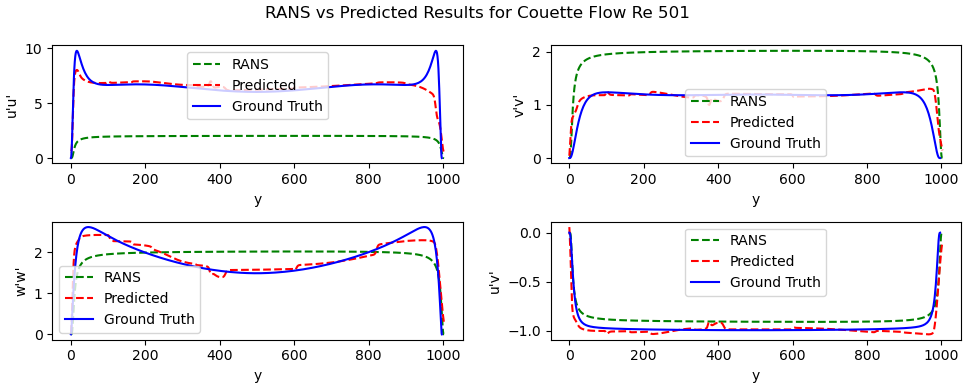

Figure 136: RANS, PINN and DNS predictions for the Reynolds Stresses for Couette Flow 500

11.3 Appendix D3: Additional Channel Flow Results

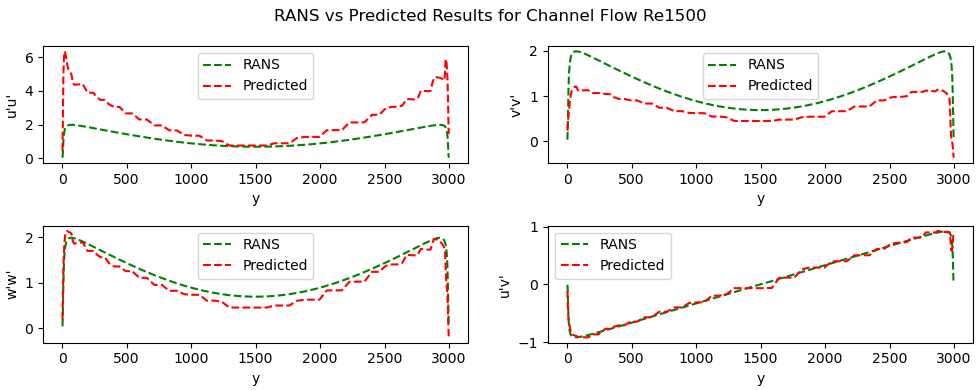

Figure 137: RANS and PINN predictions for the Reynolds Stresses for Channel Flow 1500

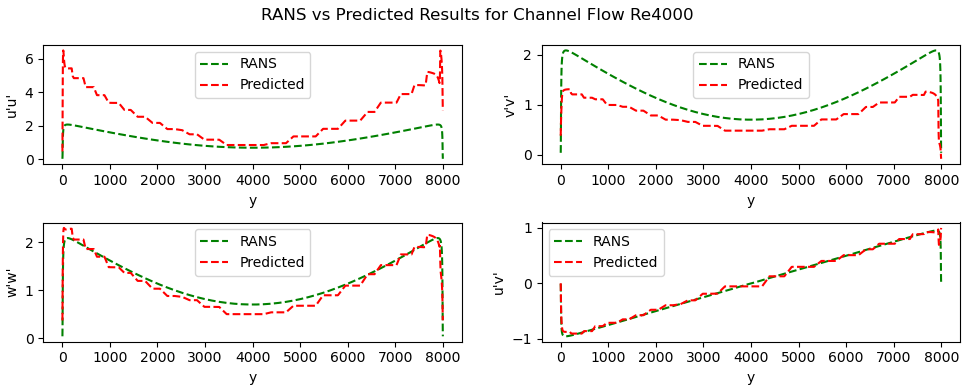

Figure 138: RANS and PINN predictions for the Reynolds Stresses for Channel Flow 4000

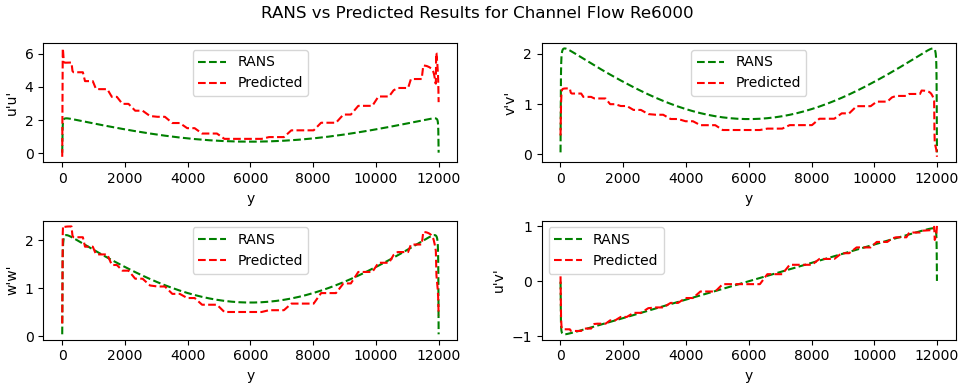

Figure 139: RANS and PINN predictions for the Reynolds Stresses for Channel Flow 6000

12 Appendix E: Team Management

12.1 Appendix E1: Meeting Agenda

MSc CSTE Group Project Meeting
14 March 2024
Agenda

1. Attendance
   Daniel James Rogers
   Muhammad Gillani
   Adam Sohail
   Gabriel Maire
   Venceslas Bignon
   Hyejoon Lee (Meeting Notes)
   Yasser El Karkouri

2. Absences
   None
3. Apologies
   None

4. Actions from the previous meeting

   Daniel James Rogers
   - Review the paper about implementation of new RANs model on fluent
   - Implement RANs model with matlab
   - Review the paper about how to derive the closure model

   Muhammad Gillani
   - Literature Review – Neural Networks (Paper NN & CNN Image Analysis)

   Gabriel Maire
   - Literature Review
   - PySindy Tutorials (Technical Aspects)
   - Miro map to sort out entire concepts

   Venceslas Bignon
   - Draw the entire structure of the classes
   - Extract the parameter of channel, couette and hill data
   - Create tests on the function
   - Get values on .vtf file (ongoing)

   Hyejoon Lee
   - Analysis flow data from database
   - Draw pipeline to link with data from CED

   Adam
   - Literature Review – Neural Networks (Paper NN & CNN Image Analysis)

   Yasser El Karkouri
   - Literature Review
   - PySindy Tutorials (Technical Aspects)
   - Linkage RANS Equations in PySindy

5. Discussion on Following Tasks

   Daniel James Rogers
   -
   CIDA (NN)
   - Understand each equation and how the model has been used
   - Provide a summary of paper

CIDA (PySindy) (Meeting Notes)

- Delve into the way how can the equation/variables be implemented by pysindy

Venceslas Bignon

- Get values on .vtf file

6. Any other business
   a) Reflect the feedback from the last meeting with professors
   b) Miro map
   c) Share the difficulties each memeber have faced and discuss about them

7. Date of next meeting

   Thursday 21/03/2024 - 11:00 (UK Time)

MSc CSTE Group Project Meeting
7 March 2024
Agenda
Microsoft Teams

1. **Attendance**:
   Adam Sohail (Meeting Notes)
   Daniel James Rogers
   Muhammad Gillani
   Gabriel Maire
   Venceslas Bignon
   Hyejoon Lee
   Yasser El Karkouri

2. **Apologies**:
   - None

3. **Absences**:
   - None

- **Actions from the previous meeting**:
  Adam SOHAIL:
  - Literature Reviews
  - Videos on Machine Learning applied to the Fluids Dynamic Field
  - Using Machine Learning/ Deep Learning to optimize images of flows

  Daniel James Rogers
  - Literature Reviews on how to drive RANS equation & viscosity models
  - Start using ICEM software (created some macros to find the y+ and the Meshes)

  Muhammad Gillani:
  - Start making the Logo of the Group
  - Videos on Machine Learning applied to the Fluids Dynamic Field
  - Literature Reviews

  Yasser El Karkouri:
  - Literature Reviews (DNS and CFD papers)
  - Videos on Machine Learning applied to the Fluids Dynamic Field
  - PySindy: Read documentation & some papers related

  Gabriel Maire:
  - Literature review
  - PySindy tutorials & documentation
  - Getting familiar with computer vision

  Hyejoon Lee:
  - PySindy & Jupyter Notebook tutorials
  - Literature reviews on PySindy models

  Venceslas Bignon:
  - Manual for GitHub newbies
  - Start creating classes for DNS data (to store and plot the data)

- **Discussion on Following Tasks**:
  - Daniel:

        - Understand the requirement of PySindy to understand how the DNS data need to be used by the PySindy team
        - Continue working on ICEM Software to automate everything.
    - Yasser El Karkouri:
        - Collaborate with PySindy Team as well as Daniel to start the pre-processing of the data
    - Muhammad Gillani:
        - Work with Adam to find models that could be used for the Images enhancement approach.
    - Gabriel Maire:
        - Collaborate with the PySindy team to start build a model
    - Venceslas Bignon:
        - Finish the structure for the DNS data
        - Presentation of the structure to be discuss with the whole team
    - Hyejoon Lee:
        - Collaborate with the PySindy team to start build a model
    - Adam Sohail:
        - Work with Muhammad to find models that could be used for the images enhancement approach.

4. **Any other business:**
    a. Submission discussion: Agreed to send a group submission document before 7 P.M for someone of the group who will be changed weekly.
    b. Discussion of new approach:
        i. Working in images using ML
    c. Risk assessment: Talk about it on the next meeting with the teacher.

5. **Date of next meeting**:
    - Monday 11/03/2020 - 12:00 (UK Time)
    - Thursday 14/03/2024 - 12:00 (UK Time)

MSc CSTE Group Project Meeting Agenda
Monday 11th March 2024

Attendance:
Adam Sohail
Daniel James Rogers
Muhammad Gillani (Meeting Notes)
Gabriel Maire
Venceslas Bignon
Hyejoon Lee
Yasser El Karkouri

Apologies:
None

Absences:
None

Actions from the previous meeting:

Daniel James Rogers:
1. Literature Review and Deep Literature Review
2. Fluent Automation of TUI

Yasser El Karkouri:
1. Collaborate with PySindy Team as well as Daniel to start the pre-processing of the data
2. Literature review and documentations on how to use PySindy

Muhammad Gillani:
1. Literature Review and Research for Machine Learning

Gabriel Maire:
1. Literature Review of PySindy

Venceslas Bignon:
1. Finish the structure for the DNS data
2. Presentation of the structure to be discuss with the whole team

Hyejoon Lee:
1. Literature Review and Research for PySindy

Adam Sohail:
1. Setting up GITHUB
2. ML Basic Lecture Review
3. Literature Review for Deep Neural Networks and Turbulence Modelling

Discussion on Following Tasks:

Feedback from the supervisor meeting
1. The document that is submitted needs to be two pages of what we have done collectively as a team, perhaps paste in some figures to support arguments.

CIDA Machine Learning Team:
1. ML team can enhance the results as the mean velocity is already quite good from the RANS model and focus on improving the RANS Reynolds Stress Tensor.

2. Do a comprehensive comparison of DNS and RANS.

CIDA PySindy Team:

1. Downsize team at some point
2. Produce some sort of results
3. Establish equation structure

CFD Team:

1. Complete automation of RANS Model.
2. Derivation of closure model.
3. DNS and RANS comparison and equation mathematics on how to obtain closure equations.

Software Team:

1. Classes for DNS data.
2. MATLAB to Python code conversion.

General Goals:

- Create the GANTT chart, TRELLO utilisation
- Previous and Next Task Discussion

Any other business:

- Submission discussion: Agreed to send a group submission document before 7 P.M for someone of the group who will be changed weekly.
- For everyone to be familiar with GITHUB
- Have TRELLO set by EOD

Date of next meeting:
Thursday 14/03/2024 - 11:00 (UK Time)
Monday 18/03/2020 - 13:00 (UK Time)

MSc CSTE Group Project Meeting Agenda
Thursday 14th March 2024

1. Attendance
   Daniel James Rogers
   Muhammad Gillani
   Adam Sohail
   Gabriel Maire
   Venceslas Bignon
   Hyejoon Lee (Meeting Notes)
   Yasser El Karkouri

2. Absences
   N/A
3. Apologies
   N/A

- Actions from the previous meeting

  Daniel James Rogers
  - Review the paper on the implementation of the new RANs model on fluent
  - Implement the RANs model with MATLAB
  - Review the paper on how to derive the closure model

  Muhammad Gillani
  - Literature Review – Neural Networks (Paper NN & CNN Image Analysis)

  Gabriel Maire
  - Literature Review
  - PySindy Tutorials (Technical Aspects)
  - Miro map to organize entire concepts

  Venceslas Bignon
  - Draw the entire structure of the classes
  - Extract the parameters of channel, couette and hill data
  - Create tests for the functions
  - Obtain values from .vtr file (ongoing)

  Hyejoon Lee
  - Analysis flow data from the database
  - Draw a pipeline to link with data from CED

  Adam

- Literature Review – Neural Networks (Paper NN & CNN Image Analysis)

Yasser El Karkouri
- Literature Review
- PySindy Tutorials (Technical Aspects)
- Miro map to organize entire concepts

4. Discussion on Following Tasks

Daniel James Rogers
- Review the code for Fluent
- Meeting with the CIDA student to define parameters and discuss
- More literature review

CIDA (NN)
- Understand each equation and how the model has been used
- Provide a summary of the paper

CIDA (PySindy) (Meeting Notes)
- Delve into the way how the equation/variables can be implemented by PySindy

Venceslas Bignon
- Obtain values from .vtf file

5. Any other business
a) Reflect the feedback from the last meeting with professors
b) Miro map
c) Share the difficulties each member has faced and discuss them

6. Date of next meeting

Thursday 18/03/2024 - 9:30 (UK Time)

MSc CSTE Group Project Meeting Agenda
Thursday 21st March 2024

1. Attendance

Daniel James Rogers
Muhammad Gillani
Adam Sohail
Gabriel Maire
Venceslas Bignon
Hyejoon Lee (Meeting Notes)
Yasser El Karkouri

2. Absences

N/A

3. Apologies

N/A

4. Actions from the previous meeting

Daniel James Rogers

- Analysing the code to achieve full automation for data processing
- Execute the code for various scenarios.
- Adjust the mesh parameter for improved outcomes.

Muhammad Gillani / Adam

- Get Results from a first Simple Neural Network Model (Training , Testing , Validation)
- Try to use the pin algorithm (First Prototype)

Gabriel Maire / Yasser El Karkouri

- Preparing Data for the Model development
- Exploring the Governing Equations for Direct Numerical Simulation Scenarios (Reynolds Numbers between 180 and 5200)
- Testing and Improving the equations obtained

Venceslas Bignon

- Write the code to retrieve all data formats.
- Assist fellow members with GitHub tasks.

Hyejoon Lee

- Creating user defined functions in C to serve as an intermediary between PySINDy and MATLAB

- Developed code to construct a class containing data relevant to "Terms in the Reynolds Stress Transport Equation (RSTE)."

5. Discussion on Following Tasks

Daniel James Rogers

- Review the code for Fluent
- Meeting with the CIDA student to define parameters and discuss
- More literature review

CIDA (NN)

- Understand each equation and how the model has been used
- Provide a summary of the paper

CIDA (PySindy) (Meeting Notes)

- Delve into the way how the equation/variables can be implemented by PySindy

Venceslas Bignon

- Obtain values from .vtf file

6. Any other business

a) Reflect the feedback from the last meeting with professors
b) Evaluation of Result Validity by All Members
c) Identification of Questions for Supervisors by All Members
d) Assessment of Methods by All Members
e) Exploration of Challenges in the Task

7. Date of next meeting

Thursday 25 /03/2024 after the supervised meeting

OUTOFBOUNDS

Cranfield University

MSc CSTE Group Project Meeting Agenda
Monday 25st March 2024

1. Attendance

Daniel James Rogers
Muhammad Gillani
Adam Sohail
Gabriel Maire
Venceslas Bignon (meeting minutes)
Hyejoon Lee
Yasser El Karkouri

2. Absences

N/A

3. Apologies

N/A

4. Actions from the previous meeting

NN Group (Muhammad Gillani / ADAM) :

- First Version Of the PINN algorithm
- Do more Research to increase the DNS data.
- Focus On getting better results for Pin method, Reviewing the existing code
- Try to use the equations discovered by the PySINDy group

CFD Group (Daniel James Rogers) :

- Reviewing The Code to get Better results
- Find user defined function in fluent and try the atomisation
- Review the result obtained from the 5 cases

PySINDy Group (Yasser El Karkouri / Gabriel Maire) :

- Testing and Improving the equations obtained
- Implement new features to get better equations that could be generalized
- Collaborate with the NN group to help them getting better results

SETC Group (Venceslas Bignon) :

- Testing Python libraries for interpolation
- Help the other members with github

CIDA Group (Hyejoon Lee) :

- developing user define functions in C that acts as a bridge between PySINDy and MATLAB
- finish the UDF C code after obtaining the PySINDy equation

5. Discussion on Following Tasks

Daniel James Rogers (CFD Group)
- UDF implementation
- Obtain result on RTI (Reynold transport equation) model
- Export y+
- Meanings on CED part

Muhammad Gillani (NN Group)
- Improving K model using ML

Adam Sohail (NN Group)
- Find covariance using PINN

Gabriel Maire (PySINDy Group)
- Improving K model using PySINDy

Yasser El Karkouri (PySINDy Group)
- Find covariance using PySINDy

Venceslas Bignon (SETC Group)
- Debugging UDF on fluent with Daniel
- Program function to get norms out of results

Hyejoon Lee (CIDA Group)
- Finish the UDF C code after obtaining the PySINDy equation
- Help on PySINDy if needed

6. Any other business

a) Reflect the feedback from the last meeting with professors (12:30 - 12h40)
b) How to improve our collaboration/work (12:40 - 12:50)
c) Have a clear goal on the final pipeline (12:50 – 13:30)

7. Date of next meeting

Thursday 29/03/2024 10h

OUTOFBOUNDS

Cranfield University

MSc CSTE Group Project Meeting Agenda
Thursday 29st March 2024

1. Attendance

Daniel James Rogers
Muhammad Gillani
Adam Sohail
Gabriel Maire
Venceslas Bignon
Hyejoon Lee (Meeting Notes)
Yasser El Karkouri

2. Absences

N/A

3. Apologies

N/A

4. Actions from the previous meeting

Daniel James Rogers

- Analysing the code to achieve full automation for data processing
- Execute the code for various scenarios.
- Adjust the mesh parameter for improved outcomes.
- Started implementing the couette flow equations in fluent
- Got first results for couette flow
- Worked on the correction of UDF with Venceslas

Muhammad Gillani / Adam

- Improved Simple Neural Network Model by way of optimizers and other parameters
- Focused on optimizing equations for K
- Try to use the pin algorithm (First Prototype)
- Improved Simple Neural Network and found good first results (Adam)

Gabriel Maire / Yasser El Karkouri

- Preparing Data for the Model development
- Exploring the Governing Equations for Direct Numerical Simulation Scenarios (Reynolds Numbers between 180 and 5200)
- Testing and Improving the equations obtained
- Created python code and csv files with RANS and DNS interpolated data for PySINDy and ML team to work with.

- Improved PySINDy models to get generalized equations giving covariances and K as functions of RANS results (U and velocity gradients).

Venceslas Bignon

- Write the code to retrieve all data formats.
- Assist fellow members with GitHub tasks.
- Wrote a python code for interpolation using norms
- Helped Daniel with Fluent
- Helped team with GitHub

Hyejoon Lee

- Creating user defined functions in C to serve as an intermediary between PySINDy and MATLAB
- Developed code to construct a class containing data relevant to "Terms in the Reynolds Stress Transport Equation (RSTE)."
- Worked on compilation of udf file with Fluent on Crescent.
- Implementation of new custom libraries for PySINDy.

5. Discussion on Following Tasks

Daniel James Rogers

- Review the code for Fluent
- Debug compilation problem of Fluent using Virtual Machines
- Work on State of the Art and introduction of the report
- Implement UDFs
- Improve Couette flow results

CIDA (NN)

- Understand each equation and how the model has been used
- Provide a summary of the paper
- Optimize NN code
- Work on report
- Work on a NN to solve the k-omega equation

CIDA (PySindy) (Meeting Notes)

- Delve into the way how the equation/variables can be implemented by PySindy
- Interpolate new data
- Generalize equations to Couette flow
- Work on report

Venceslas Bignon

- Obtain values from .vtf file
- Implement UDFs

6. Any other business

a) Reflect the feedback from the last meeting with professors
b) Evaluation of Result Validity by All Members
c) Assessment of Methods by All Members
d) Exploration of Challenges in the Task
e) Discussion about the writing of the report

7. Date of next meeting

Monday 01 /04/2024

MSc CSTE Group Project Meeting Agenda
Monday 1st April 2024

1. Attendance

Daniel James Rogers
Muhammad Gillani
Adam Sohail
Gabriel Maire
Venceslas Bignon
Hyejoon Lee (Meeting Notes)
Yasser El Karkouri

2. Absences

N/A

3. Apologies

N/A

4. Actions from the previous meeting

Daniel James Rogers

- Worked on the correction of UDF with Venceslas

Muhammad Gillani / Adam

- Improved Simple Neural Network Model by way of optimizers and other parameters
- Amend neural network model (classic NN model)
- Try to use the pin algorithm (First Prototype)
- Improved Simple Neural Network and found good first results (Adam)

Gabriel Maire / Yasser El Karkouri

- Analysis data of new interpolated data to find unusual features and pattern
- Testing and Improving the equations obtained
- Improved PySINDy models to get generalized equations giving covariances and K as functions of RANS results (U and velocity gradients).
- Interpolated Couette flow data

Venceslas Bignon

- Write the code to retrieve all data formats.
- Assist fellow members with GitHub tasks.
- Wrote a python code for interpolation using norms
- Helped Daniel with Fluent

- Helped team with GitHub

Hyejoon Lee

- Creating user defined functions in C to serve as an intermediary between PySINDy and MATLAB
- Developed code to construct a class containing data relevant to "Terms in the Reynolds Stress Transport Equation (RSTE)."
-

5. Discussion on Following Tasks

Daniel James Rogers

- Produce and create more training data
- Work on State of the Art and introduction of the report
- Implement UDFs
- Improve Couette flow results

CIDA (NN)

- Work on Report
- Test and train with interpolated data
- Improve the model

CIDA (PySindy) (Meeting Notes)

- Interpolate new data
- Generalize equations to Couette flow
- Work on report

Venceslas Bignon

- Produce and create more training data

6. Any other business

a) Reflect the feedback from the last meeting with professors
b) Evaluation of Result Validity by All Members
c) Assessment of Methods by All Members
d) Exploration of Challenges in the Task
e) Discussion about the writing of the report

7. Date of next meeting

Monday 01 /04/2024

OUTOFBOUNDS

Cranfield University

MSc CSTE Group Project Meeting Agenda
Thursday 4st March 2024

1. Attendance

Daniel James Rogers (Meeting Notes)
Muhammad Gillani
Adam Sohail
Gabriel Maire
Venceslas Bignon
Hyejoon Lee
Yasser El Karkouri

2. Absences

N/A

3. Apologies

N/A

4. Actions from the previous meeting

Daniel James Rogers

- Export Couette Data and validate the velocity magnitude and gradient with DNS data
- Create Code for automatically executing Channel Flow Meshes and Fluent Simulations with a rotated domain.
- Improve script for rotating DNS data and validate with RANS data.

Muhammad Gillani

- K-Optimization algorithm with ML. Achieved good results by training independently (Channel and Couette) and obtained good results by training both cases.
- Working on the Introduction to Machine Learning and K optimization Machine Learning
- **Adam Sohail**Make PINN work with equations
- Mirroring DNS data to adequate RANS (w/ Yasser)
- Trained model with Channel and Couette and gave good results
- Writing PINN part of results

Gabriel Maire

- Work on the PySIndy part of the report (Methodology, Interpolation and Data Analysis)
- Analysis and Interpolation Algorithm
- Help Yasser with Feature Selection
- Optimize K PySIndy Equatios

Yasser El Karkouri

- Mirroring DNS data to adequate RANS (w/ Muhammad)
- Enhance the RST PySindy optimization equations by training with Couette and Channel
- Work on the PySIndy part of the report (Methodology)

Venceslas Bignon

- Initial script to rotate the DNS data (w/ Daniel)
- Set Fluent Machine Virtual Environment for UDF implementation (w/ Faye)
- Reflect on assembling all the parts of the group

Hyejoon Lee

- Set Fluent Machine Virtual Environment for UDF implementation (w/ Vince)
- Create First Version of UDF with PySindy Equations.
- Fix several syntax errors and debug by reading the manual and some papers.
- Write PySindy Background part of the Report

5. Discussion on Following Tasks

Daniel James Rogers

- Review the code for Rotating the Data and see whether we can skip Fluent for RANS Data
- Explain DNS and RANS equations in the report.
- Implement UDF in fluent simulations

Muhammad Gillani

- Review the ML code and improve results for K.
- Working on the Introduction to Machine Learning and K optimization Machine Learning part of the report

Adam

- Optimize PINN Code
- Working on the PINN part of the Report

Gabriel

- Implement K equation in Boussines Hypothesis With RANS Data
- Continue writing the report.

Yasser

- Improve RST Equations with PySIndy
- Continue writing the report.

Venceslas Bignon

- Compile of the work regarding data organization from all the members and write the report (Interpolation, Rotation, classes…)
- Define the frame for global integration of the work.

Hyejoon Lee

- Fix UDF code compilation errors
- Meet with Daniel to set new simulations in fluent and validate code.

6. Any other business

a) Discuss whether it is worth it to Enhance the Data by rotating the domain.
b) Discussion of features and variables used in PySIndy and Machine Learning
c) Discussion of turbulent viscosity in DNS
d) Discussion of the general overview of the report

7. Date of next meeting

Monday 08/04/2024 after the supervised meeting

OUTOFBOUNDS

Cranfield University

MSc CSTE Group Project Meeting Agenda
Thursday 8th April 2024

1. Attendance

Daniel James Rogers
Muhammad Gillani
Adam Sohail (Meeting Notes)
Gabriel Maire
Venceslas Bignon
Hyejoon Lee
Yasser El Karkouri

2. Absences

N/A

3. Apologies

N/A

4. Actions from the previous meeting

5. Discussion on Following Tasks

Daniel James Rogers

- Improve RANS data by using Ryenolds Transport Equations
- Implement RANS equations and future pysindy equations in BVP
- Boundary Value Problem Tool box.

Muhammad Gillani

- Start the poster presentation.
- Working the Machine Learning section of the report (presentation aspect)

Adam

- Enhance the PINN model by adding new equations as well as boundary conditions.
- Working on the PINN part of the Report (technical part)

Gabriel

- Improve covariance model using Tamas advice.
- Continue writing the report with the PySindy team.

Yasser

- Improve covariance model using Tamas advice.
- Continue writing the report with the PySindy team.

Venceslas Bignon

- Continue writing the report on the interpolation, mirroring as well as data enhancement.

Hyejoon Lee

- Work with the PySindy team to help get better result.
- data augumentation, test robustness, validate with genetic algorithm(if possible), derive governing equation

6. Any other business

a) Discuss about how to split the work for the last two weeks (report, poster, and results from the teams).
b) Discussion of getting equations using second order derivatives (PySindy)
c) Discussion on getting new DNS data for Machine Learning Team
d) Discussion of the general overview of the report

7. Date of next meeting

Thursday 11/04/2024

OUTOFBOUNDS

Cranfield University

MSc CSTE Group Project Meeting Agenda
Thursday 11st April 2024

1. Attendance

Daniel James Rogers
Muhammad Gillani
Adam Sohail
Gabriel Maire
Venceslas Bignon (Meeting Notes)
Hyejoon Lee
Yasser El Karkouri

2. Absences

N/A

3. Apologies

N/A

4. Actions from the previous meeting

Daniel James Rogers

- Launch RSTE in ANSYS unsuccessfully for the moment
- Model the equation on MATLAB for the channel and Couette flow

Muhammad Gillani

- Focus on the poster
- Report redaction support (ML, literature review)

Adam

- Use different model for Couette and channel to generalize the pipeline

Gabriel Maire

- Integrating rans equation using pysindy
- Data analysis

Yasser El Karkouri

- Refactor the interpolation function

Venceslas Bignon

- Create some test for data interpolation and data mirroring
- Literature review on AI testing

Hyejoon Lee

- used a genetic algorithm to create a model and validate the PySINDy model

5. Discussion on Following Tasks

Daniel James Rogers

- Same tasks

Adam

- Plot all method to compare
- Write on the report

Gabriel Maire

- Work on the same with 2nd ODE
- Write report

Muhammad Gillani

- Discussion with the group to agree on what put on the poster

Hyejoon Lee

- Continue on the previous task

Yasser El Karkouri

- Interpolation couette flow
- Improve model with 2nd order equation

Venceslas Bignon

- Keep working on the testing part

6. Any other business

a) Discussion on the report (16h15 – 16h25)
b) Reflection on variables (16h30 – 16h50)

7. Date of next meeting

Monday 15 /04/2024

12.2 Appendix E2: Meeting Minutes

MSc CSTE Group Project meeting
Monday 8th March 2021

1. **Present:** Daniel James Rogers [DJR] (Meeting Notes), Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]
   Absent: No-one
   Apologies:

2. **Actions from Previous meeting (Task Review)**

   Actions 1, 2 from DJR closed.
   Actions 1,2 closed and Action 3 pending from MG.
   Actions 1,3,4 ongoing and Action 2 closed from GM.
   Action 1 closed and Action 2 ongoing from VB.
   Action 1 ongoing from HL.
   Action 1 ongoing from AS.
   Action 1 and 2 ongoing from YEK.

Minutes accepted: 15 minutes (10:00:00-10:15:00)

3. **Discussion on Following Tasks**

   Actions 1, 2 ongoing from DJR closed.
   Actions 1,2 ongoing from CIDA NN (MH and AS).
   Actions 1,2 ongoing from CIDA PySIndy (YEK, HL and GM).
   Action 1,2,3 ongoing from VB.

Minutes accepted: 10 minutes (10:35:00-10:45:00)

4. **Other bousiness**
   a. RANS Derivation Presentation [DJR]

Minutes Accepted: 15 minutes (10:15:00-10:30:00)

   b. Gant and Trello for Task Management and Tracking

Minutes Accepted: 5 minutes (10:30:00-10:35:00)

5. **Date of next meeting**
   Date of next meeting Monday 15/03/2021 at 11am

MSc CSTE Group Project meeting
Monday 8th March 2021

1. **Present:** Daniel James Rogers [DJR] (Meeting Notes), Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]
   Absent: No-one
   Apologies: No-one

2. **Actions from Previous meeting (Task Review)**
   Action 1,2,3 ongoing from AS.
   Actions 1 and Action 2 (Geo Mesh) closed and Action 2 (y+ integration) ongoing from DJR.
   Actions 1 closed and Action 2, 3 pending from MG.
   Action 1 ,2,3 ongoing and Action 1 (Report) closed from YEK.
   Actions 1,2,3 ongoing from GM.
   Action 1,2 ongoing from HL.
   Action 1 closed and Action 2 ongoing from VB.

Minutes accepted: 10 minutes (12:15:00-10:25:00)

3. **Discussion on Following Tasks**

   Actions 1 not started and Action 2 ongoing from DJR closed.
   Action 1 ongoing from YEK.
   Action 1 ongoing from MG.
   Action 1 ongoing from GM.
   Action 2 (ppt) closed, Action 2 (presentation) not started and Action 1 ongoing from VB.
   Action 1 ongoing from HL.
   Action 1 ongoing from AS.

Minutes accepted: 10 minutes (12:25:00-10:35:00)

4. **Other bousiness**
   a. Submission Details Discussion [All members]

Minutes Accepted: 15 minutes (12:00:00-12:15:00)

   b. Discussion of new Approach

Minutes Accepted: 15 minutes (12:35:00-12:50:00)

   c. Risk Assessment Approach

Minutes Accepted: 10 minutes (12:50:00-13:00:00)

5. **Date of next meeting**
   Monday 11/03/2020 - 12:00 (UK Time)
   Thursday 14/03/2024 - 12:00 (UK Time)

MSc CSTE Group Project Meeting Minutes
Monday 11th March 2024

1. **Present:** Daniel James Rogers [DJR] (Meeting Notes), Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]
   Present: CIDA Machine Learning Team [CIDAML], CIDA PySindy Team [CIDAPY], CFD Team [CFD], Software Team [S]

   Absent: No-one
   Apologies: No-one

2. **Actions from Previous meeting (Task Review)**
   Action 1,2 closed by DJR
   Actions 1, 2 closed by YEK
   Action 1 ongoing by MG
   Actions 1 ongoing by HL
   Action 1, 2 closed and Action 3 ongoing by AS

3. **Discussion on Following Tasks**

   Action 1, 2 ongoing by CIDAML
   Action 1, 2, 3, ongoing by CIDAPY
   Action 1, 2, closed and Action 3 ongoing by CFD
   Action 1, 2, ongoing by S

4. **Other business**
   a. Submission Details Discussion [All members]

Minutes Accepted: 15 minutes (13:00:00-13:15:00)

   b. Discussion of using GITHUB and Trello

Minutes Accepted: 15 minutes (13:35:00-13:50:00)

5. **Date of next meeting**
   Thursday 14/03/2024 - 11:00 (UK Time)
   Monday 18/03/2020 - 13:00 (UK Time)

MSc CSTE Group Project Meeting Minutes
Thursday 14th March 2024

1.

Present: Daniel James Rogers [DJR] (Meeting Notes), Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]

Present: CIDA Machine Learning Team [CIDAML], CIDA PySindy Team [CIDAPY], CFD Team [CFD], Software Team [S]

Absent: No-one
Apologies: No-one

2. **Actions from Previous meeting (Task Review)**

Actions 2,3 closed and Action 1 ongoing from DJR.
Actions 1 ongoing from MG.
Actions 1 closed and Action 2,3 ongoing from GM.
Action 1,2,3 closed and Action 4 ongoing from VB.
Action 1,2 ongoing from HL.
Action 1 ongoing from AS.
Action 1 closed and Action 2,3 ongoing from YEK.

Minutes accepted: 15 minutes (11:00:00-11:15:00)

3. **Discussion on Following Tasks**

Actions 1, 2, 3 ongoing from DJR closed.
Actions 1,2 ongoing from CIDA NN (MH and AS).
Actions 1 ongoing from CIDA PySIndy (YEK, HL and GM).
Action 1 ongoing from VB.

Minutes accepted: 15 minutes (11:45:00-12:00:00)

4. **Other business**

- Discussion difficulties about task [All members]

Minutes Accepted: 15 minutes (11:15:00-11:40:00)

- Discussion of submission

Minutes Accepted: 5 minutes (11:40:00-: 11:45:00)

5. **Date of next meeting**

Monday 18/03/2020 – 9:30 (UK Time)

MSc CSTE Group Project Meeting Minutes
Thursday 21st March 2024

1.

Present: Daniel James Rogers [DJR] (Meeting Notes), Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]

Present: CIDA Machine Learning Team [CIDAML], CIDA PySindy Team [CIDAPY], CFD Team [CFD], Software Team [S]

Absent: No-one
Apologies: No-one

2. Previous Tasks Review

- **NN Group (Muhammad Gillani / ADAM) :**
  Action 1 :
  - Description : Simple NN Model
  - Status : Closed

  Action 2 (NN Group : ADAM)
  - Description : First Version Of the Pin algorithm
  - Status :On-Going
- **CFD Group (Daniel James Rogers) :**
  Action 1 :
  - Description :Reviewing The Code to get fully automated code for data
  - Status :Closed

  Action 2 :
  - Description :Run the code For different Cases
  - Status :Closed

  Action 3 :
  - Description :Correct the mesh Parameter to get better results
  - Status :Closed

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**
  Action 1 :
  - Description : Data Preprocessing for the model
  - Status :Closed

  Action 2 :
  - Description : Discovering Governing Equations For DNS cases (Re in 180 … 5200)
  - Status :Closed

  Action 3 :
  - Description : Testing and Improving the equations obtained
  - Status :On-Going

- **SETC Group (Venceslas Bignon) :**

  Action 1 :
  - Description : Code to recover all the data format
  - Status :Closed (could optimize it more)

  Action 2 :
  - Description : Help the other members with github
  - Status :Closed

- **CIDA Group (Hyejoon Lee) :**
  Action 1 :
  - Description : developing user define functions in C that acts as a bridge between PySINDy and MATLAB
  - Status :On-Going

  Action 2 :
  - Description : developed some code for creating a class with data for "Terms in Reynolds Stress Transport Equation (RSTE)"
  - Status :Closed

Minutes accepted: 30 minutes (12:00:00-12:30:00)

3. Discussion on Following Tasks

- **NN Group (Muhammad Gillani / ADAM) :**
  Action 1 :
  - Description : Do more Research to increase the DNS data .

  Action 2 :
  - Description : Focus On getting better results for Pin method , Reviewing the existing code

  Action 3 :
  - Description : Try to use the equations discovered by the PySINDy group

- **CFD Group (Daniel James Rogers) :**
  Action 1 :
  - Description :Reviewing The Code to get Better results

  Action 2 :
  - Description : Find user defined function in fluent and try the atomisation

  Action 3 :
  - Review the result obtained from the 5 cases

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**
  Action 1 :
  - Description : Implement new features to get better equations that could be generalized

  Action 2 :
  - Description : Collaborate with the NN group to help them getting better results

- **SETC Group (Venceslas Bignon) :**

  Action 1 :
  - Description : Testing Python libraries for interpolation

  Action 2 :
  - Description : Help the other members with github

- **CIDA Group (Hyejoon Lee) :**
  Action 1 :
  - Description : finish the UDF C code after obtaining the PySINDy equation

  Action 2 :
  - Description : Exploring new approaches to improve PySINDy model

Minutes accepted: 30 minutes (12:30:00-13:00:00)

4. Other business

- Discussion of the validity of the results [All members]

Minutes Accepted: 15 minutes (12:15:00-12:40:00)

- Discussion of the potential questions to ask to the supervisors[All members]

Minutes Accepted: 5 minutes (12:15:00-12:20:00)

- Review the approaches [All members]

Minutes Accepted: 5 minutes (12:15:00-12:20:00)

- Discuss The Task difficulties

Minutes Accepted: 5 minutes (12:40:00-: 11:45:00)

5. Date of next meeting

Monday 25/03/2020 after the supervised meeting

MSc CSTE Group Project Meeting Minutes
Monday 25st March 2024

1.

Present: Daniel James Rogers [DJR] (Meeting Notes), Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]

Present: CIDA Machine Learning Team [CIDAML], CIDA PySindy Team [CIDAPY], CFD Team [CFD], Software Team [S]

Absent: No-one
Apologies: No-one

2. Previous Tasks Review

- **NN Group (Muhammad Gillani / ADAM) :**
  Action 1 (NN Group : ADAM)
  - Description : First Version Of the PINN algorithm
  - Status : Closed

  Action 2 :
  - Description : Do more Research to increase the DNS data.
  - Status : Closed

  Action 3 :
  - Description : Focus On getting better results for Pin method , Reviewing the existing code
  - Status : Closed

  Action 4 :
  - Description : Try to use the equations discovered by the PySINDy group
  - Status : Closed

- **CFD Group (Daniel James Rogers) :**

Action 1 :
- Description : Reviewing The Code to get Better results
- Status : Closed

Action 2 :
- Description : Find user defined function in fluent and try the atomisation
- Status : Closed

Action 3 :
- Review the result obtained from the 5 cases
- Status : Closed

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**

Action 1 :
- Description : Testing and Improving the equations obtained
- Status : Closed

Action 2 :
- Description : Implement new features to get better equations that could be generalized
- Status : Closed

Action 3 :
- Description : Collaborate with the NN group to help them getting better results
- Status : Closed

- **SETC Group (Venceslas Bignon) :**

Action 1 :
- Description : Testing Python libraries for interpolation
- Status : Closed

Action 2 :
- Description : Help the other members with github
- Status : On-going

- **CIDA Group (Hyejoon Lee) :**

Action 1 :
- Description : developing user define functions in C that acts as a bridge between PySINDy and MATLAB
- Status : On-Going

Action 2 :
- Description : finish the UDF C code after obtaining the PySINDy equation
- Status : On going

Minutes accepted: 15 minutes (13:00:00-13:45:00)

3. Discussion on Following Tasks

Daniel James Rogers (CED Group)
- UDF implementation
- Obtain result on RTI (Reynold transport equation) model
- Export y+
- Meanings on CED part

Muhammad Gillani (NN Group)
- Improving K model using ML

Adam Sohail (NN Group)
- Find covariance using PINN

Gabriel Maire (PySINDy Group)
- Improving K model using PySINDy

Yasser El Karkouri (PySINDy Group)
- Find covariance using PySINDy

Venceslas Bignon (SETC Group)
- Debugging UDF on fluent with Daniel
- Program function to get norms out of results

Hyejoon Lee (CIDA Group)
- Finish the UDF C code after obtaining the PySINDy equation
- Help on PySINDy if needed

Minutes accepted: 15 minutes (13:45:00-14:00:00)

4. Other business

- Reflect the feedback from the last meeting with professors

Minutes Accepted: 10 minutes (12:30 – 12:40)

- How to improve our collaboration/work

Minutes Accepted: 10 minutes (12:40 – 12:50)

- Have a clear goal on the final pipeline

Minutes Accepted: 40 minutes (12:50 – 13:30)

5. Date of next meeting

Thursday 29/03/2020 10h

MSc CSTE Group Project Meeting Minutes
Thursday 29st March 2024

1.

Present: Daniel James Rogers [DJR] (Meeting Notes), Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]

Present: CIDA Machine Learning Team [CIDAML], CIDA PySindy Team [CIDAPY], CFD Team [CFD], Software Team [S]

Absent: No-one
Apologies: No-one

2. Previous Tasks Review

- **NN Group (Muhammad Gillani / ADAM) :**
  Action 1 :
  - Description : Simple NN Model
  - Status : Closed

  Action 2 (NN Group : ADAM)
  - Description : First Version Of the Pin algorithm
  - Status :Canceled

  Action 3 (NN Group : Muhammad)
  - Description : Custom NN Model
  - Status : On going

  Action 4 (NN Group : ADAM)
  - Description : Custom NN
  - Status :On going

- **CFD Group (Daniel James Rogers) :**
  Action 1 :
  - Description :Reviewing The Code to get fully automated code for data
  - Status :Closed

Action 2 :
- Description :Run the code For different Cases
- Status :Closed

Action 3 :
- Description :Correct the mesh Parameter to get better results
- Status :Closed

Action 4 :
- Description : Get results for Couette flow
- Status : On going

Action 5 :
- Description : Implementation of UDFs
- Status : On going

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**

Action 1 :
- Description : Data Preprocessing for the model
- Status :Closed

Action 2 :
- Description : Discovering Governing Equations For DNS cases (Re in 180 … 5200)
- Status :Closed

Action 3 :
- Description : Testing and Improving the equations obtained
- Status :On-Going

Action 4 :
- Description : Interpolation of RANS and DNS data
- Status : On going

- **SETC Group (Venceslas Bignon) :**

Action 1 :
- Description : Code to recover all the data format
- Status :Closed (could optimize it more)

Action 2 :
- Description : Help the other members with github
- Status :Closed

Action 3 :
- Description : Interpolation of RANS and DNS data using norms
- Status : Closed

- **CIDA Group (Hyejoon Lee) :**

Action 1 :
- Description : developing user define functions in C that acts as a bridge between PySINDy and MATLAB
- Status :On-Going

Action 2 :
- Description : developed some code for creating a class with data for "Terms in Reynolds Stress Transport Equation (RSTE)"
- Status :Closed

Action 3 :
- Description : Fixing compilation problems for UDFs
- Status : On going

Action 4 :
- Description : Implementation of customized libraries for PySINDy models

- Status : On going

Minutes accepted: 30 minutes (12:00:00-12:30:00)

3. Discussion on Following Tasks

- **NN Group (Muhammad Gillani / ADAM) :**
  Action 1 :
  - Description : Do more Research to increase the DNS data .

  Action 2 :
  - Description : Testing NN models on interpolated data

  Action 3 :
  - Description : Create NN model to solve k-omega equation

- **CFD Group (Daniel James Rogers) :**
  Action 1 :
  - Description :Reviewing The Code to get Better results

  Action 2 :
  - Description : Find user defined function in fluent and try the atomisation

  Action 3 :
  - Improve Couette flow results

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**
  Action 1 :
  - Description : Implement new features to get better equations that could be generalized

  Action 2 :
  - Description : Share and explain interpolated data files

  Action 2 :
  - Description : Generalize equations to couette flow

- **SETC Group (Venceslas Bignon) :**

  Action 1 :
  - Description : Testing Python libraries for interpolation

  Action 2 :
  - Description : Help the other members with github

  Action 3 :
  - Description : Work on User Defined Functions

- **CIDA Group (Hyejoon Lee) :**
  Action 1 :
  - Description : finish the UDF C code after obtaining the PySINDy equation

  Action 2 :
  - Description : Exploring new approaches to improve PySINDy model

Minutes accepted: 30 minutes (12:30:00-13:00:00)

4. Other business

- Discussion of the validity of the results [All members]

Minutes Accepted: 30 minutes (10:00:00-10:40:00)

- Review the approaches [All members]

Minutes Accepted: 40 minutes (10:30:00-11:10:00)

- Discuss the report format and work distribution

Minutes Accepted: 25 minutes (11:10:00-: 11:35:00)

5. Date of next meeting

Monday 01/04/2024

MSc CSTE Group Project Meeting Minutes
Monday 1st April 2024

1.

Present: Daniel James Rogers [DJR], Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] (Meeting Notes), and Yasser El Karkouri [YEK]

Present: CIDA Machine Learning Team [CIDAML], CIDA PySindy Team [CIDAPY], CFD Team [CFD], Software Team [S]

Absent: No-one
Apologies: No-one

2. Previous Tasks Review

- **NN Group (Muhammad Gillani / ADAM) :**
  Action 1 :
  - Description : Simple NN Model
  - Status : Closed

  Action 2 (NN Group : Muhammad)
  - Description : Custom NN Model with DNS data
  - Status : On going

  Action 3 (NN Group : ADAM)
  - Description : Custom NN Model
  - Status :On going

- **CFD Group (Daniel James Rogers) :**
  Action 1 :
  - Description : Create more data to test
  - Status : On going

  Action 2 :
  - Description : Analysis data for Couette flow
  - Status :Closed

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**
  Action 1 :
  - Description : Analysis data of new interpolated data
  - Status :Closed

  Action 2 :
  - Description : Testing and Improving the equations obtained
  - Status :On-Going

  Action 3:
  - Description : Interpolation of RANS and DNS data with Couette flow
  - Status : On going

- **SETC Group (Venceslas Bignon) :**

  Action 1 :
  - Description : Help the other members with github
  - Status : On going

  Action 2 :
  - Description : Fix an error of running Fluent on Virtual Machine
  - Status : Closed

- **CIDA Group (Hyejoon Lee) :**
  Action 1 :
  - Description : developing user define functions in C that acts as a bridge between PySINDy and MATLAB
  - Status :On-Going

  Action 2 :
  - Description : Fixing compilation problems for UDFs
  - Status : Closed

Minutes accepted: 30 minutes (12:00:00-12:30:00)

3. Discussion on Following Tasks

- **NN Group (Muhammad Gillani / ADAM) :**
  Action 1 :
  - Description : Write a report

  Action 2 :
  - Description : Testing NN models on interpolated data

  Action 3 :
  - Description : Improve NN / K prediction model to solve k-omega equation

- **CFD Group (Daniel James Rogers) :**
  Action 1 :
  - Description :Reviewing The Code to get Better results

  Action 2 :
  - Description : Find user defined function in fluent and try the atomisation

  Action 3 :
  - Improve Couette flow results

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**

Action 1 :
- Description : Write a report

Action 2 :
- Description : Share and explain interpolated data files

Action 3 :
- Description : Generalize equations to couette flow

Action 4 :
- Description : Generalize equations to couette flow

- **SETC Group (Venceslas Bignon) :**

Action 1 :
- Description : Produce more data for training and testing

Action 2 :
- Description : Help the other members with github

- **CIDA Group (Hyejoon Lee) :**

Action 1 :
- Description : Compile and simulate Fluent with UDF C code

Action 2 :
- Description : Writing a report

Minutes accepted: 30 minutes (12:30:00-13:00:00)

4. Other business

- Discussion of the validity of the results [All members]

Minutes Accepted: 30 minutes (10:00:00-10:30:00)

- Review the approaches [All members]

Minutes Accepted: 30 minutes (10:30:00-11:00:00)

- Discuss the report format and work distribution

Minutes Accepted: 10 minutes (11:00:00-: 11:10:00)

5. Date of next meeting

Monday 01/04/2024

MSc CSTE Group Project Meeting Minutes
Thursday 4st April 2024

1.

Present: Daniel James Rogers [DJR] (Meeting Notes), Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]

Present: CIDA Machine Learning Team [CIDAML], CIDA PySindy Team [CIDAPY], CFD Team [CFD], Software Team [S]

Absent: No-one
Apologies: No-one

2. Previous Tasks Review

- **CFD Group (Daniel James Rogers) :**
  Action 1 :
  - Export Couette RANS data
  - Status: Closed

  Action 2 :
  - Create Code for automatically executing Channel Flow with a rotated domain.
  - Status: Closed

  Action 3 :
  - Description: Improve script for rotating DNS data and validate with RANS data.
  - Status: Ongoing

- **NN Group (Muhammad Gillani / Adam Sohail) :**
  Action 1 (MG):
  - Description: Improved K Optimization algorithm with Machine Learning
  - Status: Closed

Action 2 (AS)
- Description: Improved version of the PINN NN for RST
- Status: Closed

Action 3 (MG and AS):
- Description: Writing Report Machine Learning Background and Methodology
- Status: On-Going

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**

Action 1 (YEK):
- Description: Mirroring DNS data to adequate RANS
- Status: Closed

Action 2 (YEK)
- Description: Enhance the RST PySindy optimization equations
- Status: Closed

Action 3 (GM):
- Description: Data Interpolation Scripts
- Status: Closed

Action 4 (GM)
- Description: Optimize K PySindy Equations
- Status: Closed

Action 5 (MG and AS):
- Description: Writing Py-Sindy Methodology
- Status: On-Going

- **SETC Group (Venceslas Bignon) :**

Action 1:
- Initial script to rotate the DNS data (w/ Daniel)
- Status: Closed

Action 2:
- Description: Set Fluent Machine Virtual Environment for UDF implementation (w/ Faye)
- Status: Closed

Action 3
- Description: Reflect on assembling all the parts of the group
- Status: On-Going

- **CIDA Group (Hyejoon Lee) :**

Action 1:
- Set Fluent Machine Virtual Environment for UDF implementation (w/ Vince)
- Status: Closed

Action 2:
- Description: Debug UDF Code
- Status: On-Going

Action 3
- Description: Write the PySindy Background part of the Report
- Status: On-Going

3. Discussion on Following Tasks

- **CFD Group (Daniel James Rogers) :**
  Action 1 :
    - Review the code for Rotating the Data

  Action 2 :
    - Explain DNS and RANS equations in the report.

  Action 3 :
    - Implement UDF in fluent simulations

- **NN Group (Muhammad Gillani / Adam Sohail) :**
  Action 1 (MG):
    - Review the ML code and improve results for K.

  Action 2 (AS)
    - Description: Optimize PINN Code.

  Action 3 (MG and AS):
    - Continue Writing the ML and NN part of the report

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**
  Action 1 (YEK):
    - Description: Improve RST Equations with PySIndy

  Action 3 (GM):
    - Description: Implement K equation in Boussines Hypothesis With RANS Data

  Action 5 (YEK and GM):
    - Description: Writing Py-Sindy Methodology
    - Status: On-Going

- **SETC Group (Venceslas Bignon) :**

  Action 1:
    - Write Data Analysis and Transformation Report

- **CIDA Group (Hyejoon Lee) :**
  Action 1:
    - Description: Debug UDF Code

  Action 3
    - Description: Write PySindy Background part of the Report

4. Other business

- Discuss whether it is worth it to Enhance the Data by rotating the domain.
- Discussion of features and variables used in PySIndy and Machine Learning
- Discussion of turbulent viscosity in DNS
- Discussion of the general overview of the report

5. Date of next meeting

Monday 08/04/2024 after the supervised meeting

MSc CSTE Group Project Meeting Minutes
Thursday 8th April 2024

1.

Present: Daniel James Rogers [DJR], Muhammad Gillani [MG], Adam Sohai [AS] (Meeting Notes), Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]

Present: CIDA Machine Learning Team [CIDAML], CIDA PySindy Team [CIDAPY], CFD Team [CFD], Software Team [S]

Absent: No-one
Apologies: No-one

2. Previous Tasks Review

3. Discussion on Following Tasks

- **CFD Group (Daniel James Rogers) :**
  Action 1 :
  - Improve RANS data by using Ryenolds Transport Equations

  Action 2 :
  - mplement RANS equations and future pysindy equations in BVP

  Action 3 :
  - Boundary Value Problem Tool box

- **NN Group (Muhammad Gillani / Adam Sohail) :**
  Action 1 (MG):
  - Start the poster presentation
  - Working the Machine Learning section of the report (presentation aspect)

Action 2 (AS)

- Enhance the PINN model by adding new equations as well as boundary conditions.
- Working on the PINN part of the Report (technical part)

Action 3 (MG and AS):

- Continue Writing the ML and NN part of the report

- **PySINDy Group (Yasser El Karkouri / Gabriel Maire) :**

  Action 1 (YEK and GM):

  - Improve covariance model using Tamas advice
  - Continue writing the report with the PySindy team

- **SETC Group (Venceslas Bignon) :**

  Action 1:

  - Continue writing the report on the interpolation, mirroring as well as data enhancement

- **CIDA Group (Hyejoon Lee) :**

  Action 1:

  - data augumentation, test robustness, validate with genetic algorithm(if possible), derive governing equation

  Action 2:

  - Work with the PySindy team to help get better result

4. Other business

- Discuss about how to split the work for the last two weeks (report, poster, and results from the teams).
- Discussion of getting equations using second order derivatives (PySindy)
- Discussion on getting new DNS data for Machine Learning Team
- Discussion of the general overview of the report

5. Date of next meeting

Thursday 11/04/2024

MSc CSTE Group Project Meeting Minutes
Thursday 11st April 2024

1.

Present: Daniel James Rogers [DJR] (Meeting Notes), Muhammad Gillani [MG], Adam Sohai [AS], Gabriel Maire [GM], Venceslas Bignon [VB], Hyejoon Lee [HL] and Yasser El Karkouri [YEK]

Present: CIDA Machine Learning Team [CIDAML], CIDA PySindy Team [CIDAPY], CFD Team [CFD], Software Team [S]

Absent: No-one
Apologies: No-one

2. Previous Tasks Review

- **Daniel James Rogers**
  Action 1 :
  - Launch RSTE in ANSYS unsuccessfully for the moment
  - Status : On going

  Action 2 :
  - Model the equation on MATLAB for the channel and Couette flow
  - Status : On going

- **Muhammad Gillani**
  Action 1 :
  - Focus on the poster
  - Status : On going

  Action 2 :
  - Report redaction support (ML, literature review)
  - Status : On going

- **Adam**
  Action 1 :

- Use different model for Couette and channel to generalize the pipeline
- Status : Closed

- **Gabriel Maire**
  Action 1 :
  - Integrating rans equation using pysindy
  - Status : On going

  Action 2 :
  - Data analysis
  - Status : On going

- **Yasser El Karkouri**
  Action 1 :
  - Refactor the interpolation function
  - Status : On going

- **Venceslas Bignon**
  Action 1 :
  - Create some test for data interpolation and data mirroring
  - Status : On going

  Action 2 :
  - Literature review on AI testing
  - Status : Closed

- **Hyejoon Lee**
  Action 1:
  - used a genetic algorithm to create a model and validate the PySINDy model
  - Status : On going

3. Discussion on Following Tasks

- **Daniel James Rogers:**
  Action 1 :
  - Same tasks

- **Adam:**
  Action 1 :
  - Plot all method to compare

  Action 2 :
  - Write on the report

- **Gabriel Maire :**
  Action 1 :
  - Work on the same with 2nd ODE

  Action 2 :
  - Write report

- **Muhammad Gillani:**
  Action 1 :
  - Discussion with the group to agree on what put on the poster

- **Hyejoon Lee:**

Action 1 :

- Continue on the previous task

- **Yasser El Karkouri:**

  Action 1 :

  - Interpolation couette flow

  Action 2 :

  - Improve model with 2nd order equation

- **Venceslas Bignon**

  Action 1 :

  - Keep working on the testing part

4. Other business

- Discussion on the report

Minutes Accepted: 10min (16h15 – 16h25)

- Reflection on variables

Minutes Accepted: 20min (16h30 – 16h50)

5. Date of next meeting

Monday 11/04/2024

	Strength	Weakness	Team Roles
Daniel James Rogers	Leadership & Direction	Conflict Resolution	**Chair** Self-confident, commands respect, good speaker, thinks positively, good at guiding the team. (Can be domineering, bossy.)
Syed Muhammad Gillani	Completing Documentation & Reports	Presenting ideas to customer, team, module leaders	**Finisher** Painstaking, conscientious, follows through and works hard to finish things properly. Meets deadlines and pays attention to detail. (Can be over-anxious and perfectionist.)
Adam Sohail	Suggesting ideas to the team	Planning &	**Innovator** Produces ideas, imaginative, unorthodox, radical, clever, uninhibited. (Can be over-sensitive, prickly. May need careful handling.)
Gabriel Maire	Presenting ideas to customer, team, module leaders	Organisation	**Evaluator** Careful, makes intelligent judgements, tests out ideas, evaluates proposals, helps the team avoid mistakes. (Can become isolated and aloof, pessimistic or over-critical.)
Yasser El Karkouri	Conflict Resolution	Leadership & Direction	**Teamworker** Sympathetic, understanding, sensitive, shows a strong concern for social interaction, leads from behind. Places the team above personal concerns. (May be indecisive.)
Hyejoon Lee	Helping the team to function (take notes, schedule meetings, book rooms)	Completing Tasks Efficiently	**Investigator** Finds things out, always knows someone who …, brings information back to the team, enthusiastic, gregarious. (Can be lazy and complacent.)
Venceslas Bignon	Programming, Planning & Organisation	Suggesting ideas to the team	**Organiser** Methodical, hard-working, reliable, orthodox, turns ideas into plans which are feasible and gets down to tasks which need doing. (Can be inflexible and uninspiring.)