DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, and Intelligence

Learning Fluid Dynamics Models that Exhibit Turbulent Behavior

Thana Guetet

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, and Intelligence

Learning Fluid Dynamics Models that Exhibit Turbulent Behavior

Lernen von Modellen für turbulente Strömung

Author:	Thana Guetet
Supervisor:	Prof. Dr. Hans-Joachim Bungartz
Advisor:	Dr. Felix Dietrich
Submission Date:	15.07.2022

I confirm that this master's thesis in robotics, cognition, and intelligence is my own work and I have documented all sources and material used.

Munich, 15.07.2022 Thana Guetet

Acknowledgments

This work would not have been possible without the support of many people. Many thanks to my advisor, Dr. Felix Dietrich, for the continuous feedback and the enriching discussions. Also thanks to my supervisor, Prof. Dr. Hans-Joachim Bungartz, for facilitating my work with the chair for Scientific Computing in Computer Science. I would like to also thank all the members of the SCCS chair and the audience that attended my talk presenting my thesis. And finally, thanks to my family and friends who supported me and showed interest in my progress throughout the entirety of my project.

Abstract

Turbulent flows are an important research topic in fluid dynamics, as they exhibit a complex behavior over a large range of spatial orders. Despite their complexity, turbulent flows are a common phenomenon in day to day life, which fuels the motivation behind studying, understanding, and approximating them. Partial differential equations (PDE) have been used successfully to describe a multitude of physical and engineering problems, turbulent flows being one example. The Navier-Stokes equations (NSE) are the PDE that exhibit turbulent flows. To approximate solutions of the NSE, traditionally, numerical solvers for turbulence models are used. Although, fluid flows are very accurately described by the NSE, many important cases in other disciplines are yet to be explained. In those cases, considering a data-driven approach that learns the equations directly from data observed from the phenomenon is an intriguing and interesting topic. Since numerical simulations have facilitated the generation and collection of PDE solutions, machine learning has been increasingly employed in problems related to partial differential equations. The application of machine learning for PDE has been split into data-driven solution of PDEs and the data-driven learning of PDEs. This thesis tackles the latter problem. As a non-turbulent toy example, PDE-Net is used to learn the differential operators and coefficients of the diffusion equation. Physics-informed neural networks are used to the learn the parameters of the parameterized NSE for laminar flows and flows that exhibit turbulent behavior. PDE-Net shows good results for simple diffusion examples. PINN can accurately capture the parameters for laminar flows, however, for turbulent flows the predicted parameters do not accurately approximate those invoked in the NSE.

Contents

Acronyms

$k-\omega$ (SST) $k-\omega$ Shear Stress Transport. 15

2D 2-dimensional space. 1

3D 3-dimensional space. 2

ANNs artificial neural networks. 18

CFD computational fluid dynamics. 1

DL deep learning. 2

FNO Fourier Neural Operator. 21

NSE Navier-Stokes Equations. 1

OpenFOAM Open Field Operation and Manipulation. 3, 13

PDE partial differential equation. 1

PDE-FIND PDE functional identification of nonlinear dynamics. 30

PINN physics-informed neural network. 2, 23

POD proper orthogonal decomposition. 31

Re Reynolds number. 5

SINDy sparse identification of nonlinear dynamics. 30, 62

UATs Universal Approximation Theorems. 18

VOF Volume of Fluid. 37

1 Introduction

Understanding the natural phenomena has always driven humans to innovation, invention, and searching for answers. As defined in Encyclopedia Britannica [82], Science, on the simplest level, is the knowledge of the world of nature. To acquire this knowledge, the first approach was observation and experimentation, however, for more complex problems theories and mathematical models were constructed to better understand these problems. With physics being the branch of science concerned with the study of properties and interactions of space, time, matter, and energy [6], fluid dynamics [23] emerged as a subdiscpline of fluid mechanics that describes the flow of fluids, liquids, and gasses. Fluid dynamics brought together applied mathematics, physics, engineering, and mechanics [18] to study real-world flows.
Traditionally, science and engineering was divided into an experimental and a theoretical part. The theory is based on theorems, assumptions, and postulation, where mathematics helps in quantification and understanding. Whereas experimental analysis is based on observations and measurements of experiments carried out in the real world. As these two previously mentioned approaches to science have their limitations and drawbacks a third pillar has emerged in the 1950*s*, modeling and simulation. This technique facilitates solving and predicting solutions to real world problems, unable to be solved by theoretical or experimental analysis [55]. A particular technique of simulation and modeling is computational fluid dynamics (CFD). This technique predicts the flow behavior by numerically solving the governing equations of fluid flows. Using numerical methods, the coupled partial differential equation (PDE) are transformed into an algebraic set of equations, which can be solved iteratively [55]. CFD has facilitated the study of fluid dynamics, in particular laminar and turbulent flows, which are witnessed at different scales from water falling from a tap to wind flow over mountains, or islands. The results obtained from CFD simulations are mostly reliable and in agreement with the observed real-world scenarios. As an example, figure 1.1 shows the von Kármán vortex street generated by the Rishiri Island in Japan and the flow behind a cylinder generated by a CFD software. The Navier-Stokes Equations (NSE), introduced in chapter 2, are a PDE that describes the motion of fluids based on the momentum conservation principle [20]. The three central points of every PDE are about existence, uniqueness, and smooth dependency on initial data. For the NSE, mathematically proven answers to these three points are available in 2-dimensional

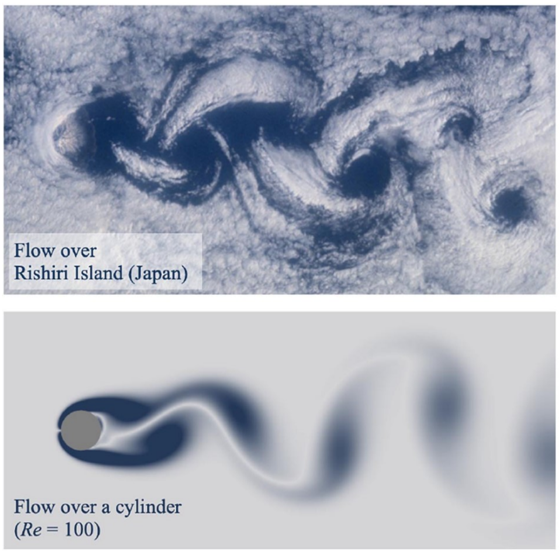


Figure 1.1: Top: Flow over the Rishiri island in Japan. Bottom: Flow over a cylinder

space (2D), in fact 2D-NSE with smooth initial data, has unique solutions, that stay smooth forever, which was proven in the 1960s. In 3-dimensional space (3D), only local existence and uniqueness are known. As a result, proving the existence and smoothness of solutions to the NSE in three-dimensional space is one of the seven Millennium Prize Problems [73].
In engineering, the NSE have been solved with help of turbulence models. Understanding turbulence modeling is at the center of this work, as it describes a very complex, yet common phenomenon, turbulent flows. Given the importance and complexity of turbulent flows, learning the models that describe them especially using reliable simulation data is the driving motivation behind this thesis, and which gives it an introductory aspect to future innovation. In fact, where the NSE can accurately describe turbulent flow, many other phenomena lack such a compact description, which conveys importance to the problem of learning equations from data observed in such phenomena. In this work, the focus is on numerically obtained solutions of the NSE in 2D and 3D spaces. Besides, simulating classical examples of fluid flows, another field that has gained an influx of attention over this decade, as a powerful data-driven method, is utilized in this thesis, namely, deep learning (DL). Deep learning has proven its powerful predictive and expressive benefits in a multitude of fields [36, 33, 35, 16]. Fluid dynamics, which are described by PDEs started benefiting from this ongoing success. Using the powerful expressive and predictive function approximation of Neural Networks, multiple studies used DL to solve the Partial Differential Equations, for example physics-informed neu-

ral network (PINN) [66, 29, 7, 62], where a physics-informed loss was used to ensure the predicted flow follows the physical laws. In [78, 13, 2, 21], mostly used for computer vision, Convolutional Neural Networks, are used to learn filters that approximate the flow at each pixel and should sum up to a low error. Gaussian Process regression [56] has be used for approximating the PDE solution . In a different approach , [43, 81, 44, 37, 38] learn the Neural Operator that learns the mapping of a function from the infinite dimensional space to another function in the infinite dimensional space, which is the solution space of PDE. Deep Learning can also be used to accelerate the predictions of CFD, as in [69, 32, 79]. A final application of Deep Learning in fluid dynamics is the data-driven discovery of differential equations. In [70], [11], and [40] use symbolic regression to find out the most fitting equation to the observed data from a pre-computed dictionary. While [4] use data-driven methods to learn the spatial derivatives of PDEs, [65] [63], and [42], focus on learning the parameters of the linear and nonlinear PDEs. Finally, PDE-Net [42, 40] uses convolutional filters to approximate differential operators and forms the PDE describing the given data.
This work will study the performance of PDE-Net [42] and PINN [65], when learning parametrized PDEs, as a semi-supervised problem with data simulated using the open-source CFD software Open Field Operation and Manipulation (OpenFOAM) [52]. The chapters are defined as follows, in 2, the NSE and turbulence models will be defined, as well as the general functioning of simulating in OpenFOAM [52]. Then, The state-of-the-art models, that have been used in fluid dynamics will be reviewed. In chapter 3 then the training process of both PDE-Net and PINN will described, the defined experiments will be presented, as well as the resulting parameters and learned CFD models. Finally, this thesis will be concluded in 4.

2 State of the art

This chapter introduces the NSE and turbulence models, it highlights the state-of-the-art methods used to infer the solutions of PDEs, as well as learn Partial Differential equations.

2.1 The Navier-Stokes Equations

Given below is the definition of a partial Differential Equation as it appears in [67].
Definition: For a function F that is sufficiently smooth with respect to variables **x**,t,u,$u_{x_i}, u_t, u_{x_i}x_j$,... such that at least one of the derivatives $F_{u_{x_i}}, F_{u_t}, F_{u_{x_i}x_j}, ...$ is nonzero over a suitable domain, then an equation of the form

$$F(\boldsymbol{x}, t, u, u_{x_i}, u_t, u_{x_i}x_j, ...) = 0, \text{ for } (\mathbf{x}, t) \in \Omega_1, \tag{2.1}$$

and is called the general partial differential equation for a function u=u(**x**,t) with $(\mathbf{x}, t) \in \Omega_1$. The order of 2.1 is the order of the highest derivative of u appearing in the equation.Besides their orders, PDEs can be classified based of their linearity type [67]:

- Linear equation: if the function F is linear with respect to any of the variables present in F $u, u_{x_i}, u_t, u_{x_i}x_j$,..., and the coefficient functions depend on the independent variables $x_1, ..., x_n$ and t only;
- Semilinear equation if the principle part of the equation is linear, and the coefficients are functions of (n+1)independent variables $x_1, ..., x_n$ and t;
- Quasilinear equation if the principle part of the equation is linear, the coefficients are functions of (n+1) independent variables and also of the dependent variable u(**x**,t), and at least one coefficient function appearing in lower order terms depends on (n+1) independent variables and also on some lower order derivative of the function u;
- Fully nonlinear equation if it is not a quasilinear equation

The 3D unsteady form of the Navier-Stokes Equations are shown, as derived independently by G.G. Stokes and M. Navier in the early 1800's [50]:

$$\begin{aligned}
\textbf{Continuity:}&\frac{\delta\rho}{\delta t}+\frac{\delta(\rho u)}{\delta x}+\frac{\delta(\rho v)}{\delta y}+\frac{\delta(\rho w)}{\delta z}=0\\
\textbf{X-Momentum:}&\frac{\delta(\rho u)}{\delta t}+\frac{\delta(\rho u^2)}{\delta x}+\frac{\delta(\rho uv)}{\delta y}+\frac{\delta(\rho uw)}{\delta z}=-\frac{\delta p}{\delta x}+\frac{1}{Re}\left[\frac{\delta\tau_{xx}}{\delta x}+\frac{\delta\tau_{xy}}{\delta y}+\frac{\delta\tau_{xz}}{\delta z}\right]\\
\textbf{Y-Momentum:}&\frac{\delta(\rho v)}{\delta t}+\frac{\delta(\rho uv)}{\delta x}+\frac{\delta(\rho v^2)}{\delta y}+\frac{\delta(\rho vw)}{\delta z}=-\frac{\delta p}{\delta y}+\frac{1}{Re}\left[\frac{\delta\tau_{xy}}{\delta x}+\frac{\delta\tau_{yy}}{\delta y}+\frac{\delta\tau_{yz}}{\delta z}\right]\\
\textbf{Z-Momentum:}&\frac{\delta(\rho w)}{\delta t}+\frac{\delta(\rho uw)}{\delta x}+\frac{\delta(\rho vw)}{\delta y}+\frac{\delta(\rho w^2)}{\delta z}=-\frac{\delta p}{\delta z}+\frac{1}{Re}\left[\frac{\delta\tau_{xz}}{\delta x}+\frac{\delta\tau_{yz}}{\delta y}+\frac{\delta\tau_{zz}}{\delta z}\right]\\
\textbf{Energy:}&\frac{\delta(E_t)}{\delta t}+\frac{\delta(uE_t)}{\delta x}+\frac{\delta(vE_t)}{\delta y}+\frac{\delta(wE_t)}{\delta z}=-\frac{\delta(up)}{\delta x}-\frac{\delta(vp)}{\delta y}-\frac{\delta(wp)}{\delta z}\\
&-\frac{1}{RePr}\left[\frac{\delta q_x}{\delta x}+\frac{\delta q_y}{\delta y}+\frac{\delta q_z}{\delta z}\right]\\
&+\frac{1}{Re}\left[\frac{\delta}{\delta x}(u\tau_{xx}+v\tau_{xy}+w\tau_{xz})+\frac{\delta}{\delta y}(u\tau_{xy}+v\tau_{yy}+w\tau_{yz})+\frac{\delta}{\delta z}(u\tau_{xz}+v\tau_{yz}+w\tau_{zz})\right],
\end{aligned}$$

where (x,y,z) are the coordinates, t is time, p is the pressure,
(u,v,w) are the velocity components, τ is the stress tensor,
Re the Reynolds number, Pr the Prandtl Number, and E_t is the total energy (2.2)

The Reynolds number (Re) "is the ratio of the scaling of the inertia of the flow to the viscous forces in the flow" [50]. The Prandtl number Pr "is the ratio of the viscous stresses to the thermal stresses" [50]. The stress tensor τ has 9 components, where each component is the second derivative of the velocity components [50]. The terms on the left side of the momentum equations are known as the convection terms. Convection describes the physical process that transports some property of the gas by the ordered motion of the flow [22, 50]. The tensor components multiplied by the inverse of the Reynolds number are the diffusion terms. "Diffusion is a physical process that occurs in a flow of gas in which some property is transported by the random motion of the molecules of the gas" [50]. Turbulence, and boundary layers (laminar, or turbulent) are the result of diffusion. Precedent to the formulation of the NSE, the Euler equations [19], were used to approximate the flow, however viscosity is neglected, which means that these equations cannot describe turbulent flows. The NSE describe the relation between the pressure, velocity, and density of a moving fluid, which makes the NSE a set of coupled equations. The independent parameters are the time and space coordinates, whereas pressure, velocity components, density ρ, and temperature T (contained in the energy equation) are all dependent. Due to all dependent variables appearing in

each equation, to solve the flow, all five equations must be solved simultaneously. The equation of state [50], that related pressure, density, and Temperature of a gas is also needed to solve the coupled system.
Due to the difficulty of solving these equations simplifications need to be introduced. In CFD, the stress tensor is approximated by turbulence models, which are introduced later in 2.2.3.
Another simplification is the assumption of incompressiblity of the fluid. In case of a low fluid velocity, the fluid can be considered incompressible and its density ρ to be constant $\frac{\delta\rho}{\delta t} = 0$. In this case and for an incompressible single-phase fluid, the flow motion equations can be written in tensor form [23] as:

$$\begin{aligned}
&\text{Continuity equation:}\\
&\nabla v = 0\\
&\text{The stress tensor for incompressible viscous fluids:}\\
&\tau = \mu(\nabla v + \nabla v^T)\\
&\nabla \cdot \tau = \mu \nabla^2 v\\
&\text{Momentum balance equation:}\\
&\rho\frac{\delta v}{\delta t} + \rho(v.\nabla)v = -\nabla p + \mu\nabla^2 v\\
&\text{momentum equation divided by the constant } \rho\text{:}\\
&\frac{\delta v}{\delta t} + (v.\nabla)v = -\frac{1}{\rho}\nabla p + \nu\nabla^2 v,\\
&\text{where}\frac{\delta\rho v}{\delta t} + \rho(v.\nabla)v \text{ is the inertia, } \nabla p \text{ is the pressure gradient,}\\
&\mu\nabla^2 v \text{ is the viscous force, } \mu \text{ the dynamic viscosity,}\\
&\text{and } \nu = \frac{\mu}{\rho} \text{ is the kinematic viscosity}
\end{aligned} \tag{2.3}$$

The Reynolds number $Re = \frac{UL}{\nu}$ is a dimensionless number that helps classify the regimes of flow. Here U is the velocity scale, L is the length scale, and ν is the kinematic viscosity. U and L are related to the geometry of the flow. If Re<1, the inertia effects are negligible in front of the viscous forces and Navier-Stokes 2.3 becomes the Stokes equation. The effect of increasing the Reynolds number on the flow over a cylinder is shown in 2.1. The resulting regimes shown in 2.1 are:

- Re<5: Laminar Attached Steady.
- 5<Re<40: Laminar Separated Steady.

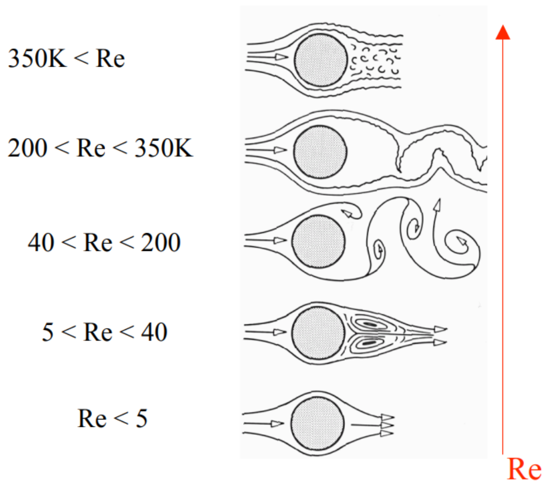


Figure 2.1: Experimental observations on the effect of increasing the Reynolds number Re on the flow over a cylinder [74].

- 40<Re<200: Laminar Separated Periodic.
- 200<Re<350K: Laminar Separation/Turbulent Wake Periodic.
- 350K<Re: Turbulent Separation Chaotic.

The laminar flow follows the object shape. Whereas, the turbulent flow besides being dominated by the object shape and dimension, it exhibits motion and evolution of small eddies [74]. Turbulent flow is challenging to compute due to its unsteady aperiodic motion, the fluid properties exhibit random spatial variations in 3D, and the flow depends strongly from the initial conditions and contains a wide range of scales (eddies). This implicates that the turbulent simulation must be always three-dimensional and time accurate with extremely fine grids. In the next section the problem of computing turbulent flow will be tackled from the CFD point of view.

2.2 Turbulence Modeling with Computational Fluid Dynamics (CFD)

As mentioned in the introduction, CFD has become a very relevant technique to understand and model fluid flow [55]. As PDEs cannot be solved analytically expect in special cases, numerical methods facilitate approximating a solution, by using a dicretization method, which approximates the PDE by a system of algebraic equations

Table 2.1: The characteristics of turbulence.

High Reynolds number	$Re \gg 1$.
Rotationality	Strong three-dimensional vortex generation mechanism.
Dissipation	The turbulent kinetic energy is transformed into internal energy by shear stress.
Diffusivity	The diffusivity of turbulence causes rapid mixing and increased rates of momentum, heat, and mass transfer.
Irregularity	Turbulence is not random since there exits a correlation between the velocities measured in two nearby points. Yet the movement of fluid particles is chaotic.

that can be solved iteratively by a computer [20]. One possible application is the simulation of turbulent flow.

2.2.1 What is turbulence?

Turbulence is an extremely complex concept, that was described as earliest as 500 years ago by Leonardo da Vinci which he depicted in many sketches. He describes the observed flow of water:*"...the smallest eddies are almost numberless, and large things are rotated only by large eddies and not by small ones and small things are turned by small eddies and large"* [31]. Multiple definitions of turbulence have been suggested over the years. In more modern definitions, S.Rodriguez linked turbulence to the use of approximations to provide solutions *"Turbulent flows is the dynamic superposition of an extremely large number of eddies with random (irregular) but continuous spectrum of sizes and velocities that are interspersed with small, discrete pockets of laminar flow (as a result of the Kolmogorov eddies that decayed, as well as in the viscous laminar sublayer and in the intermittent boundary). In this sense, turbulent flows are intractable in its fullest manifestation; this is where good, engineering common sense and approximations can deliver reasonable solutions, albeit approximate."* [68]
The characteristics of turbulence are explained in table 2.1 [51]. There are two types of turbulence structures: the macrostructure, which refers to the largest turbulence dimensions and depends on the length scale L and the velocity scale U, and its counterpart the microstructure.
For very large Reynolds numbers the nonlinear processes dominate the macrostructure, and the viscous effects are negligeable. At the limit where $Re \to \infty$, the macrostructure is independent from the Reynolds number. In this case, the large-scale structure is also independent from the fluid, given that the fluid is parameterized by its kinematic viscosity ν, which appears in the Reynolds number. This principle summarizes the

"Reynolds similarity: Turbulence is a property of the flow, not the fluid." [51].

2.2.2 Simulating Turbulent Flows

Here, the light is shed on two of the most popular numerical approaches to solving the NSE [20] for laminar and turbulent flows:

- **Direct Numerical Simualtion (DNS):** No modeling is required in this type of simulations. In such simulations all the scales of motions are resolved. However, DNS is only possible for low/moderate Reynolds number flows plus simple geometries (high number of grid points is needed).
- **Simulation of turbulence with Models:** the Navier-Stokes equations can be averaged, in time or in space. This averaging leads to the mean equations containing non linear correlations involving the unknown fluctuating variables. We speak of a closure problem. In this case, models are used to approximate expressions, or equations for the unknown correlations.
  - **Large Eddy Simulations (LES):** Here the NSE equations are filtered over space, which eliminates the need to compute small scales. Large eddies are resolved and small eddies are modeled. Simulations are 3D and unsteady. Here all variables are filtered in space and the sub-grid scale stress tensor τ^{SGS} needs to be modeled.
  - **Reynolds-Averaged Navier-Stokes equations (RANS):** Here the averaging is over time [20], describe by the equation 2.9 .All turbulence scales are modeled and simulations can be 2D or 3D with a steady or unsteady flow.

DNS simulations are correlated with high computational costs, besides that the detailed information delivered by DNS simulations is far more than the engineering requirements, which makes it unsuitable as a design tool. LES and RANS both lead to additional terms (closure terms) in the governing equations that must be modeled. Consequently, turbulence models equations are not the exact NSE since additional equations to close the system are added. In the following, the focus will be on RANS simulation as they are widely used for computational predictions of industrial flows [20]. Following is the process of obtaining the averaged NSE. Reynolds-averaged quantities are defined as:

$$u_i(x_k,t) = \overline{u_i}(x_k) + u_i'(x_k,t) \tag{2.4}$$

$$\overline{u_i}(x_k) = \lim_{T\to\infty} \frac{1}{T}\int_0^T u_i(x_k,t)dt, \tag{2.5}$$

where T is the averaging interval. This interval must be large compared to the typical time scale of the fluctuations. The mean of the fluctuating term $\overline{u_i'} = 0$. The average of two chosen variables $\overline{u_i\phi}$ results in the following expression:

$$\overline{u_i\phi} = \overline{(\overline{u_i} + u_i')(\overline{\phi} + \phi')} = \overline{u_i}\overline{\phi} + \overline{u_i}\overline{\phi'} + \overline{u_i'}\overline{\phi} + \overline{u_i'\phi'} \tag{2.6}$$

fluctuations have a zero average which leads to:

$$\overline{u_i}\overline{\phi'} = \overline{u_i'}\overline{\phi} = 0 \tag{2.7}$$

$$\overline{u_i\phi} = \overline{u_i}\overline{\phi} + \overline{u_i'\phi'}. \tag{2.8}$$

The latter term of the last equation 2.8 $\overline{u_i'\phi'}$ involving the two fluctuating terms is zero only if the two quantities are uncorrelated, which is not the case for turbulent flows. These terms are called the Reynolds stresses. The tensor notation of the continuity and momentum equations is:

$$\delta\frac{(\rho\overline{u_i})}{\delta x_i} = 0 \tag{2.9}$$

$$\frac{(\rho\overline{u_i})}{\delta t} + \frac{\delta}{\delta x_j}(\rho\overline{u_i}\overline{u_j} + \rho\overline{u_i'u_j'}) = -\frac{\delta\overline{P}}{\delta x_i} + \frac{\delta\overline{\tau_{ij}}}{\delta x_j} \tag{2.10}$$

$\overline{\tau_{ij}}$ are the mean viscous stress tensor components:

$$\overline{\tau_{ij}} = \mu(\frac{\delta\overline{u_i}}{\delta x_j} + \frac{\delta\overline{u_j}}{\delta x_i}). \tag{2.11}$$

2.2.3 RANS Turbulence Modeling

To solve the closure problem, the Reynolds stresses need to be expressed in terms of the known averaged quantities. One approach is to represent the turbulence as an increased viscosity [20]. This leads to the eddy-viscosity model for the Reynolds stresses, where the Reynolds stresses are described by the equation:

$$-\rho\overline{u_i'u_j'} = \mu_t(\frac{\delta\overline{u_i}}{\delta x_j} + \frac{\delta\overline{u_j}}{\delta x_i}) - \frac{2}{3}\rho\delta_{ij}k, \tag{2.12}$$

where k is the turbulent kinetic energy modeled as:

$$k = \frac{1}{2}\overline{u_i'u_i'} = \frac{1}{2}(\overline{u_x'u_x'} + \overline{u_y'u_y'} + \overline{u_z'u_z'}), \tag{2.13}$$

where μ_t is the turbulent viscosity. This model reduces the number of unknowns from 6 (every components of the Reynolds stress tensor) to the turbulent viscosity. According to [20], the eddy-viscosity hypothesis is not correct in detail, however due to its easy implementation, and the good results it provides when it is carefully applied, it is widely used. In a simplistic way, turbulence can be characterized by its kinetic energy k, or a velocity $q = \sqrt{2k}$, and a length scale L. A dimensional analysis determines μ_t as [20]:

$$\mu_t = C_\mu \rho q L, \tag{2.14}$$

where C_μ is a dimensionless constant whose value is usually 0.09 [52]. To define the eddy viscosity μ_t, the velocity scale and the length scale should be defined. A model for the turbulent kinetic energy k is defined by [20]:

$$\frac{\delta \rho k}{\delta t} + \frac{\delta(\rho \overline{u_j} k)}{x_j} =$$
$$\frac{\delta}{\delta x_j}(\mu \frac{\delta k}{\delta x_j}) - \frac{\delta}{\delta x_j}(\frac{\rho}{2}\overline{u'_j u'_i u'_i} + \overline{p' u'_j}) - \rho \overline{u'_i u'_j}\frac{\delta \overline{u_i}}{\delta x_j} - \mu \overline{\frac{\delta u'_i}{\delta x_k}\frac{\delta u'_i}{\delta x_k}}.$$

In the latter equation derived in 2.2.3, the terms on the left-hand side of the equation and the first term of the right-hand side need no modeling. The last term on the right-hand side of the equation represent the product of the density ρ and the dissipation ε, defined as "the rate at which turbulence energy is irreversibly converted into internal energy " [20]. The second term on the right-hand side is the turbulent diffusion of kinetic energy, which almost always modeled by use of a gradient-diffusion equation:

$$-(\frac{\rho}{2}\overline{u'_j u'_i u'_i} + \overline{p' u'_j}) \approx \frac{\mu_t}{\sigma_k}\frac{\delta k}{\delta x_j}. \tag{2.15}$$

The main weakness of the eddy viscosity is that it is a scalar, which limits its ability to represent general turbulent processes. This is mitigated by other models that are out of the scope of this thesis. The third term on the right-hand side of equation 2.2.3 is the rate of production of turbulent kinetic energy by the mean flow:

$$P_k = -\rho \overline{u'_i u'_j}\frac{\delta \overline{u_i}}{\delta x_j} \approx \mu_t(\frac{\delta \overline{u_i}}{\delta x_j} + \frac{\delta \overline{u_j}}{\delta x_i})\frac{\delta \overline{u_i}}{\delta x_j}. \tag{2.16}$$

After applying all the above mentioned modeling, the evolution of the turbulent kinetic energy can be calculated. However, the length scale is still undetermined, this can be done by the following observation [20]:

$$\varepsilon \approx \frac{k^{3/2}}{L}, \tag{2.17}$$

where ε is the dissipation, k is the turbulent kinetic energy and L is the length scale. A detailled formula for the dissipation and the intuition behind the mentioned relation

Table 2.2: RANS and LES turbulence models

Number of equations	**Examples of turbulence models**
	RANS
Zero-equation/algebraic models	Mixing length, Cebeci-Smith, Baldwin-Lomax
One-equation models	Wolfstein, Baldwin-Barth, Spalart-Allmaras, k-model
Two-equation models	k-ε, k-ω, k-τ, k-L
three-equation model	k-ε-A
four-equation model	v^2-f model
	LES
Zero-equation/ algebraic models	Smagorinsky, WALE, Germano dynamic model, Algebraic WMLES S-Omega Model Formulation

can be found in [20].
A formula for the eddy viscosity μ_t depending of all computed values is:

$$\mu_t = \rho C_\mu \sqrt{k} L \approx \rho C_\mu \frac{k^2}{\varepsilon}. \tag{2.18}$$

This specific model is referred to as the $k - \varepsilon$ model as it adds two equations to the RANS equations one for k and one for ε to fully determine the eddy viscosity and Reynolds stresses. One change affects the momentum equation, that is the dynamic viscosity μ becomes $\mu_{eff} = \mu + \mu_t$.
The $k - \varepsilon$ model is one of 200 eddy viscosity models which are classified in terms of the number of transport equations solved in addition to the RANS equations. Table 2.2 summarizes the typically used RANS models and some examples of LES turbulence models [74]. The LES models differ in the filter function used to eliminate the very small turbulence scales, while for the two-equation RANS models,for instance, the main difference is behavior near the walls of the computational domain.

Turbulence modeling is a very complex, yet very important field. As a matter of fact, most natural and engineering flows are turbulent, which justifies the necessity of turbulence modeling. The goal of turbulence modeling is to develop equations that predict the time averaged velocity, pressure, and temperature fields without calculating the complete turbulent flow pattern as a function of time. Simulating turbulent flows in any general CFD solver requires selecting a suitable turbulence model, defining initial and boundary conditions for the closure equations of the turbulent model, selecting a near-wall modeling treatment, and choosing runtime parameters. A concrete example of CFD is used to simulate turbulent flows is presented in the next subsection.

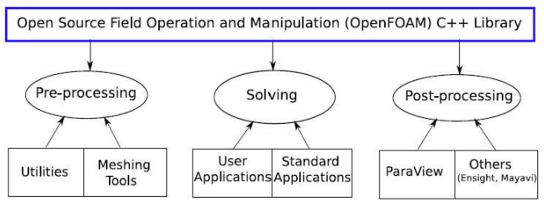


Figure 2.2: The classification of the utilities and applications offered by OpenFOAM [52].

2.2.4 CFD software: OpenFOAM

OpenFOAM [52] is an open-source CFD software released in 2004. It is foremost a C++ library, used to create excecutables, know as applications. The applications are divided into solvers, " designed to solve a specific problem in continuum mechanics" [52] , and utilities that performs tasks on data manipulation. The utilities are used for pre- and post-processing tasks, the user can define their customized applications as well. A summary of the OpenFOAM utilities and their classifications can be seen in Figure 2.2 Setting up a simulation case in OpenFOAM requires modifying the dictionary files in the case folder, as an integrated graphical user interface (GUI) for pre-processing, solver setting and monitoring the simulations does not exist. The general structure of a case setup in OpenFOAM is presented in the figure 2.3.
In directory *0*, the initial and boundary conditions of the fields are defined. As in incompressible flows, the pressure itself is a diagnostic variable, and has no physical meaning [61], its boundary conditions are set only to satisfy the continuity equations. Most of the simulations are conducted in control volumes that have a single inlet and a single outlet, the boundary condition of the pressure at the outlet is mostly defined by a fixed value in time, generally equal to the atmospheric pressure (in the order of $1e5Pa$) or equal to 0 Pa. At the inlet and walls, in the case of this work, the *fixedFluxPressure* boundary is applied, which "sets the pressure to the provided value such that the flux on the boundary is that specified by the velocity boundary condition" [52]. For the velocity, the inlet boundary condition is defined by a fixed value in time, and at the outlet boundary a *zeroGradient* boundary is chosen, which sets the boundary value to a constant in space. For the walls, a *slip* (Neumann boundary) or a *no slip* (Dirichlet boundary) boundary condition is applied for fixed walls, or a *movingWallVelocity* boundary condition, which corrects the flux $\phi = U.S_f$ due to mesh motion so that the total flux through the moving wall is zero [52]. The *fvSchemes* dictionary defines the discretization schemes, in particular the time marching and the convections schemes. The equation solvers, tolerance and algorithm controls (SIMPLE, PISO) are specified

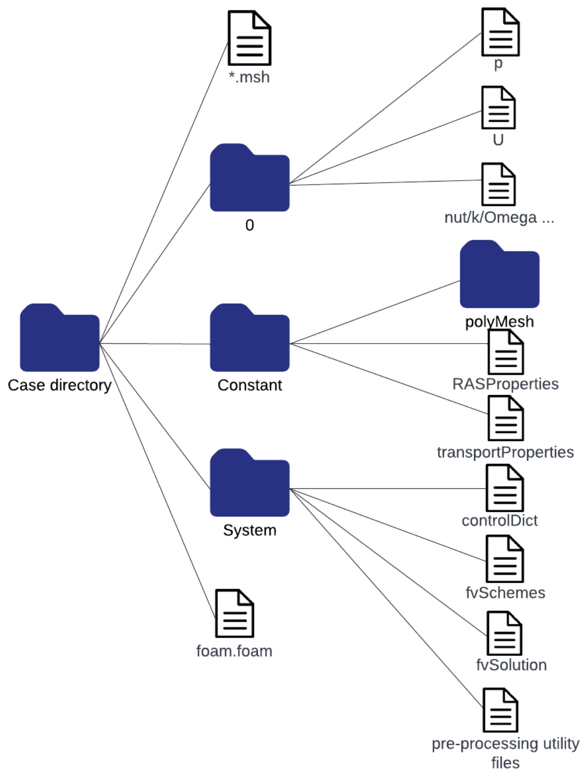


Figure 2.3: Schematic of the main directories and files in an OpenFOAM case setup.

in the *fvSolution* dictionary in the *system* directory. The solvers in question here are the linear-solvers used for each discretised equation. A linear-solver is "a method of number-crunching to solve a matrix equation" [52]. An application, *simpleFoam, pimpleFoam* is a solver that describes the entire set of equations and algorithms to solve a particular problem, this solver is specified in the *controlDict* dictionary, along side with the start and end times. As for the choice of the turbulence model, defined in the *Rasproperties* dictionary in the *constant* directory, $k-\omega$ is the turbulence model suitable for studying near wall behavior, and $k-\epsilon$ is suited for flow away from the wall. A model that facilitates the study of flow behavior near and far from the wall boundaries is $k-\omega$ Shear Stress Transport ($k-\omega$ (SST)) [52], which is the turbulence model employed in the experiments of this thesis. $k-\omega$ SST introduces two closure equations to model the turbulence kinetic energy, k, and the turbulence specific dissipation rate ω. The model equations and initialisation are defined later in section 3.2.3. To define the wall boundary conditions for the turbulence model's variables there exist three options depending on the range of the wall distance y^+:

- use wall functions ($30<y^+<300$)
- use insensitive wall functions ($1<y^+<300$)
- resolve boundary layer ($y^+<6$)

To compute the wall distance units $y^+ = \frac{\rho \times U_\tau \times y}{\mu} = \frac{U_\tau \times y}{\nu}$. y is the distance to the first cell center normal to the wall, and $U_\tau = \sqrt{\frac{\tau_\omega}{\rho}}$ is the friction, where τ_ω is the wall shear stresses [74].
Typical wall functions boundary conditions for the state variables of the turbulence model equations are summarized in table 2.3 [75]. The pre-processing utility *blockMesh* is the first command ran when running a simulation. It generates parametric meshes with grading and curved edges. The geometry is always defined in 3D since OpenFOAM only considers 3D geometries. First the vertices are defined then lists of 8 vertices are grouped into hexahedral blocks, and the number of cells in x, y, and z directions are defined for each block. *blockMesh* characterizes the different boundary layers as well.

After setting the initial and boundary conditions, and running the pre-processing applications, the central part of simulating is introduced next: the solver. The Navier-Stokes equations are solved in a sequential manner using predictor-corrector projection algorithms. The pressure-velocity coupling is handled numerically by different solution strategies. Two algorithms are highlighted here: the SIMPLE and PISO algorithm.
The SIMPLE algorithm The SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) allows to couple the Navier-Stokes equations with an iterative procedure, which can be summed up in the algorithm 1 [77].

Table 2.3: Typical wall functions for the fields of turbulence models.

Field	Wall functions-High RE	Resolved BL-Low RE
nut	nut(-)WallFunction or nutUSpaldingWall-Function	nutUSpaldingWallFunction, nutk-WallFunction, nutUWallFunction, nutLowReWallFunction or fixed-Value
k, q, R	kqRWallFunction $k_{wall} = k$	kqRWallFunction or kLowReWall-Function
epsilon	epsilonWallFunction $\epsilon_{wall} = \epsilon$	epsilonWallFunction (with inlet value) or zeroGradient or fixed-Value (with 0 or a small number)
omega	omegaWallFunction $\omega_{wall} = 10\frac{6\nu}{\beta y^2}, \beta = 0.075$	omegaWallFunction or fixedValue (both with a large number)
nutTilda	-	fixedValue (one to ten times the molecular viscosity, a small number, or 0)

Algorithm 1 SIMPLE algorithm

```
1: while t<= End time step do
2:     Initialise u and p using latest available values
3:     Construct the momentum equations
4:     Under-relax the momentum equation
5:     Solve the momentum equation to obtain an approximation for u
6:     Construct the pressure equation
7:     Solve the pressure equation for p
8:     Correct the flux for $\phi$
9:     Under-relax p
10:    Correct the velocity for u
11:    if $converged = False$ then
12:        Continue
13:    else
14:        Stop and print results
15:    end if
16: end while
```

Table 2.4: Behavior of the problems solved by the algorithms over time

algorithm	transient	steady-state
PISO/PIMPLE	YES	YES
SIMPLE	NO	YES

The PISO algorithm The PISO (Pressure Implicit with Splitting of Operators) [76] is used for solving the Navier-Stokes equations in unsteady problems. The main difference from the SIMPLE algorithm are:

- No under-relaxation is applied.
- The momentum corrector step is performed more than once.

Table 2.4 shows that the SIMPLE algorithm, and subsequently the OpenFOAM solver based on it *simpleFoam*, solves problems that are constant with respect to time. PIMPLE algorithm, however, solves unsteady problems that achieve a steady-state as well. For the case of unsteady or transient incompressible flow the *icoFoam, pimpleFoam, pisoFoam* solvers are suitable. Given that turbulent flows are usually unsteady the *pimpleFoam, pisoFoam* solvers are suitable here, as differently from *icoFoam*, they deal with turbulent flows.

Since PDEs describe many complex science end engineering problems, many approaches have been adapted to solve them. Often times a fine discretization is needed to capture the phenomena being modeled, which leads traditional numerical solvers to be slow [37]. Especially in the domain of engineering, design leads to the necessity of repeatedly testing different parameters and solving the corresponding equations with use of numerical solvers, which leads to very time consuming experiments and high computational expenses. These limitations motivated the work on faster methods especially with the use of machine learning that can be trained offline, then used to give flow predictions fast in real-time. Data-driven methods used for partial differential equations can be employed either for the task of solving the PDE or the task of discovery of the PDE. The upcoming sections will highlight the state of the art methods for both tasks.

2.3 Data-Driven Methods to infer the solutions of the Partial Differential Equations

There has been multiple works on solving partial differential equations using data-driven methods. Mesh-dependent methods such as CNNs [24] interpret the input initial and boundary conditions and the output flow as images. During training, the network re-learns the solutions and can then be tested on unseen scenarios of boundary and initial conditions to verify its generalization ability. Other approaches like Physics-informed Neural Networks [48], introduce a physics-informed loss that ensures that the governing equations are respected by the NN's predictions. Newer methods have tackled the problem of implicitly learning the solution operator. Two prime examples of this method are introduced: DeepONet and Fourier Neural Operator.

2.3.1 DeepONets

In this paper [43], the authors discuss the efficiency of relying on classical calculus and partial differential equations to represent new and emergent fields (e.g., social dynamics), due to their slow development. As an alternative, framework that facilitates learning dynamical systems. The new framework, named DeepONets, rely on the universal approximation theorem for operator [14]. The widely used theorem for neural network is the universal approximation theorem for functions. Next, both theorems are introduced.

universal approximation theorem for functions

In the mathematical theory of artificial neural networks (ANNs), Universal Approximation Theorems (UATs) establish the density of a class of NNs within a space of mappings. Per this definition, one can derive that NNs represent a variety of mappings in the presence of appropriate weights and biases [84]. A NN can be characterized by its activation function (e.g., ReLU, sigmoid, etc.), its width (the number of neurons per layer), its depth (the number of layers), and its connectivity (e.g., feedforward or recurrent) [84]. In 1969, Minsky and Papert demonstrated that a two-layer perceptron (input and output layers) cannot approximate functions outside of some special cases (e.g AND, OR operators) [26]. What followed is the formulation of the universal approximation theorem for functions, which states that the standard multilayer feed-forward networks with a single hidden layer that containing a finite number of hidden neurons, and utilizing an arbitrary activation function are able of approximating continuous or other kinds of functions defined on compact sets in R^n [26].

Definition of a function, functional, and operator

Functions are the mapping from from ${R^d}_1 \rightarrow {R^d}_2$. Meaning that image classification, segmentation, and regression, etc., used for the different applications in research and industry, can be understood as function approximation that rely on the universal approximation theorem for functions. Functionals are a further generalization of functions. They map from a $\infty - dim$ function $\rightarrow R^d$ Finally, Operators represent the mapping from an $\infty - dim$ function to another $\infty - dim$ function. Examples are the derivative and integral operators, dynamic systems, etc. [43].

Universal Approximation Theorem for Operator

In 1995, Chen and Chen introduced the Universal Approximation Theorem for Operator [14]. The theorem as presented in [43] reads as follows: Suppose that σ is a continuous non-polynomial function, X is a Banach Space, $K_1 \subset X, K_2 \subset R^d$, are two compact sets in X and R^d, respectively, V is a compact set in C(K_1), G is a nonlinear continuous operator, which maps V into C(K_2), then for any $\varepsilon > 0$, there are positive integers n,p,m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in R, w_k \in R^d, x_j \in K_1, i = 1, ..., n, k = 1, ..., p, j = 1, ..., m$, such that:

$$|G(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k) \sigma(w_k.y + \zeta_k)| < \varepsilon \tag{2.19}$$

holds for all $u \in V$ and $y \in K_2$. The universal approximation theorem ensures the possibility of learning linear and nonlinear operators using NNs.
The figure 2.4 shows the general architecture of DeepONet and the constraints made on the inputs. Besides the usual factors that affect the efficiency and speed of training for NNs such as the optimization parameters and network size, here depending the complexity of the space of the input functions, the number of sensors defined to sample these input function is problem dependent. For higher dimensional problems, y the output sampling locations is a vector with d components, where m the number of sensors locations for the input functions u is different from the dimension of the output locations y, which calls to using two separate networks to handle each of $[u(x_1), u(x_2), .., u(x_m)]$ and y. Due to the non-specific structure of the inputs, using convolutional neural networks does not make the favorable choice, the authors opt to using feed forward neural networks as a baseline [43]. The Universal Approximation Theorem for Operator, proves the existense of a NN with one hidden layer that approximates the operator, DeepONet, however, uses multiple hidden layers, as known from other deep learning models (CNNs, RNNs .etc.), which boosts the expressive power of the NNs.
The architecture of DeepONet is composed of two NNs as mentioned before due to the different dimensions of the sensor locations x and the output locations y. This upcoming paragraph describes the different implementations of DeepONet. **Stacked DeepONet:**
In this case, the authors utilize the previous equation 2.19 to determine the architecture of DeepONets as follows:

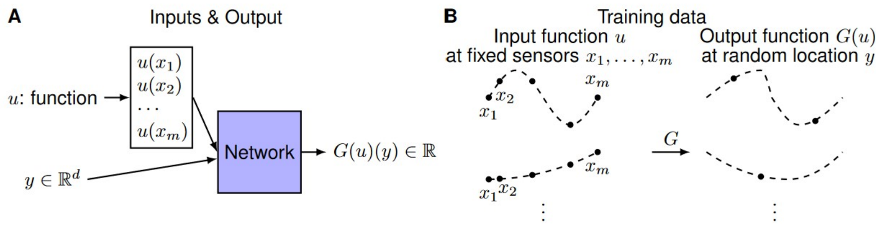


Figure 2.4: (A) For DeepONet to learn the Operator G: u $\rightarrow$ G(u) it takes the input functions u evaluated at a fixed number of sensors $[u(x_1), u(x_2), .., u(x_m)]$ and random locations y. (B) shows the training data with the input functions u evaluated at the same $x_1, x_2, ...x_m$ sensors, while the number and location of y is random [43].

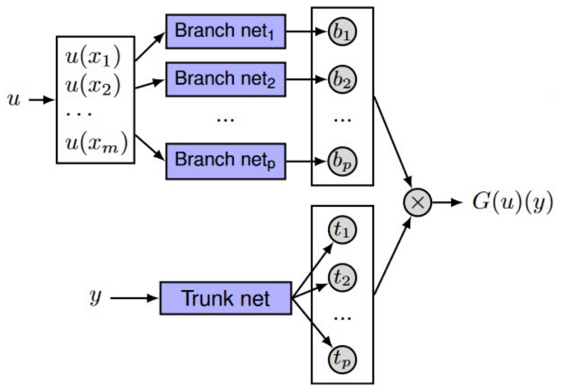


Figure 2.5: Stacked DeepONet: p stacked branch networks and one trunk network [43].

- Branch network:

$$\sum_{i=1}^{n} c_i^k \sigma(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k), \tag{2.20}$$

- Trunk network:

$$\sigma(w_k.y + \zeta_k), \tag{2.21}$$

where p branch networks are stacked. p being the number of summations of the multiplication of the two activation functions in 2.19. A schematic representation of the stacked DeepONet is given by the figure below 2.5.
Unstacked DeepONet:
In this case, one trunk network and one branch network constitute the DeepONet. The trunk network is the same as in the stacked DeepONet2.3.1. However, the p branch

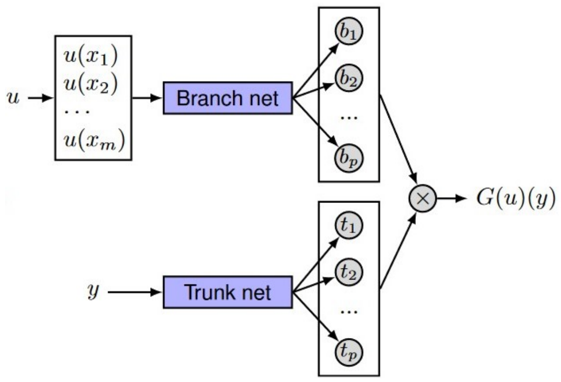


Figure 2.6: Unstacked DeepONet: one branch network and one trunk network [43].

networks are merged into one network.A schematic of the unstacked DeepONet is seen in the figure below 2.6. Both stacked and unstacked DeepONet lead to $G(u)(y) \approx \sum_{k=1}^{p} b_k t_k + b_0$, where b_k is the output of the k-th branch net or the k-th output of the branch net (for the unstacked DeepONet) and t_k is the k-th output of the trunk network. Different experiments in [43] demonstrated that DeepONets can learn various explicit operators (e.g integrals, fractional Laplacians etc.) as well as implicit operators in the form of deterministic ordinary and statistical partial differential equations. To simplify the operator learning task prior knowledge can be made use of as well which results in Physics-informed-DeepONets[81, 80].
Similar to DeepONet, however for the specific case of Partial Differential Equations, Fourier Neural Operator (FNO) [37] was introduced based on Graph Kernel Networks [39] to learn the mapping from any functional parametric dependence to the solution of the PDE.

2.3.2 Fourier Neural Operator

The Fourier Neural Operator builds on the Graph Kernel Network, which is a class of Neural Operators that represents the infinite-dimensional mapping by composing non-linear activation functions and a class of integral operators with the kernel integration computed by message passing on graph networks [45].In [83], it has been proven that the Graph Kernel Network is unstable when the number of hidden layers is increased. In the case of the Fourier Neural Operator, the operator is learned by parameterizing the integral kernel directly in Fourier space.

The Fourier Layer starts from the input v, then applies a Fourier transform F. It then applies a a linear transform on the lower Fourier modes and filters out the higher modes.Finally, the inverse Fourier Transform F^{-1} is applied and summed with the

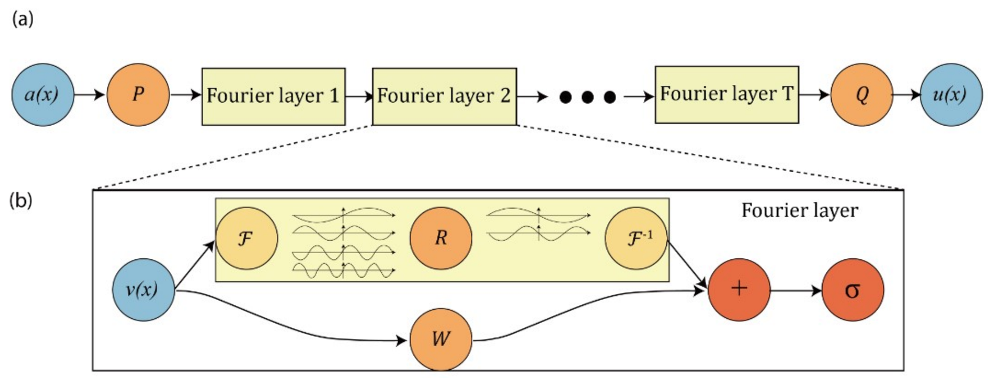


Figure 2.7: Fourier Neural Operator: (a) The architecture of the Fourier Neural Operator, (b) Fourier Layer [37].

linear transform of the initial input v. The output of the Fourier Layer is then obtained as the result of the activation function applied on the constructed sum. A summarizing schematic of FNO is presented in 2.7.
In this section two emergent methods that can be applied for predicting the solutions of partial differential equations were introduced, DeepONet and Fourier Neural Operator. In this case the aim is to implicitly learn the nonlinear Operator and apply it to retrieve the solutions. Besides learning the solutions of partial differential equations, machine learning has been employed for the discovery of governing equations from data observations or from numerical solutions.

2.4 Data-Driven Methods to learn the Partial Differential Equations

Extracting the governing equations from data or observations is a task of high importance as it can utilize the high expressive power of deep learning to re-produce ODEs/PDEs. This conveys more transparency to the black-box property of NNs. And given the complexity of turbulence modeling obtaining reliable deep learning models for this special case uncovers knowledge about the amount of data, the pre-processing, and the overall training, validation, and testing process needed to describe such a fundamental problem. This knowledge can then motivate identifying governing equations for other emerging system dynamics that are yet to be mathematically describes. In this section two deep learning methods able of learning PDEs are introduced: The physics informed neural networks and the PDE-Net.

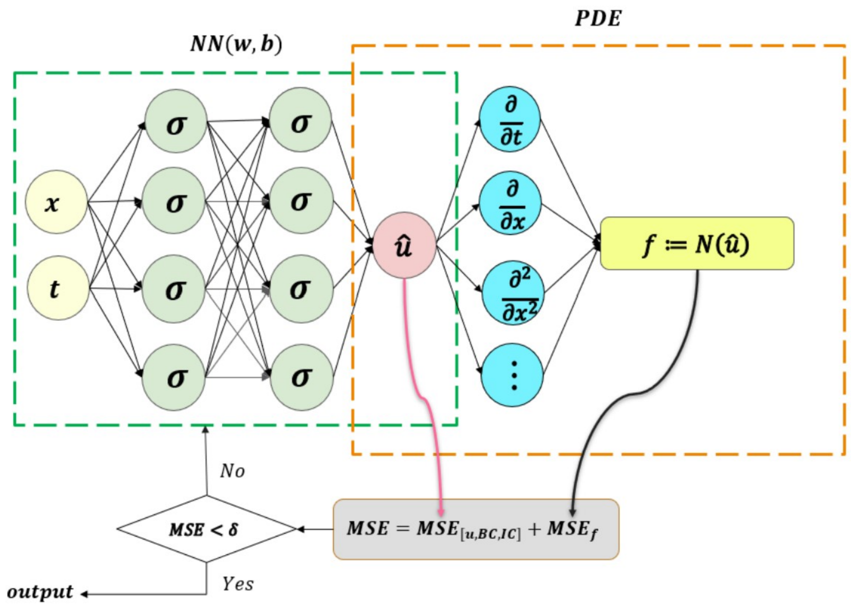


Figure 2.8: Physics Informed Neural Network [25].

2.4.1 Physics Informed Neural Network (PINN)

PINN is a class of NNs that gained a lot of attention as it brought together machine learning and physical laws. In relation to PDEs, PINNs had two applications. Firstly, the solution of partial differential equations in the case of both the presence of sparse and scattered data or the unsupervised case. Secondly, the discovery of the partial differential equations with the help of a limited number of points extracted from a pre-computed solution [48, 65]. In this case, we focus on the case of the discovery of the partial differential equations. Here again, there exit two approaches depending of the problem at hand: the discrete time models, typically related to ODEs, and the continuous time models, typically related to PDEs. For the continuous time PINN models, the general form of the studied parameterized and nonlinear partial differential equations is defined by :

$$u_t + N[u, \lambda] = 0, x \in \Omega, t \in [0, T], \tag{2.22}$$

The left hand-side of the equation is defined by $f(t, x)$ the PDE of the given data and that should be minimized by PINN. In the schematic 2.8, x are locations scattered in space, t are discrete time steps, $\hat{u}$ is the output of the Feed Forward NN. The latent variable $\hat{u}$ is then differentiated, using Automatic Differentiation [5], to construct the

flow variables and the their partial derivatives involved in the PDE. PINN approximates both the solution of the problem u(t,x) and the parameters lambda conditioning the governing equations: $u_t + N[u;\lambda] = 0$
f= $u_t + N[u;\lambda]$, with f the PDE part of the loss function to be minimized. The Mean Squared Error is defined as follows:

$$J(\theta) = MSE_u + MSE_f \tag{2.23}$$

$$MSE_u = \frac{1}{N}\sum_{i=1}^{N} |\hat{u}^i - u(x^i, t^i)|^2, \tag{2.24}$$

$$MSE_f = \frac{1}{N}\sum_{i=1}^{N} |f(x^i, t^i)|^2, \tag{2.25}$$

$$\tag{2.26}$$

N is the number of training points in the space-time domain. MSE_f requires the NN to satisfy the constraints of the partial differential equation, therefore the form of the PDE need to be known beforehand [25].
θ are the weights and biases parameters of the FNN. The loss function 2.23 is minimized by optimizing the aforementioned parameters following the formula [25]:

$$w^* = argmin_{w\in\theta}(J(w)) \tag{2.27}$$

$$b^* = argmin_{b\in\theta}(J(b)). \tag{2.28}$$

$$\tag{2.29}$$

For the studied experiments, the network is able to identify the underlying partial differential equation with remarkable accuracy, even in the case where the scattered training data is corrupted with noise [65].

Instead of using a feed forward NN, Gaussian process regression [63, 62] was used for PINN to predict the parameters from two consecutive time steps. As GPs lose efficiency in high dimensional spaces, FNN is deemed more efficient and robust to noise.
To construct the loss function of the PINN, the form of the PDE should be known, one approach that requires less constraints about the form of the PDE is PDE-Net.

2.4.2 PDE-Net

The introduction of PDE-Net [42] is the result of studying previous methods that tackled the same problem of " deducing optimal equations of motion from observations of time-dependent behavior" as stated in the 1987 work of Crutchfield and McNamara [15]. Multiple works [8, 72, 70, 9, 11] used symbolic and sparse regression to search the space of ordinary differential equations and choose the one that best fit the data. [65] presented a framework that can be employed when the form of the nonlinear response of a PDE is known, except for some scalar parameters. Physics informed neural networks, presented previously 2.4.1, can learn the unknown parameters by introducing regularity between two consecutive time steps using a Gaussian process [63] and later with NNs in [65] .

Different works using NNs for predictive purposes, can confirm that deeper neural networks have more expressive power and therefore can be used for more complex dynamics. However, the limitation of deep learning nowadays is the sole focus on expressive power and prediction accuracy. These networks usually lack transparency to unravel underlying PDE systems, although they may perfectly fit the observed data and perform accurate predictions, as seen in the applications of section 2.3. Therefore, the proposed methods need to combine both Deep Learning and applied mathematics so that one can learn the underlying PDEs of the dynamics and make accurate predictions at the same time [42]. Such an approach is a feed-forward network, named PDE-Net, based on the following generic nonlinear PDE equation:

$$u_t = F(x, u, \nabla u, \nabla^2 u, ...), x \in \Omega \subset R^2. \tag{2.30}$$

The objective of the proposed NN is to learn the form of the nonlinear response F and to perform accurate predictions. Unlike PINNs, the PDE-Net approach requires only a minor knowledge of the form of the nonlinear response F. It only requires the highest order of the differential operators involved, however, it doesn't require any knowledge about which differential operators are involved.

The nonlinear response F can be learned using neural networks, while the discrete approximations of the differential operators are learned using convolution kernels (i.e filters) jointly with the learning of the response F. The data is presented as a series of measurements of some physical quantities $u(t, .) : t = t_0, t_1, ...$ on the spatial domain $\Omega \in R^2$, with u(t,.): $\Omega \in R$.

In the 2-dimensional space the general form of the PDE associated with the observed data is:

$$u_t(t, x, y) = F(x, y, u, u_x, u_y, u_{xx}, u_{xy}, u_{yy}, ...), (x, y) \in \Omega \subset R^2, t \in [0, T] \tag{2.31}$$

The architecture of PDE-Net has two components:

- First a part to automatically determine the differential operators involved in the PDE and their discrete approximations.
- A second part to approximate the nonlinear response function F.

Next, the connection between the order of sum rules of filters and the order of differential operators is explained.

Convolutions and Differentiations

The work of Cai et al. [10], introduced the relation between the variational and the wavelet frame methods for image restoration. The variational method views images as functions defined on a continuum (analog images), whereas the wavelet frame based approach aims to restore a sequence from an observed sequence in a digital image [10].

Definition of the order of sum rules: for a fitter q, α is said to be the order of sum rules of q, where $\alpha = (\alpha_1, \alpha_2), \alpha \in Z^2_+$, under the condition that

$$\sum_{k \in Z^2} k^\beta q[k] = 0, \tag{2.32}$$

for all $\beta = (\beta_1, \beta_2) \in Z^2_+$ with $|\beta| = \beta_1 + \beta_2 \leq |\alpha|$,
and for all $\beta \in Z^2_+$, with $|\beta| = |\alpha|$

If 2.32 holds for all $\beta \in Z^2_+$with$|\beta| < K$, except for $\beta \neq \beta_0$ with certain $\beta_0 \in Z^2_+$ and $|\beta_0| = J < K$, then we say q to have total sum rules of order K\(J+1)

Relation between a filter and a differential operator: Let q be a filter with sum rules of order $\alpha \in Z^2_+$. Then for a smooth function F(x) on R^2, we have

$$\frac{1}{\varepsilon^{|\alpha|}} \sum_{k \in Z^2} q[k] F(x + \varepsilon k) = C_\alpha \delta^\alpha / \delta x^\alpha F(x) + O(\varepsilon), \text{ as } \varepsilon \to 0 \tag{2.33}$$

C_α defined in 2.33 is equal to

$$C_\alpha = \frac{1}{\alpha!} \sum_{k \in Z^2} k^\alpha q[k] \tag{2.34}$$

If in addition, q has the total sum rules of order K\α +1 for some K>$|\alpha|$, then

$$\frac{1}{\varepsilon^{|\alpha|}} \sum_{k \in Z^2} q[k] F(x + \varepsilon k) = C_\alpha \delta^\alpha / \delta x^\alpha F(x) + O(\varepsilon^{K-|\alpha|}), \text{ as } \varepsilon \to 0 \tag{2.35}$$

If K = $|\alpha|$, then the approximation is of first order. According to this work [17], an αth order differential operator can be approximated by the convolution of a filter with α order of sum rules. And according to 2.35, one can obtain a high order approximation of a given differential operator if the corresponding filter has an order of total sum rules with K>$|\alpha|$+k, $k \geq 1$

Moment Matrix: For a given filter, the moment matrix is be used to constrain filters in the PDE-Net. For an N x N filter q, the moment matrix q is defined as:

$$M(q) = (m_{i,j})_{NxN}, \text{ where } m_{i,j} = \frac{1}{(i-1)!(j-1)!} \sum_{k \in Z^2} k_1^{i-1} k_2^{j-1} q[k_1, k_2], \text{ for i,j =1,2,..N.} \tag{2.36}$$

The (i,j)-element of M(q) is called the (i-1,j-1)-moment of q. Combining 2.36 and 2.33, one can design the filter q to approximate any differential operator at any given approximation order by imposing constraints of M(q). For example, if the goal is to approximate $\frac{\delta u}{\delta x}$ up to a constant by convolution $q * u$, where q is a 3 x 3 Filter. The moment matrix can take, in this case, one of the presented shapes:

$$M(q) = \begin{pmatrix} 0 & 0 & * \\ 1 & * & * \\ * & * & * \end{pmatrix}$$

$$M(q) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & * \\ 0 & * & * \end{pmatrix}$$

The * entry means that there is no constraint on the value. The Moment matrix 2.4.2 guarantees that the approximation accuracy is at least of first order, whereas 2.4.2 guarantees an approximation of at least second order.In the case that all entries are pre-determnined as in:

$$M(q) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

In this case, the corresponding filter can be uniquely determined,Here we speak of a "frozen" filter.In the PDE-Net all filters are learned subject to partially defined moment matrices. The trade-off here is between the size of the filters and the memory overhead and computation costs. As bigger filters have a stronger representation capability, to approximate higher order differential operators, however can lead to memory overhead.

Architecture of PDE-Net

In the proposed PDE-Net [42], the considered time discretization is the forward Euler. The training for long-time prediction is performed layerwise, one layer is called a δ-Block.

One δt-Block:
Let $\tilde{u}(t_{i+1},.)$ be the predicted value of u at time t_{i+1} based on the value of u at t_i

$$\tilde{u}(t_{i+1},.) = D_0 u(t_i,.) + \Delta t.F(x,y,D_{00}u,D_{10}u,D_{01}u,D_{20}u,D_{11}u,D_{02}u,...) \tag{2.37}$$

Here D_0 and D_{ij} are convolution operators with the underlying filters denoted by q_0 and q_{ij}, i.e $D_0 u = q_0 * u$ and $D_{ij}u = q_{ij} * u$.
The operators D_{10}, D_{01}, D_{11}etc. approximate differential operators ,i.e $D_{ij} \approx \frac{\delta^{i+j}u}{\delta^i x \delta^j y}$. Instead of using the identity, the authors opted to using average operatorsD_0 and D_{00} to improve stability and enable the network to learn more complex dynamics. The sole assumption on the governing equation is that it is of the form 2.31 and its highest order is smaller or equal to a pre-defined scalar. Approximating F is equivalent to a multivariate regression problem, using a neural network with shared weights across the computation domain Ω [42].

PDE-Net (Multiple δt-Block:) One δt-Block only guarantees the accuracy of one-step dynamics, which does not take error accumulation into consideration. This leads to instability in the prediction. To enable stability and long-term prediction multiple δt-Blocks are stacked to construct a deep network.
Loss function: Considering a data set $u_j(t_i,.) : i, j = 0, 1, ...,$ where j indicates the j-th solution in dimension j with a certain initial condition of the unknown dynamics. The PDE-Net is trained with n δ-blocks. For a given $n \geq 1$, every pair of the data $u_j(t_i,.), u_j(t_{i+n},.)$, for each i and j, is a training sample, where $u_j(t_i,.)$ is the input and $u_j(t_{i+n},.)$ is the ground truth to be matched to the output of the PDE-Net. Here, the l_2 loss function is used:

$$L = \sum_{ij} l_{ij} \text{ , where } l_{ij} = \left\| u_j(t_{i+n},.) - \tilde{u_j}(t_{i+n},.) \right\|, \tag{2.38}$$

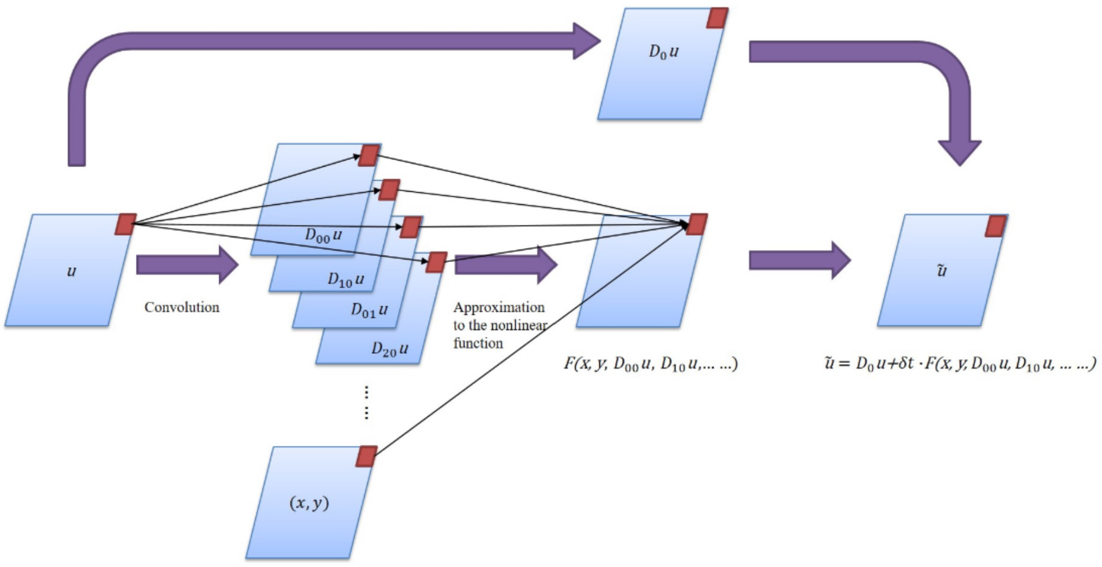
where $\tilde{u_j}(t_{i+n},.)$ is the output of the PDE-Net with $u_j(t_i,.)$ as input.
Constraining the filters: All filters can be constrained thanks to the aforementioned moment matrices. In the case of the average filters q_0 and q_{00}

$$(M(q_0))_{1,1} = 1(M(q_{00}))_{1,1} = 1 \tag{2.39}$$

in the case of the filters q_{ij} that control the parameters D_{ij} where $i + j > 0$

$$\begin{aligned} (M(q_{i,j}))_{k_1,k_2} = 0, k_1 + k_2 \leq i + j + 2, (k_1, k_2) \neq (i+1, j+1) \\ (M(q_{i,j}))_{i+1,j+1} = 1 \end{aligned} \tag{2.40}$$

A schematic representation of the δt-Block and the overall PDE-Net architecture for n stacked δt-Blocks is presented in figure 2.9 and 2.10 respectively.



$D_0 u$
u
Convolution
$D_{00} u$
$D_{10} u$
$D_{01} u$
$D_{20} u$
Approximation to the nonlinear function
$F(x, y, D_{00}u, D_{10}u, \dots \dots)$
$\tilde{u}$
$\tilde{u} = D_0 u + \delta t \cdot F(x, y, D_{00}u, D_{10}u, \dots \dots)$
(x, y)

Figure 2.9: δt-Block [42].

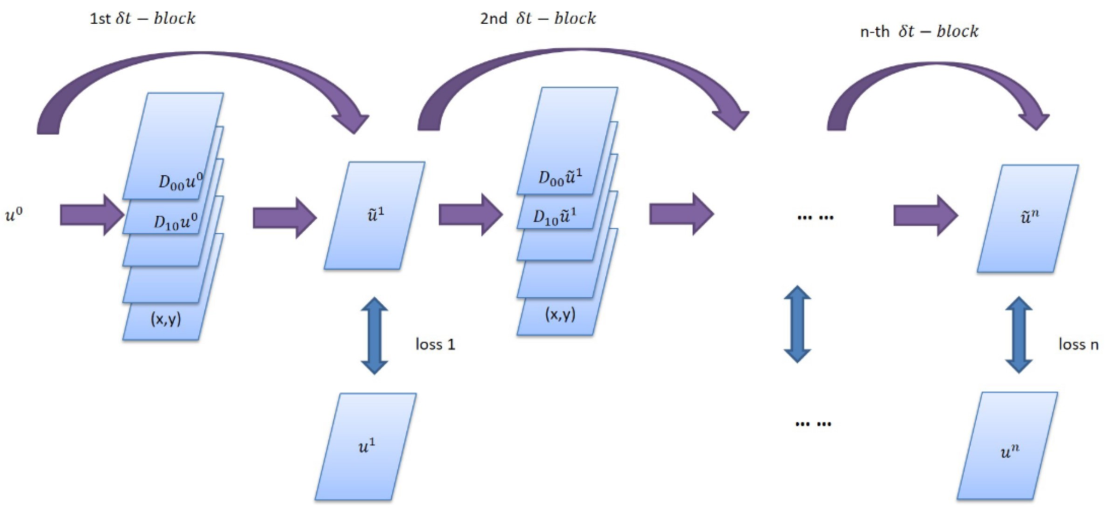


Figure 2.10: PDE-Net:n δt-Blocks [42].

3 Learning Fluid Dynamics Models that Exhibit Turbulent Behavior

This section represents the main methods and results derived in this thesis. First, the numerical experiments for data generation are introduced, then the architectures and training process of both PDE-Net and PINN are explained. Finally, the results of the different experiments are presented.

3.1 The difference between solving and learning the PDE using data-driven methods

Since the invention of calculus, ODEs and PDEs have been the most used class of models to describe and understand natural phenomena. Due to the difficulty of solving partial differential equations, even with the numerical methods simplifications it requires expert knowledge to design numerical solvers and setup experiments that reliably approximate the solutions, which itself requires high computational power. However, with the rise of data-driven methods and improvement of run time for simulations, formed the idea of defining a semi-supervised problem and training NNs to work as solvers for PDEs. The process is as follows, given a PDE $L(u) = f$, for which the domain, initial and boundary conditions are known, the task is to compute an approximation of the solution to the PDE, knowing that NNs are universal approximators for any function or operator given proper definition and training. Independently from the deep learning model used, the goal is to minimize the Mean Squared Error (MSE) $||L(\hat{u}) - f||$ for the training data and generalize to unseen scenarios. This approach produces end-to-end differential simulators, that can be optimized by back-propagation. The problem of data-driven discovery of equations for system control [46], or differential equations either ordinary or partial has been a prominent topic of research since the 1980s. For ODE discovery, sparse regression was proven to be a suitable approach to approximate dynamical systems governed by ODEs, sparse identification of nonlinear dynamics (SINDy) being the latest development for the application of sparse dynamics. Building on this methods, PDE functional identification of nonlinear dynamics (PDE-FIND), attempts at discovering an ODE describing the data obtained as a solution of

PDE. This is achieved by applying dimensionality reduction methods, such as proper orthogonal decomposition (POD), to the high-dimensional PDE solutions, to obtain only a few dominant coherent structures. Due to the importance of high-dimensionality in PDE, which leads numerical solutions to suffer from the curse of dimensionality in the case of complex problems, newer methods that identify PDE systems directly from their solutions were introduced. Here two main approaches have emerged. On the one hand, inferring the parameters of a given parametrized PDE, from a given set of observations of the solutions. This line of research introduced a new family of NNs known as the Physics-informed Neural Networks 2.4.1. On the other hand, to offer more generalization, other approaches aimed at learning the derivative and differentials that define the PDE as well, one model is PDE-Net 2.4.2, which approximates the differential operators using convolutional networks. While both data-driven models used to solve or learn PDEs approximate a function F that approximates the PDE solutions, the models that learn PDE predict an additional equation that describe the PDE used to generate the data.

3.2 The experiments

The computational experiments for this work consist of the data generation using the open-source CFD software OpenFOAM [52] and the deep learning models' training. The numerical problems solved here are governed by:

- the Diffusion equation which is a PDE that describes the physical phenomena where particles,energy, or other physical quantities are propagated in a physical system [12].
- The Navier-Stokes equations explained in section 2.1

3.2.1 Diffusion Equation: scalarTransportFoam

The scalarTransportFoam solver uses a complete convection-diffusion equation, in the incompressible form (the equation is divided by the density) [71]:

$$\frac{\delta T}{\delta t} + \nabla(UT) - \nabla^2(DT) = 0, \tag{3.1}$$

where T is the transported scalar, U is the fluid velocity, and $D[m^2 s^{-1}]$ is a constant representing the fluid diffusivity divided by the fluid density In this case we set the fluid velocity to 0, and we deal with the just the diffusion problem (no convection).
Using the *blockMeshDict* dictionary, a 3-dimensional mesh is defined, with x, y, and

z all ranging from $[0, 2\pi]$. The *convertToMeters* keyword is set to 1 meaning that the geometry is defined in meters. Eight vertex coordinates are introduced, which are then used to define the hexahedral block and the mesh size. Since the simulation is run in 2-dimensional space, only one cell was reserved for the z-direction, while 150 cells were defined for each the x and y directions. The *boundary* keyword defines the boundary of the mesh as list of patches or regions. Each patch is assigned a name that describes its functionality (e.g. inlet, outlet, wall, etc.) and contains information about:

- type: the patch type, a type *patch* means that some boundary conditions are applied.
- faces: a list of block faces that make up the patch. a face is a list of vertices ordered in a counter-clockwise manner.

A patch of type *empty* is used to omit block faces especially in the case of a problem in 2-dimensional space. For the diffusion case, two boundary types are defined an inlet with on faces and the rest being side boundaries. The *controlDict* dictionary helps setting the essential input parameters. As the OpenFOAM solvers begin all run by creating a database, which controls the input and output data [53]. Since the output data is requested at intervals of time during the run, time control and *writeInterval* parameters are mandatory. The time control defines the start and end time of the simulation, and its time step deltaT. The time step Δt, the length between mesh elements Δx, and the magnitude of the velocity u define a dimensionless value named the Courant number, which satisfies the Courant-Friedrichs-Lewy (CFL) condition, which states that "the distance that any information travels during the time step length within the mesh must be lower than the distance between mesh elements. In other words, information from a given cell or mesh element must propagate only to its immediate neighbors" [27]. The equation for the Courant number is:

$$C = u\frac{\Delta t}{\Delta x}. \tag{3.2}$$

The Courant number must be below 1 and should ideally be below 0.7. " If the Courant number exceeds 1, the time step is too large to see the particle in one cell, it “skips” the cell. If it is smaller than 0.7, the particle stays in the cell for at least two time steps" [27]. In *controlDict*, the entry *maxCo* can set the maximum Courant number allowed, and *adjustTimeStep* can be used to adjust deltaT to satisfy the condition on maxCo. From the data writing entries, only the *writeInterval* entry is mandatory to define, which is a scalar that specifies the exact time point data is saved to the database. Other entries are set to default values and can be modified if needed. Finally, in addition to time control, the control dictionary offers the possibility to import libraries and run function

Table 3.1: General parameters for the scalar transport experiment

command	**Dictionary**	**parameters**
blockMesh	blockMeshDict	150x150x1 cells 2 patches (inlet, sides)
setExprField	setExprFieldsDict	sets the initial value of the scalar according to 3.3
scalarTransportFoam	controlDict/ fvSchemes/fvSolution	startTim:0 EndTime:0.6 deltaT:0.015 writeInterval:0.015
postProcess -func internalCloud	internalCloud	150x150 grid cellPoint interpolation scheme

at run-time. These functions range from field calculation to sampling for graph plotting. After running the simualtion, Paraview [57] is an open-source, multi-platform data analysis and visualization application that can be used to visualize the OpenFOAM simulation results and export field values a selection of locations to text files. However, since the mesh might be denser in some areas and coarser in other (in the upcoming fluid flow experiment), the *postProcess* utility from OpenFOAM can write out the values of fields interpolated to a specified cloud of points. The command line *postProcess -func "samplingFunc"* is ran in the experiment directory and outputs a *postProcessing* directory with the field values at the defined regular grid and fol the different saved time steps. The interpolation scheme used to define the grid is *cellPint*, which given the cell centre values and the vertex values, decomposes into tetrehedra and perform linear interpolation with them [52].
The table 3.1 summarizes the parameters necessary to run the scalar transport experiment.

3.2.2 Learning Process: PDE-Net for learning the diffusion equation

The code used for training PDE-Net was modified from the official PDE-Net implementation [59]. The used Machine Learning framework is PyTorch [58]. While the initial implementation uses an older version of PyTorch, 0.3.1, this work uses a newer and stable version 1.10.2 and Python 3.9.7, created as an Anaconda [3] environment to train and test the deep learning models. The training process of PDE-Net is a layer-wise training, where each δ-block, as explained previously in subsection 2.4.2, represents a layer. Meaning that a 10-layer PDE-Net with δt=0.015 and starting from time step

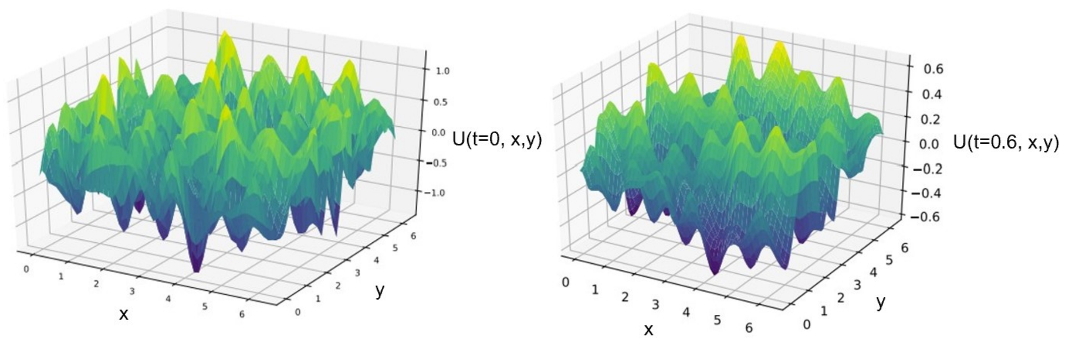


Figure 3.1: **Left:** initial value of U, **Right:** values of U at last timestep.

0, the output of PDE-Net will be an approximation of the solution at time step 0.15. To construct the data set we follow a similar scheme to the one proposed in [41]. The initial value of $u_0(x,y)$ is

$$u_0(x,y) = \sum_{|k|,|l| \leq N} \lambda_{k,l} \cos(kx + ly) + \gamma_{k,l} \sin(kx + ly), \tag{3.3}$$

where N=9, $\lambda_{k,l}, \gamma_{k,l} \sim \mathcal{N}(0, \frac{1}{50})$, and k and l are two random integers ranging from -9 to 9. The evolution of the field from the initial values to t=0.6 can be seen in the plots from figure 3.1. As mentioned before the solution is interpolated to a 150x150 mesh. To ensure robustness to limited and lower-resolution data, the data fed for training is downsampled from the originally generated data to a 50x50 mesh. Each grid value at the initial time step are considered input and the grid values at the final time step are the true output values. The training is done batch-wise, here we choose a batch size of 24, meaning that before training a layer l=i, 24 simulation will be generated on the fly with different initial conditions for each simulation. The options for the training, such as the batch size, number of layers, filter size, and constraint on moment matrix 2.4.2, are summarized in a YAML file. The *constraint* option can be set to: free, frozen, moment. If set to free PDE-Net is a CNN, if set to frozen all entries in the moment matrix are defined and the convolutional kernel can be deterministically determined. In case of the *moment* constraint, for the first δ-block, PDE-Net is trained with frozen moments matrices and a batch of data, this is the warm-up step. The resulting filters are used as initialization and the training is restarted from the first δ-block. In [42], this was proven to speed up the training. This procedure is repeated until all n-δ-blocks are trained. All parameters are shared between the layers, which also speeds up the training. After training each layer, the difference between the predicted and true values is calculated. According to [41], training layer-wise opposing to training the entire network directly, prevents the failure of training and slow convergence. As for the filter sizes, the dimension of the filter should be higher than the highest order of the differential operators, to be able to represent every possible differential operator than can occur in the PDE. Here, the maximum order pre-defined is 2 and tested filters were of sizes 3x3 and 7x7. The authors provide an applied mathematics pyTorch extension (aTEAM) [59] to help set the constraints to the moment matrices, convert the moment matrix to the convolutional kernel, perform interpolation, and use Finite Difference to approximate partial derivatives. The total number of trainable parameters per δ-block is approximately 16k. A batch containing 24 samples is used to training each δ-block, δ=0.015, and considering that PDE-Net was trained for 10 layers, meaning that 240 data samples are required per batch. During training, the L-BFGS optimizer [34] is used, with F is the function to optimize, the coefficients with the frozen contraint as an initial guess and 1500 maximum iterations of the L-BFGS algorithm. The results of training PDE-Net to learn the diffusion equation are presented in subsection 3.3.1.

3.2.3 Laminar and Turbulent Fluid Flow simulation with pimpleFoam

The solver used here is *pimpleFoam*, which merges both the PISO and Simple algorithms introduced previously in section 2.2.4. It solves the continuity equation, and the momentum equation defined as follows:

$$\nabla \cdot U = 0, \tag{3.4}$$

$$\frac{\delta U}{\delta t} + \nabla \cdot (UU) - \nabla \cdot R = -\nabla p + S_U, \tag{3.5}$$

$$R = \mu(\nabla U + \nabla U^T) - 2/3\mu I(\nabla \cdot U), \tag{3.6}$$

$$S_U = -(\mu D + 1/2\rho U F)U, \tag{3.7}$$

where U is the velocity vector, p the pressure, R the stress tensor, and $\vec{S}_U$ the Momentum source. For incompressible, turbulent flow of Newtonian fluids the momentum equation can be written as follows:

$$\frac{\delta v}{\delta t} + (v \cdot \nabla)v = -\nabla p + (\nu + \nu_t)\nabla^2 v, \tag{3.8}$$

where ν is the kinematic viscosity, and ν_t is the eddy viscosity introduced by the turbulence model.
The turbulence model used here is k-omega Shear Stress Transport (SST) [30], which is a RANS turbulence model. It adds two closure equations for the turbulence model. The first equation describes the turbulence kinetic energy defined by:

$$\frac{D}{Dt}(\rho k) = \nabla \cdot (\rho D_k \nabla k) + \rho G - \frac{2}{3}\rho k(\nabla \cdot u) - \rho \beta^* \omega k + S_k, \tag{3.9}$$

and the second equation is the turbulence specific dissipation rate equation given by:

$$\frac{D}{Dt}(\rho\omega) = \boldsymbol{\nabla} \cdot (\rho D_{\omega} \nabla \omega) + \frac{\rho \gamma G}{\nu} - \frac{2}{3}\rho\gamma\omega(\boldsymbol{\nabla} \cdot u) - \rho\beta\omega^2 - \rho(F_1 - 1)CD_{kw} + S_{\omega}. \quad (3.10)$$

Using these two equations, the turbulence viscosity can be obtained as:

$$\nu_t = a_1 \frac{k}{max(a_1\omega, b_1 F_{23} S)}, \quad (3.11)$$

where F_1 and F_{23} are the first and second blending functions respectively, and the model default coefficients are defined in [49]. Besides, the initialisation of U and p, k, omega, and ν_t need to be initialized. For homogeneous isotropic turbulence, the turbulence kinetic energy can be estimated by:

$$k = \frac{3}{2}(I|u_{ref}|)^2, \quad (3.12)$$

where I is the turbulence intensity, $I = \frac{u'}{U}$, where u' is the root-mean-square of the turbulent velocity fluctuations and U is the Reynolds averaged mean velocity, and u_{ref} is a reference velocity. In the experiments, k is initialised to 3.75e-3m^2s^{-2}. The turbulence specific dissipation rate is initialized as:

$$\omega = \frac{k^{0.5}}{C_{\mu}^{0.25} L}, \quad (3.13)$$

where C_{μ} is a constant equal to 0.09, and L is the turbulence length scale, which is a physical quantity describing the size of large energy-containing eddies in a turbulent flow. $\omega = 0.1s^{-1}$ is the initialization of the turbulent specific dissipation rate.

2D fluid flow around a cylinder

The flow past a cylinder is a classical case in CFD. Here, the case is adapted from [54] to simulate the flow over a rectangular cylinder, with incrementally higher Reynolds number:

$$Re = \frac{UL}{\nu}, \quad (3.14)$$

where U is the streamwise far-field flow speed (in this case U= 1.0 ms^1), L is the characteristic length (the diameter of the cylinder = 1 m), and μ is the kinematic viscosity of the fluid in m^2s^1. The general process to simulate this case is summarized in table 3.2

Table 3.2: General parameters for the laminar/turbulent fluid flow in 2D

command	**Dictionary**	**parameters**
blockMesh	blockMeshDict	32 vertices 8 blocks 5 patches
pimpleFoam	controlDict /fvSchemes/fvSolution	startTim:0 EndTime:20 deltaT:0.001 writeInterval:0.1
postProcess -func internalCloud	internalCloud	80x80 grid cellPoint interpolation scheme

Fluid Flow in 3D

For the 3D case with an external obstacle object, the geometry needs to be extracted first using *surfaceFeatureExtract*, which extracts all edges whose adjacent surface normals are at an angle less than the angle defined in the surfaceExtractDict dictionary. In this case a 180 degree angle extracts all edges and one of zero degrees selects no edges. After the 3D object is extracted, *snappyHexMesh* is used to iteratively fit the mesh created by *blockMesh* to the triangulated surface geometries from the object's .stl file. The *decomposePar* utility can be used here to split the generated mesh to multiple processors and run the solver in parallel, and finally, after the run is terminated, run *reconstructPar* to attach the mesh. The two cases studied here are adapted from the tutorial cases provided by OpenFOAM in *tutorials/multiphase/interFoam/RAS/DTCHull* and *tutorials/incompressible/simpleFoam/airFoil2D*. Airfoils have been a classical example to study aerodynamics, as they are cross-sections of wings, blades of propellers, rotors, or turbines. The general parameters to construct the mesh and run the simulation are summarized in table 3.3. The Duisburg Test Case (DTC) is a hull design, that can be compared to that of container ships. It was introduced for benchmarking and validation of numerical methods. Simulation parameters are defined in table 3.4. This case helps simulate a hydrodynamics problem, for this case a multiphase solver called *interFoam* [28] is used. It solves the NSE for 2 incompressible, isothermal immiscible fluids. This means that the material properties are constant in the region filled by one of the two fluids except at the interphase. It uses the Volume of Fluid (VOF) approach for phase-fraction capturing, which utilizes optimal mesh motion and mesh topology

Table 3.3: General parameters for 3D airfoil case

command	**Dictionary**	**parameters**
surfaceFeatureExtract blockMesh snappyHexMesh createPatch	surfaceFeatureExtractDict blockMeshDict snappyHexMeshDict createPatchDict	angle: 150 8 vertices 9 patches
pimpleFoam	controlDict /fvSchemes/fvSolution	startTim:0 EndTime:0.1 deltaT:0.001 writeInterval:0.001
postProcess -func internalCloud	internalCloud	20x20x20 tensor cellPoint interpolation scheme

Table 3.4: General parameters for the 3D DTCHull case

command	**Dictionary**	**parameters**
surfaceFeatureExtract blockMesh topoSet refineMesh snappyHexMesh	surfaceFeatureExtractDict blockMeshDict topoSetDict refineMeshDict snappyHexMeshDict	angle: 150 28 vertices 6 boxes 6 patches
interFoam	controlDict /fvSchemes/fvSolution	startTim:0 EndTime:0.1 deltaT:0.001 writeInterval:0.001
postProcess -func internalCloud	internalCloud	50x10x50 tensor cellPoint interpolation scheme

changes including adaptive re-meshing. The momentum equations is defined as [28]:

$$\frac{\delta(\rho u_i)}{\delta t} + \frac{\delta}{\delta x_j}(\rho u_j u_i) = -\frac{\delta p}{\delta x_i} + \frac{\delta}{\delta x_j}(\tau_{ij} + \tau_{t_{ij}}) + \rho g_i + f_{\sigma_i}, \tag{3.15}$$

where u represents the velocity, g_i the gravitational acceleration, p the pressure, τ_{ij} and $\tau_{t_{ij}}$ are the viscose and turbulent stresses respectively, and f_{σ_i} is the surface tension. The density ρ is defined as follows:

$$\rho = \alpha\rho_1 + (1-\alpha)\rho_2, \tag{3.16}$$

where α is 1 inside fluid 1 with density ρ_1 and 0 inside inside fluid 2 with the density ρ_2. At the interphase between the two fluids α varies between 0 and 1. For the DTC hull case, fluid 1 is water ($\rho_1 = 998.9\frac{kg}{m^3}$) and fluid 2 is air ($\rho_2 = 1\frac{kg}{m^3}$). The surface tension f_{σ_i} is modeled by the equation:

$$f_{\sigma_i} = \sigma\kappa\frac{\delta\alpha}{\delta x_i}, \tag{3.17}$$

σ is the surface tension constant and κ the curvature approximated as follows:

$$\kappa = -\frac{\delta n_i}{\delta x_i} = -\frac{\delta}{\delta x_i}\left(\frac{\delta\alpha/\delta x_i}{|\delta\alpha/\delta x_i|}\right) \tag{3.18}$$

3.2.4 Learning Process: PINN for learning NSE

As the computational domain defined to solve the NSE in OpenFOAM is composed of hundreds of thousands of points, the region of interest where most the field changes occur is downstream of the geometry. This observation enables us to define 3D or 2D meshes to sample the velocity and pressure fields in a rectangular region downstream of the obstacle and construct the data set that will be used to train the PINN. Two examples of the rectangular region chosen for the flow over the rectangular cylinder and the DTC hull are shown in figure 3.2. For the DTC hull case, Red represents water, blue is air and the orange colored box is the area in question placed in air as well and in both cases the outlet boundary is defined Generally, N_x =6400 locations were used to sample the 2D flow, and N_x =8000 locations were used to sample the 3D flow.
The trained models for PINN were modified from the first official implementation of PINN [60]. The Machine Learning platform used is Tensorflow 1.13.1 [1]. This older version used for the initial implementation was kept, due to the moderate size of the fully connected neural network, the graph creation and optimization is not the most time consuming task, computing the automatic differentiations of the latent functions using *tf.gradient* with Tensorflow CPU-only is time consuming while training. That is why it is not suspected that the performance would be higher using a newer and more

efficient Tensorflow version.
To facilitate, post processing and analyzing the results, I added the utility to save the models (for performing predictions and testing), loss, and learned parameters during training. In addition, the implementation has been modified to be able to input 3D location points and construct the momentum equation in 3D, while making the true assumption about the latent function 3.21 to predict the velocities in x, y, and z-direction while fulfilling the continuity equation. The feed forward NN output a variable $\hat{x} = [p, \Psi]$, where Ψ is a latent function for the velocity field. The pressure needs to be learned directly and not computed from Ψ as for the velocity field, because the pressure is not a solution of the NSE, but is approximated so that the continuity equation is enforced. the pressure gradient that appears in the momentum equation is needed to solve the NSE and consequently its approximation error should be minimized. The velocity fields are derived from Ψ to satisfy the continuity equation: $\nabla \cdot \vec{u} = 0$. Let u, v, and w denote the flow velocity in x, y , and z-direction respectively. In the work of Raissi, Perdikaris, and Karniadakis [64], u and v are defined as follows:

$$\begin{aligned} u &= \Psi_y, \\ v &= -\Psi_x, \\ \Psi_i &= \frac{\delta \Psi}{\delta i}, \\ u_x + v_y &= 0. \end{aligned} \tag{3.19}$$

As explained previously in subsection 2.4.1, the loss function in PINN is composed of the velocity loss and the constructed PDE loss. With use of automatic differentiation of u and v, the momentum equations is built as follows:

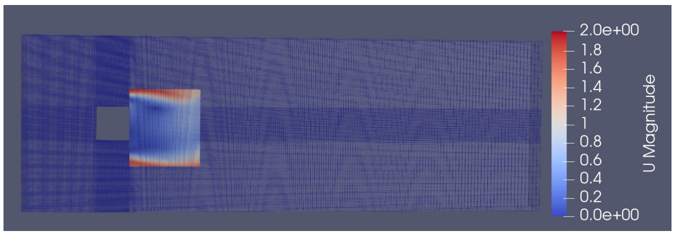
$$\begin{aligned} f &= u_t + \lambda_1(uu_x + vu_y) + p_x - \lambda_2(u_{xx} + u_{yy}), \\ g &= v_t + \lambda_1(uv_x + vv_y) + p_y - \lambda_2(v_{xx} + v_{yy}). \end{aligned} \tag{3.20}$$

In 3D, I define u, v, and w using Ψ as follows:

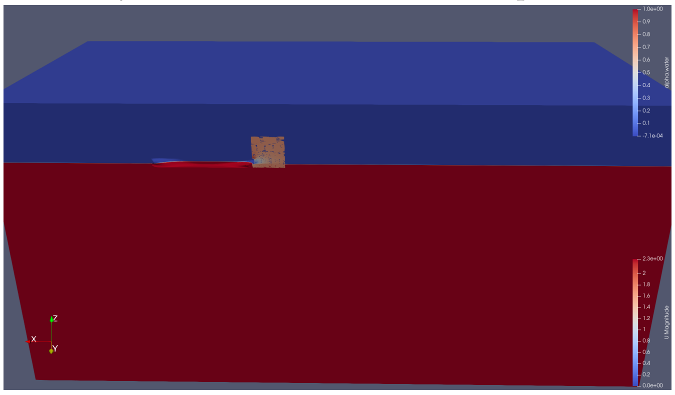
$$\begin{aligned} u &= \Psi_{yz}, \\ v &= -0.5\Psi_{xz}, \\ w &= -0.5\Psi_{xy}, \\ u_x + v_y + w_z &= 0. \end{aligned} \tag{3.21}$$

The physical laws added to the loss function here are defined as follows:

$$\begin{aligned} f &= u_t + \lambda_1(uu_x + vu_y + wu_z) + p_x - \lambda_2(u_{xx} + u_{yy} + u_{zz}), \\ g &= v_t + \lambda_1(uv_x + vv_y + wv_z) + p_y - \lambda_2(v_{xx} + v_{yy} + v_{zz}), \\ h &= w_t + \lambda_1(uw_x + vw_y + ww_z) + p_z - \lambda_2(w_{xx} + w_{yy} + w_{zz}). \end{aligned} \tag{3.22}$$



(a) Selected area for the flow over a cylinder case. The velocity colored area shows the area in question.



(b) Selected area for the 3D DTC Hull case.

Figure 3.2: selected area for training data in the flow over a cylinder and DTC hull case.

The feed forward NN for most cases is composed of 8 layers with 20 neurons per layer. The ADAM optimizer (with a *tanh* activation function) is used to optimize the weights of the FNN with a learning rate of 0.001 for 10000 iterations. Then, the L-BFGS optimizer, is ran for 2500 iterations to further optimize the λ parameters.

3.3 Results

In this section, the results of training the PDE-Net and PINN for learning the two PDE equations will be shown.

3.3.1 PDE-Net for learning the diffusion equation

To show the possibility identifying the diffusion equation using PDE-Net. A 10-layer PDE-Net was trained, where the maximum order assumed is 2, and assuming the Euler temporal dicretization with $t_{i+1} = t_i + \Delta t$, then the filed values at time t_{i+1} can be determined from the previous time step as follows:

The Euler approximation method:

$$\frac{T(t_{i+1},.) - T(t_i,.)}{\Delta t} \approx \frac{\delta T}{\delta t}, \tag{3.23}$$

In general, the time derivative of T is described by the PDE:

$$\frac{\delta T}{\delta t} = F(T, T_x, T_y, ...), \tag{3.24}$$

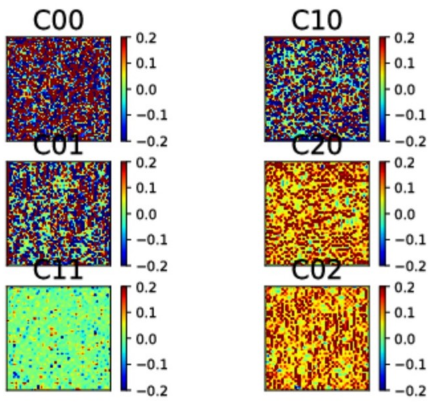
In the case of the diffusion equation:

$$\frac{\delta T}{\delta t} = D_t \nabla^2 T \tag{3.25}$$

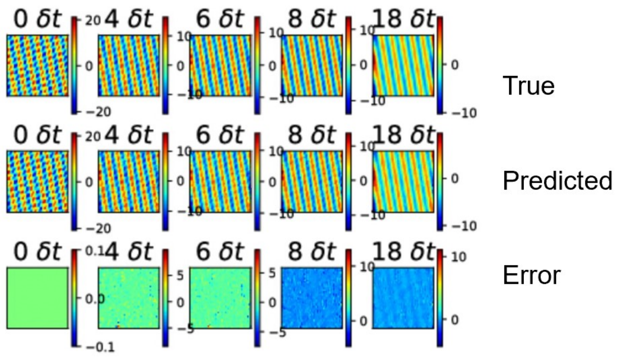
The parametrized F function learned by PDE-Net is:

$$\tilde{T}(t_{i+1},.) = D_0 T(t_i,.) + \Delta t(C_{00}T + C_{10}T_x + C_{01}T_y + C_{11}Txy + C_{20}T_{xx} + C_{02}T_{yy}), \tag{3.26}$$

where C_{ij} are the coefficients of the differential operators that appear in the PDE, C_{00} being an averaging coefficient. $T_{ij} = \frac{\delta T}{\delta i \delta j}$. Meaning that for the diffusion equation, the coefficient C_{20} and C_{02} should be equal to the diffusion coefficient D. To show the expressive power of PDE-Net to predict the time evolution of the diffusion phenomenon, and its ability to uncover the differential operator involved in the PDE and their respective coefficients, multiple 10-layer PDE-Nets, with 24 data samples per batch per layer (δt-block), with 1500 iterations for the L-BFGS optimizer, and 7x7 convolutional kernels, were trained with data produced using different magnitudes of the diffusion coefficients, namely $D = 0.02, 0.2,$ and 1.0. The results of the learned coefficients and the long-time prediction on one test sample are shown in the figures below. The



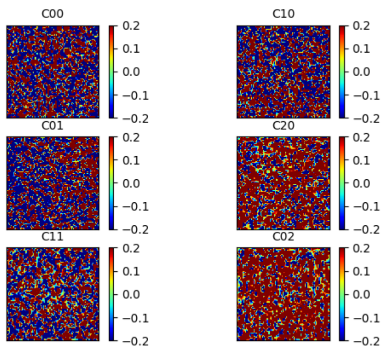
(a) Learned coefficients for test sample.



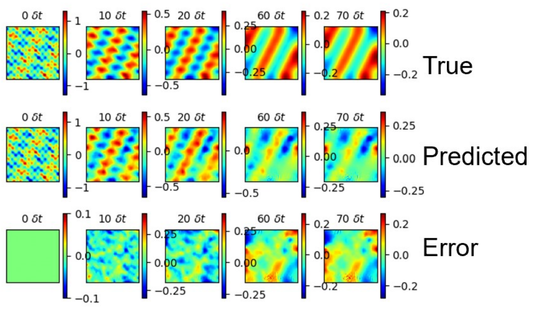
(b) The true, predicted dynamics, and error for the diffusion case.

Figure 3.3: Failed experiment for $D = 0.2$ for wrong initialization of field values.

coefficients depicted in the figures show the local prediction of the PDE coefficient C_{ij} from equation 3.26 at each grid point (x,y) for a test case generated using the distribution defined in equation 3.3 to initialize the case. First starting with $D = 0.2$, multiple experiments deemed unsuccessful, especially in identifying the coefficients. An example for one test case can be seen in the figures of 3.3. As can be seen when comparing the true and predicted dynamics in figure 3.3b, the initial conditions (0 δt) does not show a random distribution, as the initial value function 3.3 was composed of only two sine and cosine terms, which might lead to overfitting during training explaining the ability to recreate the dynamics but not learning the coefficients and their associated differential operators as seen in coefficient images 3.3a. Another factor that was studied is the magnitude of δt the temporal discretization parameter or the step between two time steps, in most experiment δt was chosen to be 0.015, in one test case it was set to 0.05. The results here were similar to those from 3.3, which proves the necessity of using a sufficiently small time increments, to make sure the Euler discretization still approximates the time evolution of T. In a third experiment, a 10-layer PDE-Net was trained with data ranging from the start time zero till 0.15. This experiment does not produce the right PDE nor the long time predictions. A test case for this experiment is presented in figure 3.4, where in image 3.4a the learned coefficients are noisy, despite the average of each coefficient is approximately equal to the true coefficient. In image of 3.4b, the first row shows the true field values at 5 different time steps, the second row shows the predicted solution of the learned PDE at the same time steps, and the last row is the difference between the true and predicted



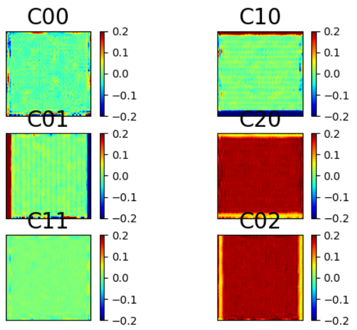
(a) Learned coefficients for test sample.



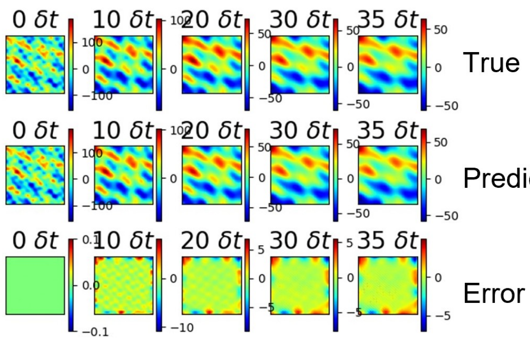
(b) The true, predicted dynamics, and error for the diffusion case.

Figure 3.4: Failed Experiment for $D = 0.2$. using data in the interval [0,015)

solution for the test case. The first two time steps $0\delta t$ and $10\delta t$ correspond to the first and last layers of the trained PDE-Net and the last 3 time steps $20\delta t$, $60\delta t$, and $70\delta t$ are constructed by applying the learned PDE, which explains the low error at the first two time step by the minimal change in the solution, which leads to low coefficient values $T(t_{i+1}, .) \approx T(t_i, .)$, and the inability to predict long-time behavior. Trial and error led to using training data with time steps ranging from $30\delta t = 0.45$ to $40\delta t = 0.55$, this leads to achieving good results for both the learned coefficients and solution prediction. Results for one test example can be seen in figure 3.5. The image 3.5a shows a uniform prediction of all coefficients, which are in agreement with the true coefficients. Image 3.5b also shows a small error for time steps outside the range using for training as well ($10\delta t$ and $20\delta t$). Despite not being mentioned in the initial PDE-net paper [42], the implementation shows the use of a scaling factor of 100 to the data, which in the case of this work does not show a noticeable improvement in the convergence of the L-BFGS algorithm. The scaling factor of 100 explains the higher magnitude of the values in the color maps in image 3.5b. To study the influence of the size of the convolutional kernel, a small filter size was used for training, 3x3 kernel size and smaller batch size of 8, the results for a test sample are shown in figure 3.6. Here the long-time prediction ability of this PDE-Net model was tested, since the data used for training is located between the 30th and 40th timesteps, the solution for the test case was shown for time steps before and after the range used for training. The observation here is the presence of scattered noise in the C_{00} coefficient, which can be related to the less expressive power of the model now with less parameters. This argumentation justifies using a higher size for the convolutional kernel and a higher batch size to obtain more

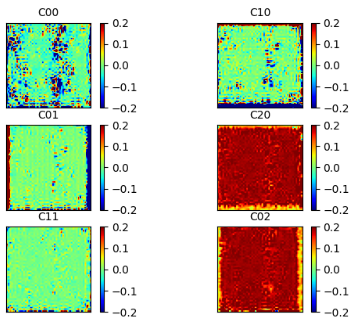


(a) Learned coefficients for test sample.

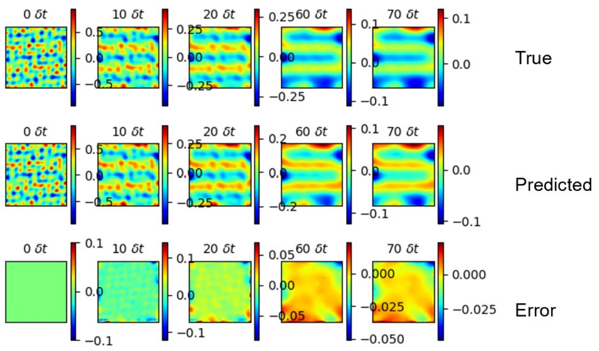


(b) The true, predicted dynamics, and error for the diffusion case.

Figure 3.5: Experiment for $D = 0.2$ and data scaling of 100. Success in PDE coefficient approximation and long-time prediction for test case.



(a) Learned coefficients for test sample.



(b) The true, predicted dynamics, and error for the diffusion case.

Figure 3.6: Experiment for $D = 0.2$, Kernel size 3x3, and batch size 8. Appearance of scattered noise in C_{00}.

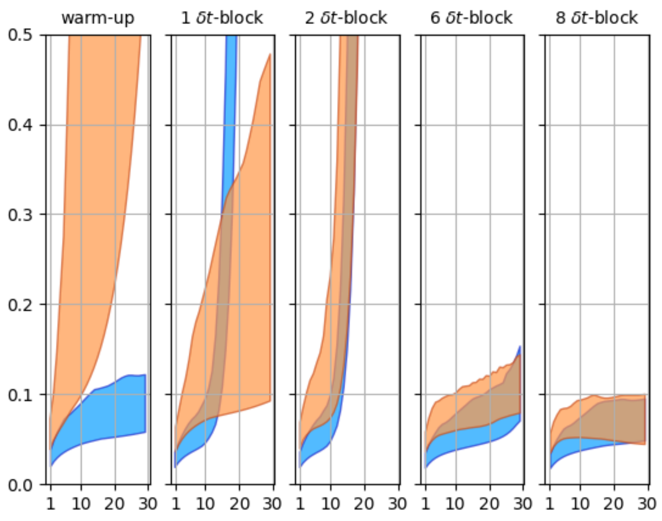
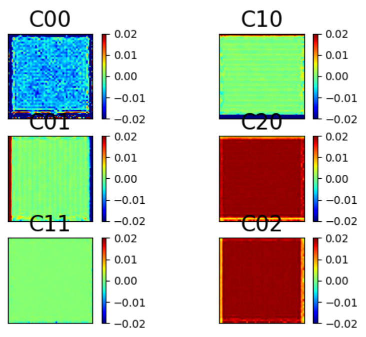
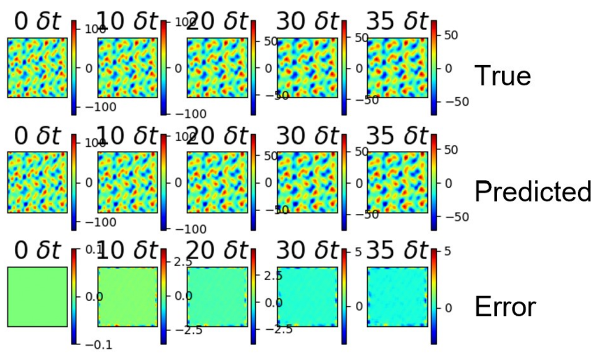


Figure 3.7: Prediction errors of the PDE-Net with 7x7 (blue) and 3x3 (orange) filters. In each plot, the horizontal axis indicates the time of the of prediction in the interval (0, 30$*\delta t] = (0, 0.45]$, and the vertical axis shows the error value. The banded curves indicate the 25% and 75% percentile of the error value among 30 test samples.

training data. To compare the prediction errors of the PDE-Net with 7x7 filters and 3x3 filters, the plot 3.7 is to be referred to. The error values show that using 7x7 filters outperforms the 3x3 filters after the warm-up layer, both models have higher errors in the initial layers, but converge to lower error values towards the later layers. A further experiment decreasing the diffusion coefficient 10 magnitude order to $D = 0.02$ and training a 10-layer PDE-Net with 7x7 filters, and 24 batch size, and data in the interval [0.45,0.55) produces the coefficients and predictions presented in figure 3.8. The learned coefficients show non-zero values for the C_{00} coefficient, signifying a dependency on the previous time step value. This can be justified by the slow modification of the field values, due to smaller Diffusivity. One final experiment, uses a diffusion coefficient $D = 1.0$, leads to high errors between the true and predicted solutions, despite learning the true coefficients. These observations are reported in figure 3.9. Computing the loss values for 30 test cases confirms this observation 3.10

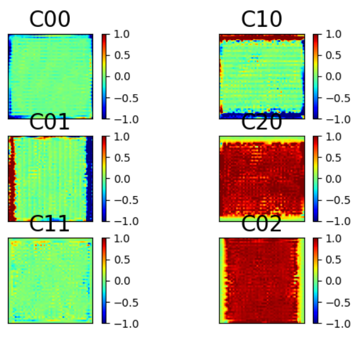


(a) Learned coefficients for test sample.

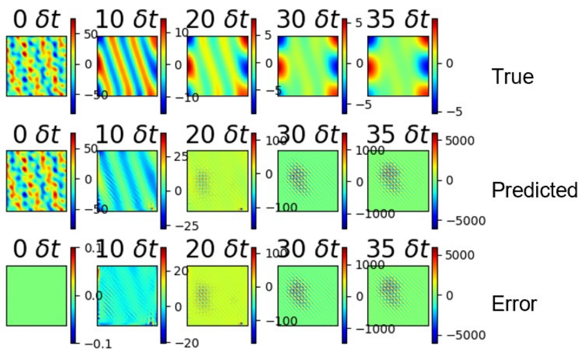


(b) The true, predicted dynamics, and error for the diffusion case.

Figure 3.8: Experiment for $D = 0.02$.



(a) Learned coefficients for test sample.



(b) The true, predicted dynamics, and error for the diffusion case.

Figure 3.9: Experiment for $D = 1.0$.

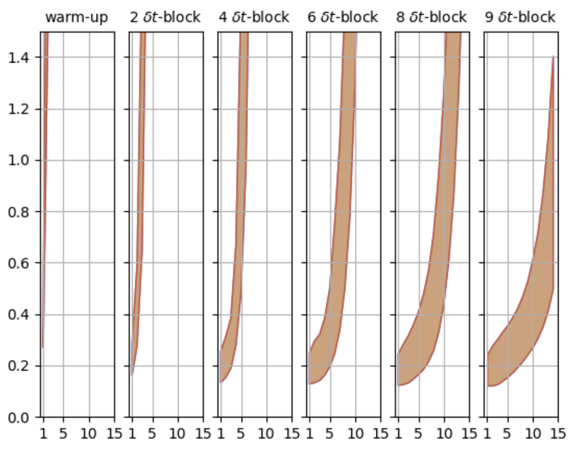


Figure 3.10: Prediction errors of the PDE-Net. The horizontal axis indicates the time of the of prediction in the interval (0, 30$*\delta t] = (0, 0.45]$, and the vertical axis shows the error value. The banded curves indicate the 25% and 75% percentile of the error value among 30 test samples generated with $D = 1.0$.

3.3.2 PINN for learning the parameters of the NSE

For the majority of the experiments in the section the Reynolds number is calculated by the following formula:

$$Re = \frac{UL}{\nu}, \tag{3.27}$$

where U=1ms^{-1}, L=1m, and the varying factor is the kinematic viscosity ν. For the flow over a cylinder, flows with a Reynolds number small than 200 are laminar, above this number the regime of the flow is increasingly turbulent, the results of applying PINN to identify the NSE is presented in this section.

2D flow over a cylinder

Re=20: in this case the Reynolds number is equal to 20 for the case of the kinematic viscosity ($\nu = 0.05m^2.s^{-1}$). 200 data samples in the time interval [0,20) are used for training, with 6400 for each data sample and running the ADAM optimizer for 5000 iterations and the L-BFGS optimizer for 2500 iterations leads to the parametrization of the PDE as in the table 3.5.
Re=100: in this case the Reynolds number is equal to 100 for the case of the kinematic viscosity ($\nu = 0.01m^2.s^{-1}$). The number of training samples and parameters are identical

Table 3.5: The correct and identified PDE for Re=20.

Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.05(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.05(v_{xx} + v_{yy})$
Identified PDE	$u_t + 0.97184(uu_x + vu_y) = -p_x + 0.04808158(u_{xx} + u_{yy})$ $v_t + 0.97184(uv_x + vv_y) = -p_x + 0.04808158(v_{xx} + v_{yy})$

Table 3.6: The correct and identified PDE for Re=100.

Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.01(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.01(v_{xx} + v_{yy})$
Identified PDE	$u_t + 0.94420(uu_x + vu_y) = -p_x + 0.00978612(u_{xx} + u_{yy})$ $v_t + 0.94420(uv_x + vv_y) = -p_y + 0.00978612(v_{xx} + v_{yy})$

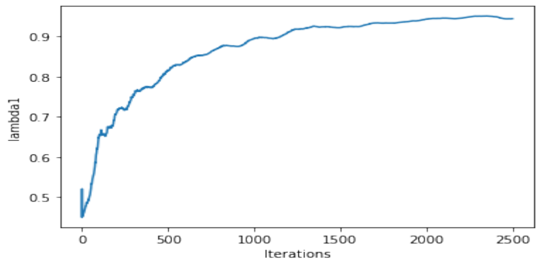
to those defined in the previous paragraph. The correct and identified PDE are given in the table 3.6. Figure 3.11 shows the convergence of both parameters λ_1 and λ_2 to the true values 1.0 and 0.01 respectively.

Re=200: in this case the Reynolds number is equal to 200 for the case of the kinematic viscosity ($\nu = 0.005m^2.s^{-1}$). In this case, 10000 iterations were used for the ADAM optimizer while keeping the other parameters identical to the previous experiments. The velocity profile at the last time step (t=20) showing the turbulent wake periodic regime is presented in figure 3.12 and the learned PDE in table 3.7.

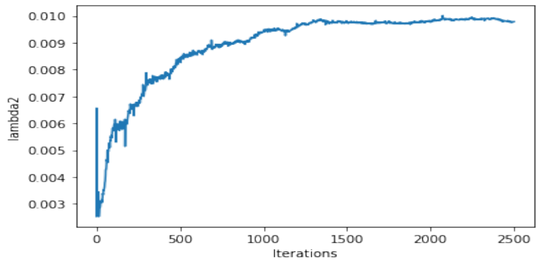
Re=500: This experiment was motivated by the failure in identifying the PDE in a highly turbulent case $\nu = 1.5e - 5m^2.s^{-1}$, where $\lambda_1 = 0.96328$ and $\lambda_2 = 0.01048093$ after training. To closely study the reasoning behind the previous results, I slowly increase the kinematic viscosity's value to $\nu = 0.002m^2.s^{-1}$ this results in Re=500. In this case the velocity profile starts showing separation of the eddies as can observed in figure 3.13. As a reminder, the turbulence model models a turbulent viscosity that is summed to the kinematic viscosity $\nu_{Eff} = \nu + \nu_t$, which is used to solved the momentum equation. The plots in figure 3.14, show the norm of the turbulent viscosity over space for each time step for the Re=100 and Re=500 cases. For the Re=100 case

Table 3.7: The correct and identified PDE for Re=200.

Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.005(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.005(v_{xx} + v_{yy})$
Identified PDE	$u_t + 0.94299(uu_x + vu_y) = -p_x + 0.00528514(u_{xx} + u_{yy})$ $v_t + 0.94299(uv_x + vv_y) = -p_y + 0.00528514(v_{xx} + v_{yy})$



(a) The evolution of λ_1. The true value for $\lambda_1 = 1.0$



(b) The evolution of λ_2. The true value for $\lambda_2 = 0.01$

Figure 3.11: The learned parameter values for Re=100.

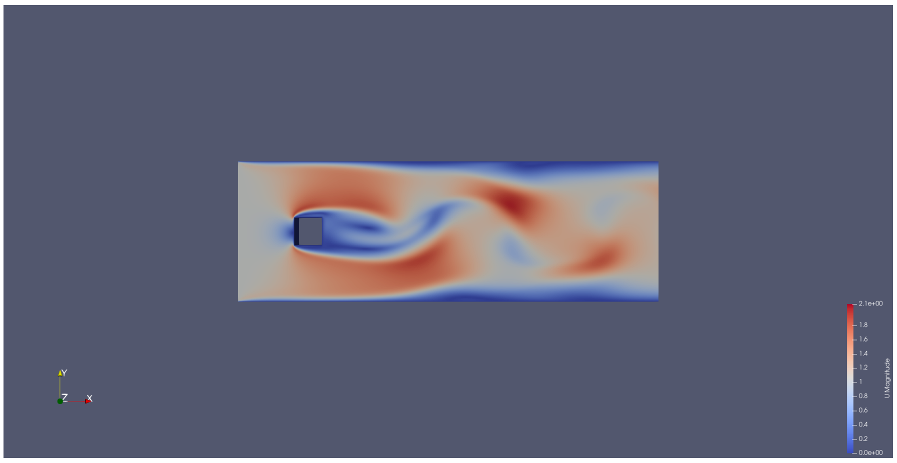


Figure 3.12: The velocity magnitude for time 20 for Re=200.

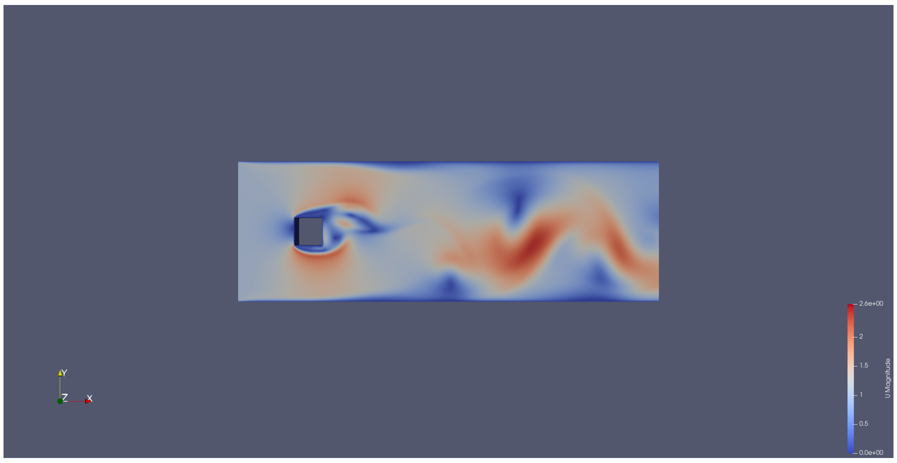
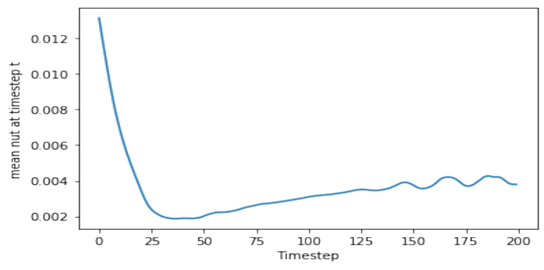
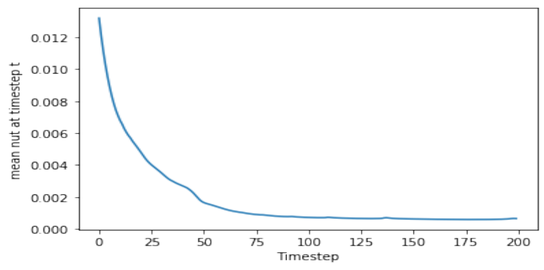


Figure 3.13: The velocity magnitude for time 20 for Re=500.

the turbulent viscosity converges to low values instantly, which justifies ignoring its influence. This observation is in alignment with the case setup as laminar. However, for Re=500, despite the convergence of the eddy viscosity the overall magnitude is considered high in comparison with the kinematic viscosity, and its influence cannot be neglected. The previously mentioned reasoning for neglecting the eddy viscosity in the case of laminar flow, enables us to feed the learned λ_2 value and re-run the simulation. The resulting space averaged eddy viscosities are in agreement with each and their magnitudes can be compared in figure 3.15. Using 200 data samples for the Re=500 case leads to an approximation of of λ_1 that is close to the true value 1.0, however, λ_2 is approximated to be in the orders of 0.006. This value compared to the kinematic viscosity 0.002 value could be explained by the addition of the eddy viscosity that change magnitude in time and space. To limit the change of the eddy viscosity in time, the data samples are limited to two from the initial 200, the space averaged $\nu_t \approx 0.00076$, meaning $\nu_{Eff} \approx 0.00276$. In this case, a 150x150 mesh is used to sample the simulation leading to 22500 points per data sample, a wider FNN is used here with 8 layers and 80 neurons per layer. After running the ADAM optimizer for 10000 iterations, and the L-BFGS for 5000 iterations, the resulting learned parameters are $\lambda_1 \approx 0.57$, and $\lambda_2 \approx 0.0033$. Figure 3.16 shows the evolution of λ_1, λ_2, and the PINN loss function. The loss function is decreasing and can achieve order of magnitude lower than 1, meaning that a higher number of iterations would lead to better approximation of the parameters. A higher learning rate (compared to the 0.001 used here) as well could lead to faster convergence of the parameters as the evolution shows slow convergence.
To be able to use a higher number of data samples calls for using methods that are able to learn spatially varying values which will be discussed in the next section.

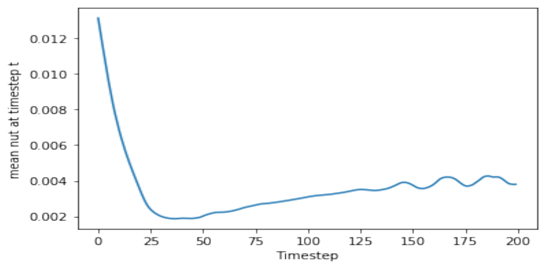


(a) The space averaged eddy viscosity for Re=100.

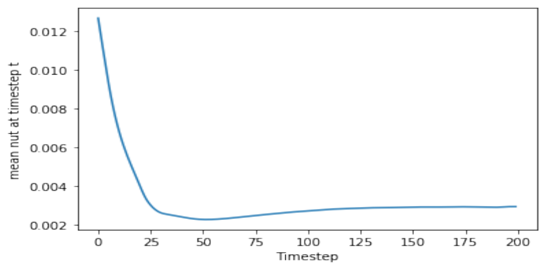


(b) The space averaged eddy viscosity for Re=500.

Figure 3.14: The averaged eddy viscosity for Re=100 and Re=500.

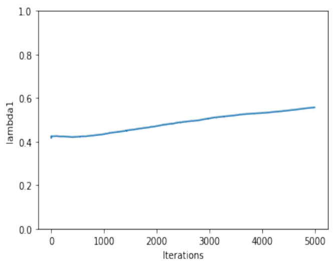


(a) The space averaged eddy viscosity for Re=100.

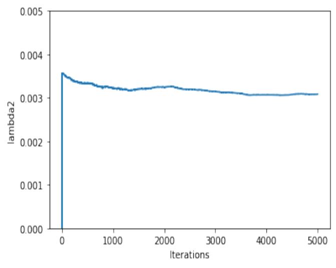


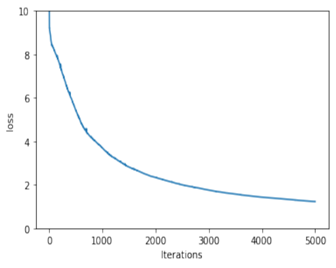
(b) The space averaged eddy viscosity for Re=100 for the simulation ran using the learned λ_2.

Figure 3.15: The averaged eddy viscosity for Re=100.



(a) The learned λ_1 parameter for Re=500. The true λ_1=1.0.





(b) The learned λ_2 parameter for Re=500. (b) The PINN loss for Re=500.

Figure 3.16: λ_1, λ_2, and loss function for Re=500.

Learning the PDE for turbulent flows in 3D

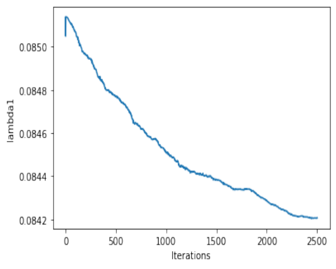
Up to this point, the results discussed were for 2D turbulence, which is a simplification introduced by RANS/URANS simulations. In 2D turbulence, small eddies disappear and merge in larger eddies, leading to more energy produced by the mean flow and less energy in the fluctuations. However, in 3D, with the presence of vortex stretching, the problem is much more complex and is rela As previously introduced, two 3D cases were studied for the 3D turbulent flow: firstly the multiphase flow for the DTC hull case, and the airflow around an airfoil.

The DTC Hull case Here the reference velocity $U = 1.668 m.s^{-1}$ in x-direction, the characteristic length $L = 0.244m$, and the kinematic viscosity of air is set to $\nu = 0.001 m^2.s^{-1}$. The resulting Reynolds number is $Re = 406.992$. Considering the PDE solved by the interFoam solver defined in equation 3.15, and the PDE added to the PINN loss function 3.22, the momentum equation for the z velocity component is slightly changed to equate for the z component of the gravity acceleration as seen in equation 3.28.

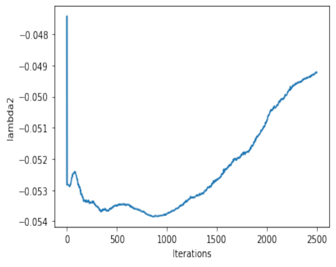
$$h = w_t + \lambda_1(uw_x + vw_y + ww_z) + p_z - \lambda_2(w_{xx} + w_{yy} + w_{zz}) + 9.81. \tag{3.28}$$

10 data samples in the time interval [0.001,0.01] with 25000 points per samples were used for training. An 8-layer FNN with 80 neurons per layer is used. The images of figure 3.17, show the evolution of the learned parameters, the PINN loss function, and the space average eddy viscosity. Despite the eddy viscosity being in the order of $\approx 7e-6$, the learned $\lambda_1 = 0.084$, $\lambda_2 = -0.04922$. Looking at the different components of the PINN loss function, the difference between the true and predicted velocity in x direction is approximately $1.7e-3$, the difference between the true and predicted velocity in y direction is approximately $5.7e-1$, the difference between the true and predicted velocity in z direction is approximately 3.08, and the difference between the true and predicted pressure is approximately $9.99e-1$. Seeing the high error value especially for the pressure and the velocity in z-direction could signify an inherent problem in structuring the latent function Ψ as defined in 3.21 and might suggest bypassing defining the velocities with help of the latent function to satisfy the continuity equation, and rather predict u, v, and w with the FNN directly and add the continuity equation to the PINN loss function. In addition, the high error for the pressure could be an indication to use a deeper NN to be increase the expressive power.

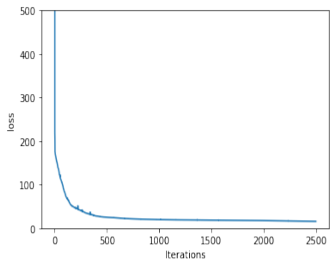
The 3D airfoil case Here the reference velocity $U = 1.0 m.s^{-1}$ in x-direction, the characteristic length (Local chord length of the airfoil) $L = 1m$, and the kinematic



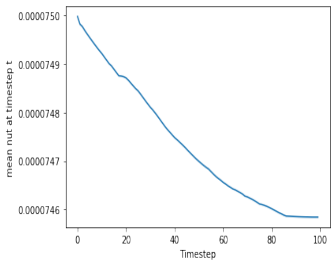
(a) The learned λ_1 parameter for the 3D DTC hull case. The true λ_1=1.0.



(b) The learned λ_2 parameter for the 3D DTC hull case.



(c) The PINN loss for the 3D DTC hull case.



(d) The space averaged turbulent viscosity for the 3D DTC hull case.

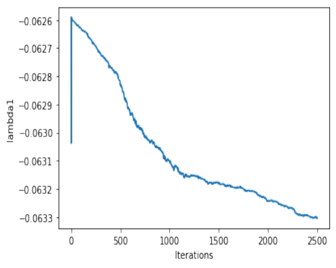
Figure 3.17: λ_1, λ_2, loss value, and averaged turbulent viscosity for the DTC hull case.

viscosity of air is set to $\nu = 1e - 5m^2.s^{-1}$. The resulting Reynolds number is $Re = 1e5$. The results for this case are similar to the previous case and due to the higher Reynolds number, the turbulent viscosity is higher as can be seen in the fourth image of figure 3.18. Due to the change of magnitude of ν_t, only two data samples with 8000 points per sample were used for training. The results for both the learned λ_1, and λ_2 are reported in figure 3.18, plus the higher loss hints to the significance of using multiple parameters, specifically 6 instead of two, which leads to a more complex line search for L-BFGS and significantly higher iteration number for the problem to converge.

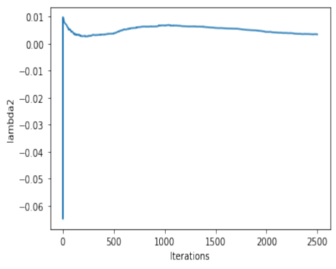
3.4 Learning turbulent flow models

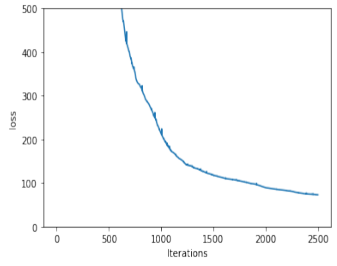
The machine learning models studied in this section show that PDE-Net can be used to learn the governing equations, namely of linear PDEs, and in this case diffusion equation. A second version of PDE-Net [40] uses symbolic regression and Equation Learner (EQL) [46], to group (limited to summation and muliplication for the time being) the learned differential operators and can then equate for nonlinear PDEs. Figure 3.19 is a schematic of *SymNet*, where each layer is given all the differential operators identified by PDE-Net and output a the multiplication of a number of differential operators, this new term is passed to the next layer. In the last layer the coefficients that parameterize each differential operator are determined. This extension enables PDE-Net to learn nonlinear PDEs and motivates future work in using it to learn the NSE.

However, as seen for the turbulent flow cases from PINN, learning two scalars cannot capture the turbulence. As explained previously turbulence is a flow's property and not a fluid's property. Meaning that there is less dependency on the fluid's kinematic viscosity ν and the flow motion is dominated by the eddy viscosity ν_t. This explains the experiments being limited to laminar flow (Re=100) in the PINN works [65], [63], and [62]. Given the relevancy of eddy viscosities for turbulent flows, a separate Machine Learning model can be used for its modelling. In fact, as ν_t is modeled using the turbulent kinetic energy, and the turbulent dissipation as introduced in section 2.2.3, a PINN can be used used to learn the models for both the turbulent kinetic energy and dissipation rate, then subsequently construct the eddy viscosity ν_t. This can mitigate the problem when using one scalar λ_2 to quantify the turbulence, this case fails due to inferring a summation of the kinematic viscosity and an average over space and time of the eddy viscosity. Recently, there has been surrogate Machine Learning method for RANS modeling acceleration [47] that can predict the pointwise steady-state turbulent eddy viscosities, which can be utilised not only for steady-state eddy viscosity but for time increment changes in said turbulent viscosities. A final discussion point, despite

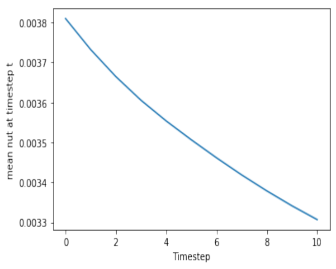


(a) The learned λ_1 parameter for the 3D airfoil case. The true λ_1=1.0.





(b) The learned λ_2 parameter for the 3D airfoil case. (c) The PINN loss for the 3D airfoil case.



(d) The space averaged turbulent viscosity for the 3D airfoil case.

Figure 3.18: λ_1, λ_2, loss value, and averaged turbulent viscosity for the 3D airfoil case.

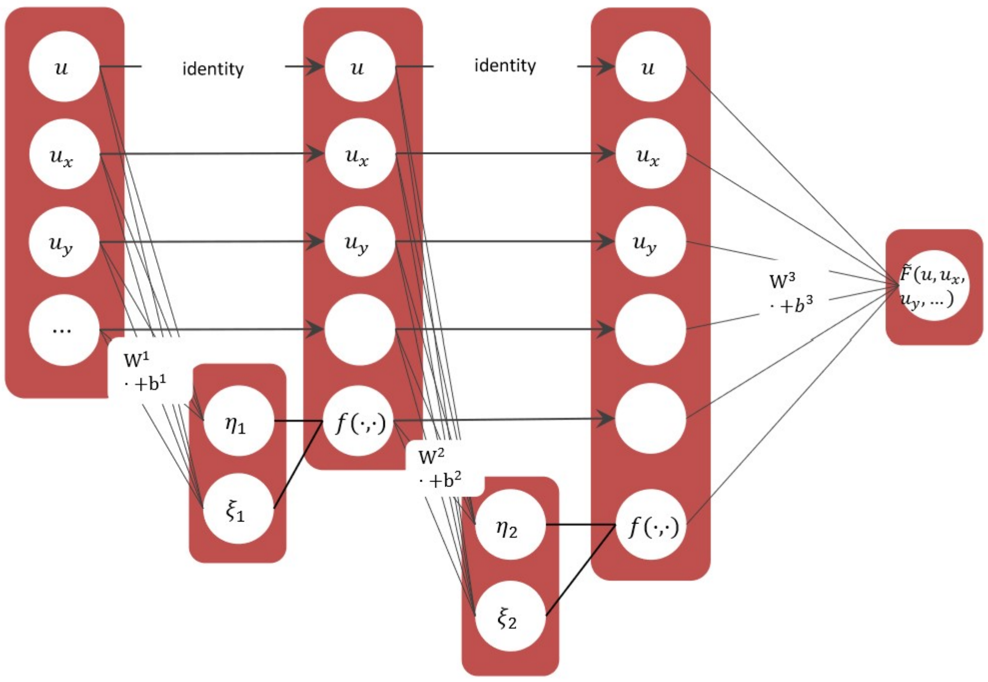


Figure 3.19: *SymNet* as defined for PDE-Net 2.0 [40].

RANS simulations being computationally cost effective and widely used, the Reynolds averaging approach introduces a simplification that cannot be applied for experimental data, as the fluctuation terms cannot be computed by an equation as assumed for RANS. Meaning that for higher fidelity simulations, the RANS model solution should be compared with experimental data, or one should turn to LES or DNS simulations, which solve most (all in the case of DNS) the time and space dependencies in 3D. One can conclude that in the case of turbulent flows in 3D, high fidelity simulation data is needed and a method that learns time and spatially varying coefficients is needed.

4 Conclusion

This work is the intersection of multiple fields of mathematics, engineering, physics, and Machine Learning. It brings together calculus, numerical methods, fluid dynamics, computational fluid dynamics, and Deep Learning. In calculus, PDEs as the equations that govern system dynamics were introduced, and different methods to analytically and numerically solving them. As it is difficult to compute closed form solutions of PDEs, numerical methods have gained popularity to approximate the solutions of Fluid Dynamics problems, since they provide reliable and consistent results with experimental results. Here the method used is computational fluid dynamics (CFD).
This work focuses on turbulent flows, which are a central topic of fluid dynamics in general and fluid mechanics in particular. In chapter 2, an introduction into turbulent flows was given, where the PDE describing turbulent flows and the simplifications that made it possible to obtain approximations of the PDE solution for turbulent flows from the assumption of incompressibility to the different turbulence models were introduced. Then, the state of the art deep learning models for solving and learning partial differential equations were introduced. In chapter 3, after defining the data generation process, different results of the trained deep learning models when learning the PDEs in question (Diffusion equation and Navier-Stokes equations) were reported.
The Navier-Stokes equations are the PDEs that describe turbulent flows. In the context of CFD, turbulence models introduce closure equations to be able to solve the NSE. As an example of the Reynolds-Averaged Navier Stokes turbulence models, is the $k - \varepsilon$ model, which adds two equations modeling the turbulent kinetic energy, and turbulent dissipation rate. These two additional models help define a turbulent viscosity that describes the formation of turbulent eddies.
Recently, Machine Learning in general, and Deep Learning in particular have been applied in different fields. After the success of deep neural networks in computer vision and function approximation, research has emerged in applying ML for PDEs. The application of ML for PDEs takes two forms, either solving the PDEs, or identifying the PDEs from experimental or numerical solutions.
For solving PDEs, CNNs have been mainly used for supervised function approximation from a given set of boundary and initial conditions. The class of physics informed neural networks has been used to infer the solutions of PDEs from a small number of available data points. As neural networks are known to be universal function approximators,

PINNs results in data-efficient neural networks that encode underlying physical laws in its loss function. Recently, the universal approximation theorem for operators, which states that neural networks are operator approximators, and is a generalization of the universal approximation theorem for functions, was utilized to introduce a new class of neural networks the DeepONets, which implicitly approximates the operator that governs the given data, and as a result infers the solutions to partial differential equations.
Identifying ODEs from data was mainly a task of applied mathematics for the first works on this topics. PDE functional identification of nonlinear dynamics (PDE-FIND) utilized sparse regression in a similar manner to SINDy to identify ordinary differential equations that describe reduced PDE solution. High-dimensionality being one of the most important properties of PDE, called to developing data-driven methods that do not modify the PDE solutions, only in this manner can the partial differential equations be uncovered.

The thesis focused on studying and evaluating two models that learn PDEs in two different approaches. PDE-Net approximates the differential operators using convolutional kernels, then construct the equations by learning the associated coefficients.PINNs can learn the parameters of given parameterized nonlinear partial differential equations.
This work focused mainly on using PDE-Net and PINN to discover the equations that govern dynamic models. As the training data was obtained from computational fluid dynamics solvers, the governing equations are known and can serve as knowledge to compare the resulting PDEs with the true ones. As diffusion is a critical part of turbulence formation, PDE-Net succeeds in learning the diffusion equations, however the limitation of this implementation is its limited application to linear PDEs. This limitation is reported to be mitigated using symbolic regression, in a fellow-up work, which was not tested in the scope of this work. PINNs succeed in uncovering the momentum equations for 2D laminar and wake turbulent periodic flows. However, for turbulent flows in 3D, the problem of uncovering the momentum equations and turbulence models is more complex due to the addition of an eddy viscosity, which is a spatio-temporal varying coefficient. In fact, a PINN can be used to learn the equations for the turbulent kinetic energy and the turbulent dissipation rate, which are used to calculate the eddy viscosity in the used turbulence model. Other methods have been used to learn the pointwise eddy viscosity from the PDE solutions [47]. Future work can bring together PINN with the previously mentioned ML surrogate models to identify the PDE for each spatio-temporal point in function of the eddy viscosity. To test this blended model, a number of points outside the defined area for training data can be chosen and their eddy viscosity values can be predicted then the learned PINN can be used to infer the solution of the PDEs. Finally, the inferred solution is to be compared with the true solution.

As a final point, an observation that can be made while reviewing the literature for this work is the difference in quantity between the research on solving PDEs in comparison with learning PDEs using machine learning. While the former seems to gain more attention from the research community, the latter is rather limited to a limited number of deep learning methods, PINNs being the most famous of them. In fact, the introduction of DeepONets [43] hints to the shift towards *calculus-agnostic* methods, which do not resort to prior calculus knowledge, but train deep neural networks to learn nonlinear operators implicitly. Despite the possibility of these models to express complex systems using experimental or numerical data, formulating governing equations, especially for emerging disciplines, such as social dynamics, is of great importance for mathematicians to be able to prove the existence and smoothness of solutions mathematically. With this being said, future work can use the newer version of PDE-Net [40] to learn the NSE of laminar flows to first get an overview of the parameters needed to learn the nonlinear PDE, in a second step, as convolutional filters are used for this model, and the original work have used space varying convection terms, one can attempt to use this property to learn the varying eddy viscosity.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. "Tensorflow: A system for large-scale machine learning." In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] Y. Afshar, S. Bhatnagar, S. Pan, K. Duraisamy, and S. Kaushik. "Prediction of aerodynamic flow fields using convolutional neural networks." In: (2019). DOI: `https://doi.org/10.48550/arXiv.1905.13166`.

[3] *Anaconda Software Distribution*. Version Vers. 2-2.4.0. 2020.

[4] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. "Learning data driven discretizations for partial differential equations." en. In: *Proceedings of the National Academy of Sciences* 116.31 (July 2019). arXiv: 1808.04930, pp. 15344–15349. ISSN: 0027-8424, 1091-6490. DOI: `10.1073/pnas.1814058116`.

[5] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. "Automatic differentiation in machine learning: a survey." en. In: *arXiv:1502.05767 [cs, stat]* (Feb. 2018). arXiv: 1502.05767.

[6] J. Bernstein. *A Palette of Particles*. Harvard University Press, 2013. ISBN: 9780674072510.

[7] M. Bode, M. Gauding, Z. Lian, D. Denker, M. Davidovic, K. Kleinheinz, J. Jitsev, and H. Pitsch. "Using physics-informed enhanced super-resolution generative adversarial networks for subfilter modeling in turbulent reactive flows." en. In: *Proceedings of the Combustion Institute* 38.2 (2021), pp. 2617–2625. ISSN: 15407489. DOI: `10.1016/j.proci.2020.06.022`.

[8] J. Bongard and H. Lipson. "Automated reverse engineering of nonlinear dynamical systems." In: *Proceedings of the National Academy of Sciences* 104.24 (2007), pp. 9943–9948. DOI: `10.1073/pnas.0609476104`. eprint: `https://www.pnas.org/doi/pdf/10.1073/pnas.0609476104`.

[9] S. L. Brunton, J. L. Proctor, and J. N. Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems." en. In: *Proceedings of the National Academy of Sciences* 113.15 (Apr. 2016), pp. 3932–3937. ISSN: 0027-8424, 1091-6490. DOI: `10.1073/pnas.1517384113`.

[10] J.-F. Cai, B. Dong, S. Osher, and A. Shen. "Image Restoration: Total Variation, Wavelet Frames and Beyond." In: *Journal of the American Mathematical Society* 25 (Oct. 2012), pp. 1033–1089. DOI: 10.2307/23265126.

[11] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. "Data-driven discovery of coordinates and governing equations." en. In: *Proceedings of the National Academy of Sciences* 116.45 (Nov. 2019), pp. 22445–22451. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1906995116.

[12] S. Chandrasekhar. "Stochastic Problems in Physics and Astronomy." In: *Rev. Mod. Phys.* 15 (1 Jan. 1943), pp. 1–89. DOI: 10.1103/RevModPhys.15.1.

[13] J. Chen, J. Viquerat, and E. Hachem. *U-net architectures for fast prediction in fluid mechanics*.

[14] T. Chen and H. Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems." In: (1995). DOI: https://doi.org/10.1109/72.392253.

[15] J. P. Crutchfield and B. S. McNamara. "Equations of Motion from a Data Series." In: *Complex Systems 1* (1987), pp. 417–452.

[16] B. Debus, H. Parastar, P. Harrington, and D. Kirsanov. "Deep learning in analytical chemistry." en. In: (2021). DOI: https://doi.org/10.1016/j.trac.2021.116459.

[17] B. Dong, Q. Jiang, and Z. Shen. "Image Restoration: Wavelet Frame Shrinkage, Nonlinear Evolution PDEs, and Beyond." In: *Multiscale Modeling & Simulation* 15.1 (2017), pp. 606–660. DOI: 10.1137/15M1037457.

[18] M. Eckert. *The Dawn of Fluid Dynamics: A Discipline Between Science and Technology*. Jan. 2006. ISBN: 978-3-527-40513-8. DOI: 10.1002/9783527610730.

[19] *Euler Equations*. https://www.grc.nasa.gov/WWW/k-12/rocket/eulereqs.html.

[20] J. H. Ferziger, M. Perić, and R. L. Street. *Computational Methods for Fluid Dynamics*. 2020.

[21] B. Font, G. D. Weymouth, V.-T. Nguyen, and O. R. Tutty. "Deep learning of the spanwise-averaged Navier-Stokes equations." en. In: *Journal of Computational Physics* 434 (June 2021). arXiv: 2008.07528, p. 110199. ISSN: 00219991. DOI: 10.1016/j.jcp.2021.110199.

[22] *Gas properties*. https://www.grc.nasa.gov/WWW/k-12/rocket/gasprop.html.

[23] P. R. Guido Visconti. *Fluid Dynamics*. Springer Cham. DOI: https://doi.org/10.1007/978-3-030-49562-6.

[24] X. Guo, W. Li, and F. Iorio. "Convolutional Neural Networks for Steady Flow Approximation." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).

[25] Y. Guo, X. Cao, B. Liu, and M. Gao. "Solving Partial Differential Equations Using Deep Learning and Physical Constraints." In: *Applied Sciences* 10.17 (2020). ISSN: 2076-3417. DOI: `10.3390/app10175917`.

[26] K. Hornik, M. StinchCombe, and H. White. "Multilayer Feedforward Networks are Universal Approximators." In: (1989). DOI: `https://doi.org/10.1109/72.392253`.

[27] *How To Keep the Courant Number Below 1?* `https://www.simscale.com/knowledge-base/what-is-a-courant-number/`.

[28] *InterFoam*. `https://openfoamwiki.net/index.php/InterFoam`.

[29] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. "NSFnets (Navier-Stokes Flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations." en. In: *Journal of Computational Physics* 426 (Feb. 2021). arXiv: 2003.06496, p. 109951. ISSN: 00219991. DOI: `10.1016/j.jcp.2020.109951`.

[30] *k-omega Shear Stress Transport (SST)*. `https://www.openfoam.com/documentation/guides/latest/doc/guide-turbulence-ras-k-omega-sst.html`.

[31] A. Kiselev. "Small scales and singularity formation in fluid mechanics." In: (Dec. 2018). DOI: `10.1007/s00332-018-9452-3`.

[32] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. "Machine learning–accelerated computational fluid dynamics." In: *Computational Science – ICCS 2021 pp 373–385* (2021). DOI: `https://doi.org/10.1007/978-3-030-77964-1_29`.

[33] A. Krizhevsky, I. Sutskeve, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks." en. In: *Advances in neural information processing systems* (2017).

[34] *L-BFGS algorithm*. `https://docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html`.

[35] M. Lai. "Deep Learning for Medical Image Segmentation." en. In: (May 2015). DOI: `10.1007/978-3-030-76508-8_21`.

[36] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning." en. In: *Nature* (2015). DOI: `https://doi.org/10.1038/nature14539`.

[37] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. “Fourier Neural Operator for Parametric Partial Differential Equations.” en. In: *arXiv:2010.08895 [cs, math]* (May 2021). arXiv: 2010.08895.

[38] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. “Multipole Graph Neural Operator for Parametric Partial Differential Equations.” en. In: *arXiv:2006.09535 [cs, math, stat]* (Oct. 2020). arXiv: 2006.09535.

[39] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. “Neural Operator: Graph Kernel Network for Partial Differential Equations.” In: (2020). DOI: `https://doi.org/10.48550/arXiv.2003.03485`.

[40] Z. Long, Y. Lu, and B. Dong. “PDE-Net 2.0: Learning PDEs from Data with A Numeric-Symbolic Hybrid Deep Network.” en. In: *Journal of Computational Physics* 399 (Dec. 2019). arXiv:1812.04426 [physics, stat], p. 108925. ISSN: 00219991. DOI: `10.1016/j.jcp.2019.108925`.

[41] Z. Long, Y. Lu, and B. Dong. “Supplementary Materials for PDE-Net.” In: (2018), p. 3.

[42] Z. Long, Y. Lu, X. Ma, and B. Dong. “PDE-Net: Learning PDEs from Data.” en. In: (2018), p. 9.

[43] L. Lu, P. Jin, and G. E. Karniadakis. “DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators.” en. In: *arXiv:1910.03193 [cs, stat]* (Apr. 2020). arXiv: 1910.03193.

[44] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators.” en. In: *Nature Machine Intelligence* 3.3 (Mar. 2021), pp. 218–229. ISSN: 2522-5839. DOI: `10.1038/s42256-021-00302-5`.

[45] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis. “A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data.” In: (2021). DOI: `https://doi.org/10.48550/arXiv.2111.05512`.

[46] G. Martius and C. H. Lampert. *Extrapolation and learning equations*. en. Number: arXiv:1610.02995 arXiv:1610.02995 [cs]. Oct. 2016.

[47] R. Maulik, H. Sharma, S. Patel, B. Lusch, and E. Jennings. “Accelerating RANS turbulence modeling using potential flow and machine learning.” In: (Oct. 2019).

[48] P. P. Maziar Raissi and G. E. Karniadakis. “Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations.” In: (2017).

[49] F. Menter, M. Kuntz, and R. Langtry. "Ten years of industrial experience with the SST turbulence model." In: *Heat and Mass Transfer* 4 (Jan. 2003).

[50] *Navier-Stokes Equations 3-dimensional-unsteady*. `https://www.grc.nasa.gov/WWW/k-12/rocket/nseqs.html`.

[51] F. T. M. Nieuwstadt, B. J. Boersma, and J. Westerweel. *The Characteristics of Turbulence*. Cham: Springer International Publishing, 2016, pp. 55–74. ISBN: 978-3-319-31599-7. DOI: `10.1007/978-3-319-31599-7_4`.

[52] *OpenFOAM*. `https://www.openfoam.com/`.

[53] *OpenFOAM v9 User Guide*. `https://doc.cfd.direct/openfoam/user-guide-v9/`.

[54] *OpenFOAM-pimpleFOAM-Rectangular$_{C}$ylinder − main*. `https://github.com/Interfluo/OpenFOAM-Cases-Interfluo/`.

[55] D. S. N. Pachpute. *An Introduction To Computational Fluid Dynamics (CFD)*. `https://cfdflowengineering.com`.

[56] G. Pang, L. Yang, and G. E. Karniadakis. "Neural-net-induced Gaussian process regression for function approximation and PDE solution." en. In: *Journal of Computational Physics* 384 (May 2019). arXiv: 1806.11187, pp. 270–288. ISSN: 00219991. DOI: `10.1016/j.jcp.2019.01.045`.

[57] *ParaView*. `https://www.paraview.org`.

[58] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[59] *PDE-Net*. `https://github.com/ZichaoLong/PDE-Net/blob/PDE-Net/linpdeplot.py`.

[60] *PINNs*. `https://github.com/maziarraissi/PINNs`.

[61] *Pressure in OpenFOAM*. URL: `https://www.cfd-online.com/Forums/openfoam-solving/190173-pressure-openfoam.html`.

[62] M. Raissi. *Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations*. en. arXiv:1801.06637 [cs, math, stat]. Jan. 2018.

[63] M. Raissi and G. E. Karniadakis. "Hidden Physics Models: Machine Learning of Nonlinear Partial Differential Equations." en. In: *Journal of Computational Physics* 357 (Mar. 2018). arXiv:1708.00588 [cs, math, stat], pp. 125–141. ISSN: 00219991. DOI: 10.1016/j.jcp.2017.11.039.

[64] M. Raissi and G. E. Karniadakis. "Machine Learning of Linear Differential Equations using Gaussian Processes." en. In: *Journal of Computational Physics* 348 (Nov. 2017). arXiv: 1701.02440, pp. 683–693. ISSN: 00219991. DOI: 10.1016/j.jcp.2017.07.050.

[65] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations." en. In: *arXiv:1711.10566 [cs, math, stat]* (Nov. 2017). arXiv: 1711.10566.

[66] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations." en. In: (2017). DOI: https://doi.org/10.1016/j.trac.2021.116459.

[67] A. K. Razdan and V. Ravichandran. "Fundamentals of Partial Differential Equations." In: (2022). DOI: https://doi.org/10.1007/978-981-16-9865-1.

[68] S. Rodriguez. *Applied Computational Fluid Dynamics and Turbulence modeling: Practical Tools, Tips and Techniques*. Springer Cham. DOI: https://doi.org/10.1007/978-3-030-28691-0.

[69] K. Rojek, R. Wyrzykowski, and P. Gepner. "AI-Accelerated CFD Simulation Based on OpenFOAM and CPU/GPU Computing." In: *Computational Science – ICCS 2021 pp 373–385* (2021). DOI: https://doi.org/10.1007/978-3-030-77964-1_29.

[70] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. "Data-driven discovery of partial differential equations." en. In: *arXiv:1609.06401 [nlin]* (Sept. 2016). arXiv: 1609.06401.

[71] *ScalarTransportFoam*. https://openfoamwiki.net/index.php/ScalarTransportFoam.

[72] M. Schmidt and H. Lipson. "Distilling Free-Form Natural Laws from Experimental Data." In: *Science* 324.5923 (2009), pp. 81–85. DOI: 10.1126/science.1165893. eprint: https://www.science.org/doi/pdf/10.1126/science.1165893.

[73] M. Shinbrot. *concerning the Navier-Stokes equations*. http://www.navier-stokes-equations.com/. 2012.

[74] *Simulation of Turbulent Flows*. http://web.stanford.edu/class/me469b/handouts/turbulence.pdf.

[75] C. Soulaine. *Introduction to open-source computational fluid dynamics using OpenFOAM®technology*. https://www.cypriensoulaine.com/openfoam.

[76] *The PISO algorithm in OpenFOAM*. https://openfoamwiki.net/index.php/OpenFOAM_guide/The_PISO_algorithm_in_OpenFOAM.

[77] *The SIMPLE algorithm in OpenFOAM*. https://openfoamwiki.net/index.php/OpenFOAM_guide/The_SIMPLE_algorithm_in_OpenFOAM.

[78] N. Thuerey, K. Weissenow, L. Prantl, and X. Hu. "Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows." en. In: (2018). DOI: https://doi.org/10.48550/arXiv.1810.08217.

[79] K. Um, R. Brand, Yun, Fei, P. Holl, and N. Thuerey. "Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers." en. In: *arXiv:2007.00016 [physics]* (Jan. 2021). arXiv: 2007.00016.

[80] S. Wang and P. Perdikaris. "Long-time integration of parametric evolution equations with physics-informed DeepONets." en. In: *arXiv:2106.05384 [physics]* (June 2021). arXiv: 2106.05384.

[81] S. Wang, H. Wang, and P. Perdikaris. "Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets." en. In: *arXiv:2103.10974 [cs, math, stat]* (Mar. 2021). arXiv: 2103.10974.

[82] L. P. Williams. *history of science*. https://www.britannica.com/science/history-of-science. Feb. 2022.

[83] H. You, Y. Yu, M. D'Elia, T. Gao, and S. Silling. "Nonlocal Kernel Network (NKN): a Stable and Resolution-Independent Deep Neural Network." In: (2022). DOI: https://doi.org/10.48550/arXiv.2201.02217.

[84] A. Yu, C. Becquey, D. Halikias, M. E. Mallory, and A. Townsend. "Arbitrary-Depth Universal Approximation Theorems For Operator Neural Networks." In: (2021). DOI: arXiv:2109.11354v1[cs.LG].