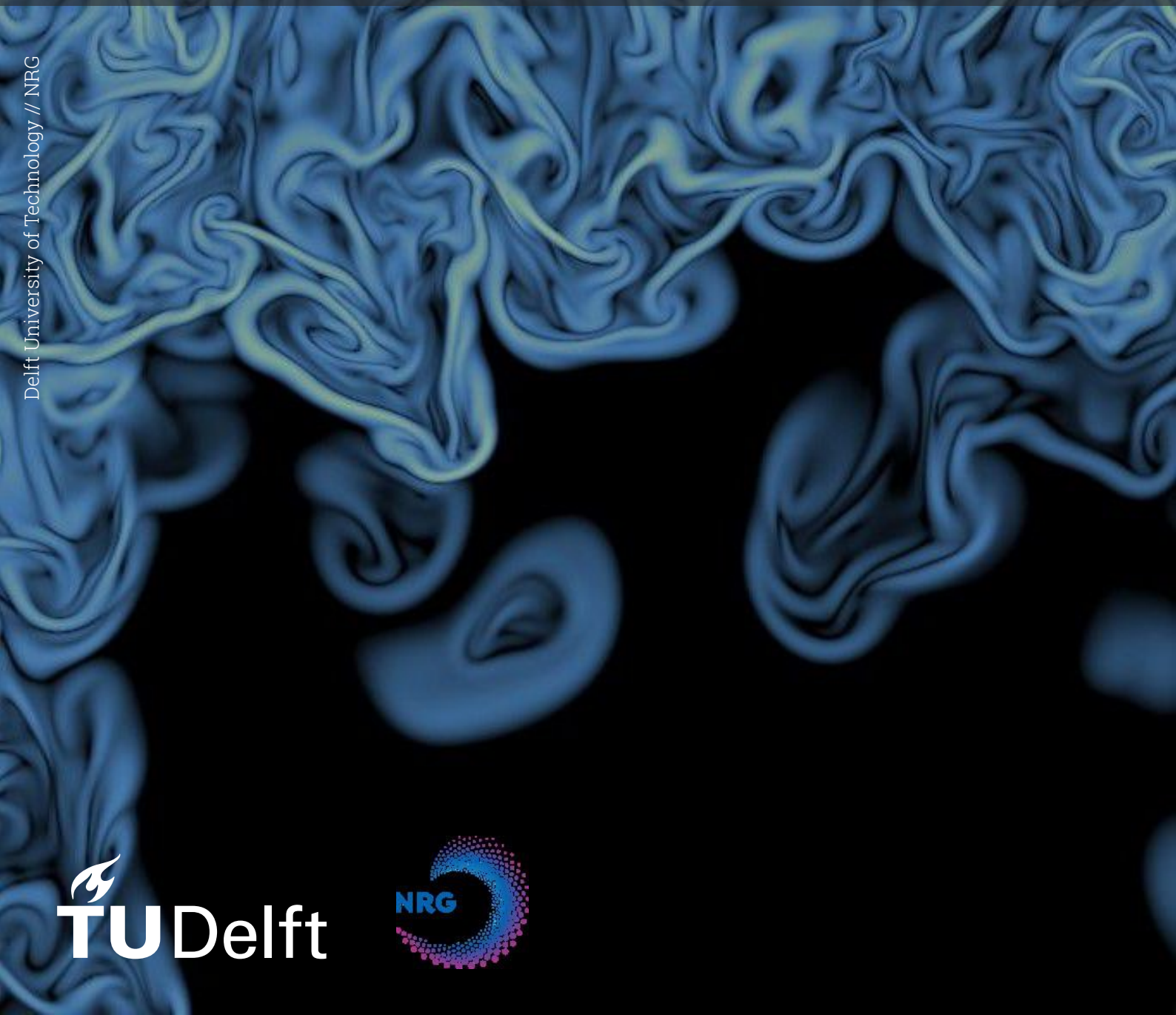
Data-driven turbulence modeling of two-phase flows in nuclear reactors

G. Bonilla



Delft University of Technology // NRG

TU Delft

NRG

Data-driven turbulence modeling of two-phase flows in nuclear reactors

by

G. Bonilla

Student Name	Student Number
G. Bonilla	5850177

NRG	Supervisor:	Dr.ir. E.M.A. (Edo) Frederix
TU Delft	Supervisor:	Dr. D. (Deepesh) Toshniwal
TU Delft	Supervisor:	Dr. A. (Alexander) Heinlein
Project Duration:		November, 2023 - July, 2024
Faculty:		EEMCS

Cover: Fluid mechanics wallpaper from Wallpapercave.

Contents

1 Introduction

In nuclear engineering, understanding the behaviour of multiphase flows is crucial. Two-phase flow occurs in different processes, from how coolants move in nuclear reactors to analyzing safety scenarios with different fluid phases. Numerical simulations are often used to study these phenomena since they provide a tool to analyze scenarios that, for cost or safety reasons, can not be done experimentally, such as nuclear accidents. The equations that describe these flows can be directly solved in what is called direct numerical simulations (DNS). However, for turbulent flows, this approach is too computationally expensive, making it unfeasible for anything but very simple scenarios. Consequently, turbulence modelling is used as a more cost-effective tool to represent turbulent flows in two-phase systems.

The development of turbulence models for multiphase flows is an active area of research. While some models for particular cases are available, single-phase models are often used. However, single-phase models are inadequate due to the presence of the interface between phases, which causes accuracy issues. While some authors have tried to mitigate this issue with conventional modelling, their success has been limited.

In recent years, machine learning has seen a considerable increase in popularity within scientific domains, including the field of fluid mechanics. Moreover, the increased available computational power has also led to a rise in high-fidelity data from turbulent flow numerical simulations. This data can be used to gain a deeper understanding of turbulence, but it can also be used to develop data-driven turbulence models with the help of machine learning techniques. Nevertheless, data-driven turbulence modelling research has mostly focused on single-phase flows.

Given the shortcomings of conventional turbulent models for two-phase flows and the remarkable progress that has been made in single-phase turbulence with scientific machine learning, one possible avenue in the research of two-phase turbulence modelling is to extend the data-driven modelling techniques developed for single-phase flows. Throughout this project, the capabilities and limitations of data-driven turbulence modelling for two-phase flows will be researched with the ultimate aim of finding an algebraic expression that can be used to improve the predictive accuracy of these models without decreasing their robustness. The main question and sub-questions for this research project are as follows:

- Q1: Can machine learning be used to regress a symbolic expression to capture the influence of the interface of turbulence models in two-phase flows?
  - S1: What are the relevant input features for the regression?
  - S2: How does the proposed methodology compare to other damping approaches in terms of robustness, accuracy and generalizability?
  - S3: How does using a neural network model for turbulence damping compare to using a symbolic expression?

The thesis is structured as follows. In chapter 2, the fundamentals of fluid mechanics and turbulence modelling are introduced. Chapter 3 gives an overview of the machine-learning techniques that will be used and summarizes recent research in the field of machine learning for data-driven turbulence modelling. In chapter 4, the machine learning framework used in this project is explained. Chapter 5 presents the numerical solvers employed for the simulations of the different models of the machine learning framework and the test cases used in this project. The results of the thesis are discussed in chapter 6. Finally, the conclusions of this work are outlined in chapter 7.

2

Two-phase flow and turbulence

This chapter provides an overview of fluid mechanics, focusing on single-phase flow, two-phase flow, and turbulence modeling. The sources used as guides include [1] for single-phase flow, [2], [3], and [4] for turbulence modeling, and [5] for multiphase flow.

2.1. Turbulence

This section introduces single-phase turbulence. First, a brief overview of the equations that govern fluids is given. Then, the Reynolds-averaged Navier-Stokes (RANS) turbulence models are introduced, and essential aspects of the Reynolds stress tensor are discussed.

2.1.1. Introduction

Fluids are substances that deform continuously under the action of a shear stress. These substances are formed by particles, but their equations of motion can be formulated using the continuum hypothesis. This hypothesis assumes that at any point in a fluid, properties such as density, velocity, and temperature vary smoothly and continuously, even though the fluid is composed of discrete molecules [1]. The continuum approach simplifies the mathematical description of fluid behaviour, allowing the use of partial differential equations to model fluid flow.

The partial differential equations that model fluid flow are derived from the conservation laws of mass, momentum and energy. In a fluid where no mass is being produced, the conservation of mass equation is given by

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i) = 0, \tag{2.1}$$

where ρ is the fluid density and u_i are the components of the velocity field. In the previous equation, we used Einstein index notation, which means that repeated indices implied summation like

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_1}(\rho u_1) + \frac{\partial}{\partial x_2}(\rho u_2) + \frac{\partial}{\partial x_3}(\rho u_3) = 0.$$

For the remainder of this work, the same index notation will be used unless the indices are expressed using Greek letters such as α, β, etc.

If the fluid is incompressible then the material derivative, which quantifies the variation of a fluid property along the trajectory of a fluid particle, of the density is zero

$$\frac{D\rho}{Dt} = \frac{\partial \rho}{\partial t} + u_i \frac{\partial \rho}{\partial x_i} = 0. \tag{2.2}$$

Therefore, combining equations (2.1) and (2.2) the mass conservation law reduces to

$$\frac{\partial u_i}{\partial x_i} = 0. \tag{2.3}$$

Equation (2.3) is known as the continuity equation and states that the divergence of the velocity field of an incompressible fluid is zero.

In a similar way, the linear momentum of the fluid must be conserved unless a force is applied to it, which can be expressed as

$$\frac{D}{Dt}(\rho u_i) = \frac{\partial \sigma_{ij}}{\partial x_j} + \rho f_i, \tag{2.4}$$

where the first term on the right-hand side refers to the superficial forces and the second one to the given volumetric forces f_i. However, in order to obtain a closed system of equations, the Cauchy stress tensor, σ_{ij}, needs to be expressed in terms of the flow variables. This is done by constitutive equations [1]. For a Newtonian fluid, the constitutive equation is

$$\sigma_{ij} = -pI + 2\mu S_{ij}, \tag{2.5}$$

where p is the pressure, μ is the viscosity of the fluid and S_{ij} is the strain rate tensor of the velocity

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right).$$

Introducing equation (2.5) into (2.4) for fluids with constant density and viscosity yields

$$\rho\left(\frac{\partial u_i}{\partial t} + u_j\frac{\partial u_i}{\partial x_j}\right) = -\frac{\partial p}{\partial x_i}I + \mu\frac{\partial^2 u_i}{\partial x_j^2} + \rho f_i. \tag{2.6}$$

Equation (2.6) is referred to as the momentum equation and, together with the continuity equation, forms a complete set of equations known as the Navier-Stokes equations [1].

Two terms of the momentum equation are particularly important when it comes to understanding turbulence. The first one is the convective term, $\rho u_j \frac{\partial u_i}{\partial x_j}$. It is a non-linear term that represents the inertial forces on the fluid. The other one is the diffusive term, $\mu\frac{\partial^2 u_i}{\partial x_j^2}$, which represents the viscous forces on the fluid. The ratio between the inertial forces and the viscous forces is quantified by the Reynolds number

$$\mathrm{Re} = \frac{\mathcal{U}\mathcal{L}\rho}{\mu},$$

where $\mathcal{U}$ is the characteristic velocity of the flow and $\mathcal{L}$ is the characteristic length scale. These characteristic quantities are representative of the case of interest; for example, for a 2-dimensional channel, $\mathcal{L}$ could be the height of the channel and $\mathcal{U}$ the maximum velocity of the fluid. When the Reynolds number is large, the inertial forces are larger than the viscous forces, and the fluid behaves chaotically, which is known as turbulence.

Turbulent flow can be simulated using the Navier-Stokes equations in what is known as direct numerical simulations (DNS). However, due to the different ranges of spatial and temporal scales which would need to be solved this is computationally not feasible in most applications. Turbulent flow has a macrostructure formed by large eddies, which is where energy gets generated. The large eddies transfer their energy to smaller eddies and so on until the microstructure is formed by the smallest eddies. It is in this microstructure where the energy gets dissipated due to diffusion. The Kolmogorov hypothesis states that the scales of the microstructure can be related to the kinematic viscosity, $\nu = \mu/\rho$, and the energy dissipation rate ϵ. These are the characteristic length $\eta = \left(\nu^3/\epsilon\right)^{1/4}$, time $\tau = (\nu/\epsilon)^{1/2}$ and velocity $v = (\nu\epsilon)^{1/4}$ of the Kolmogorov scales.

In order to perform an accurate DNS, all the microstructure phenomena must be captured by the simulation. Therefore, the mesh size and timestep must be of the same order as η and τ, respectively. The number of cells needed for a 3-dimensional domain of size $\mathcal{L}^3$ is roughly

$$\mathcal{N} \sim \left(\frac{\mathcal{L}}{\eta}\right)^3 \sim \mathcal{O}\left(\mathrm{Re}^{\frac{9}{4}}\right),$$

and the number of timesteps for a simulation of $\mathcal{T}$ time units I

$$\mathcal{M} \sim \frac{\mathcal{T}}{\tau} \sim \mathcal{O}\left(\mathrm{Re}^{\frac{1}{2}}\right).$$

The total computational power needed will scale with the product of the number of cells by the number of timesteps which almost scales with the cube of the Reynolds number. For turbulent flows with large Reynolds numbers, performing DNS is unfeasible.

$$\mathcal{N} \times \mathcal{M} \sim \mathcal{O}\left(\mathrm{Re}^{\frac{11}{4}}\right).$$

Therefore, turbulence models have been developed to allow for simulations of turbulent flows with a much smaller computational cost. One of the most used types of turbulent models are RANS models, which are introduced in the next section.

2.1.2. RANS

Reynolds-averaged Navier-Stokes models are a class of modelling techniques used to simulate turbulent flows. These models are based on the Reynolds-averaging process applied to the Navier-Stokes equations. Each of the flow variables is decomposed in a mean value $\bar{\varphi}$ and an instantaneous fluctuation φ':

$$\varphi = \bar{\varphi} + \varphi'.$$

To be able to apply the averaging operator to the Navier Stokes equations the averaging operator has to satisfy a set of conditions known as the Reynolds conditions [3]. This means that averaging operators such as the temporal average $\bar{\cdot}^T(t)$, for a time T, or the spatial average $\bar{\cdot}^L(t)$, for a length L, are not suitable for this purpose.

$$\bar{\varphi}^T(t) = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} \varphi(t+\tau)d\tau, \qquad \bar{\varphi}^L(x) = \frac{1}{L}\int_{-\frac{L}{2}}^{\frac{L}{2}} \varphi(x+\xi)d\xi.$$

Instead, the ensemble average is used:

$$\bar{\varphi} = \lim_{N\to\infty} \frac{1}{N}\sum_{\alpha=1}^{N} \varphi^{(\alpha)}$$

This averaging process can be interpreted as follows. Imagine running the same experiment with small perturbations N times. The result for each of the experiments is $\varphi^{(\alpha)}$, then all the obtained results get averaged. The ensemble average operator can be applied to the momentum equation (2.6). After some simplification, equation (2.7) is reached. Note that while the average of a fluctuation, $\overline{u_i'}$, is zero by definition, the average of the product of fluctuations, $\overline{u_i'u_j'}$, is not and will therefore appear in the averaged equations.

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j\frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\nu\frac{\partial \bar{u}_i}{\partial x_j} - \overline{u_i'u_j'}\right) + \bar{f}_i. \tag{2.7}$$

The obtained equation resembles the original Navier-Stokes equation but with an additional term called the Reynolds stress, $R_{ij} = \overline{u_i'u_j'}$. In order to obtain a closed system of equations, this term needs to be modelled in what is known as the closure problem. Different turbulence models focus on different ways of doing so.

The Reynolds stress can be decomposed into an isotropic (equal in all directions) and an anisotropic part

$$R_{ij} = \frac{2}{3}k\delta_{ij} + a_{ij},$$

where the first term is the isotropic part, a_{ij} is the anisotropic and $k := \frac{1}{2}\mathrm{trace}(R_{ij})$ is the turbulent kinetic energy. The anisotropic part can be non-dimensionalized, yielding the non-dimensional Reynolds stress anisotropy tensor, b_{ij}, which will be further studied later in this section

$$b_{ij} = \frac{R_{ij}}{2k} - \frac{1}{3}\delta_{ij}.$$

From the Navier-Stokes equation, a transport equation can be derived for k

$$\frac{\partial k}{\partial t} + U_j\frac{\partial k}{\partial x_j} = \tau_{ij}\frac{\partial U_i}{\partial x_j} - \epsilon + \frac{\partial}{\partial x_j}\left(\nu\frac{\partial k}{\partial x_j} - \frac{1}{2}\overline{u_i'u_i'u_j'} - \frac{1}{\rho}\overline{p'u_j'}\right),$$

where $U_i = \bar{u}_i$. The left-hand side of this equation is simply the material derivative of the turbulent kinetic energy. The different terms in the right-hand side of this equation are referred to as turbulent kinetic energy budgets [4]. These budgets are an integral tool for understanding turbulent flows.

The first term of the right-hand side, which can be rewritten as $\tau_{ij}S_{ij}$, is known as the production. It represents the energy that is being transferred from the mean flow to the turbulence and can be interpreted as the rate at which work is done by the mean strain rate against the turbulent stresses.

The second term is the dissipation. It represents the rate at which turbulent energy is dissipated into thermal energy and it is given by

$$\epsilon = \nu \overline{\frac{\partial u_i'}{\partial x_k} \frac{\partial u_i'}{\partial x_k}}.$$

The last three terms are transport terms which can only redistribute the turbulent kinetic energy but cannot produce it or destroy it. The first one of those terms is molecular diffusion and represents the diffusion of k caused by molecular transport processes. The second one is turbulence transport, and it is the turbulent energy that gets transported due to turbulent fluctuations in the velocity. The last one is the pressure diffusion and it is caused by pressure fluctuations.

2.1.3. Linear eddy viscosity models

The most widespread class of turbulence models are the linear eddy viscosity models (LEDM), in which the Boussinesq hypothesis is employed [3]. This hypothesis assumes a linear relationship between the anisotropy tensor and the mean strain rate tensor

$$a_{ij} = -\nu_t \left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right) = -2\nu_t S_{ij}.$$

For these models to be closed the turbulent viscosity, ν_t, needs to be specified. Linear eddy viscosity models can be classified depending on the amount of additional equations introduced to close the model. Zero equation models use an algebraic relationship between the turbulent viscosity and the length scales of the mean flow and therefore they cannot account for the history effects of turbulence. One example of such models is given by the following equation

$$\nu_t = l_{\text{mix}}^2 \left| \frac{dU}{dy} \right|,$$

where the second term is the derivative of the mean streamwise flow velocity with respect to the perpendicular direction of the flow, and l_{mix} is the mixing length, which needs to be specified for different flows.

One equation models solve the transport equation for one turbulent quantity, typically for the turbulent kinetic energy or the turbulent viscosity. However, as we saw before with the k equation, these transport equations contain unclosed terms that need to be modelled to obtain a closed system. Models based on the Prandtl mixing length theory solve the transport equation for k and then use

$$\nu_t = c\sqrt{k} l_{\text{mix}},$$

to obtain the turbulent viscosity. Therefore a length scale of the flow needs to be specified and thus the model is incomplete. One popular one-equation model is the Spalart-Allmaras turbulence model [4], where a transport equation for a modified turbulent viscosity is solved. This model does not need the mixing length to be specified and, thus, is complete.

Two equation models solve the transport equations of two turbulence quantities. Two of the most popular turbulence models are the $k - \epsilon$ and $k - \omega$ models [4], where the turbulence kinetic energy equation is solved in conjunction with the dissipation or the dissipation rate, respectively.

In the $k - \epsilon$ model, the turbulent viscosity is obtained by

$$\nu_t = C_\mu \frac{k^2}{\epsilon}.$$

The turbulent quantities transport equations are

$$\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} = \tau_{ij} \frac{\partial U_i}{\partial x_j} - \epsilon + \frac{\partial}{\partial x_j}\left(\left(\nu + \frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right),$$
$$\frac{\partial \epsilon}{\partial t} + U_j \frac{\partial \epsilon}{\partial x_j} = C_{\epsilon 1}\frac{\epsilon}{k}\tau_{ij}\frac{\partial U_i}{\partial x_j} - C_{\epsilon 2}\frac{\epsilon^2}{k} + \frac{\partial}{\partial x_j}\left(\left(\nu + \frac{\nu_t}{\sigma_\epsilon}\right)\frac{\partial \epsilon}{\partial x_j}\right).$$

The model is closed by setting the additional parameters

$$C_{\epsilon 1} = 1.44, \quad C_{\epsilon 2} = 1.92, \quad C_\mu = 0.09, \quad \sigma_k = 1, \quad \sigma_\epsilon = 1.3.$$

Similarly in the $k-\omega$ model, the turbulent viscosity is

$$\nu_t = \frac{k}{\tilde{\omega}}, \quad \tilde{\omega} = \max\left\{\omega, C_{\lim}\sqrt{\frac{2S_{ij}S_{ji}}{\beta^*}}\right\}, \quad C_{\lim} = \frac{7}{8},$$

and the transport equations are

$$\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} = \tau_{ij}\frac{\partial U_i}{\partial x_j} - \beta^*\kappa\omega + \frac{\partial}{\partial x_j}\left(\left(\nu + \sigma^*\frac{\kappa}{\omega}\right)\frac{\partial k}{\partial x_j}\right),$$
$$\frac{\partial \omega}{\partial t} + U_j \frac{\partial \omega}{\partial x_j} = \alpha\frac{\omega}{\kappa}\tau_{ij}\frac{\partial U_i}{\partial x_j} - \beta\omega^2 + \frac{\sigma_d}{\omega}\frac{\partial \kappa}{\partial x_j}\frac{\partial \omega}{\partial x_j} + \frac{\partial}{\partial x_j}\left(\left(\nu + \sigma\frac{\kappa}{\omega}\right)\frac{\partial \omega}{\partial x_j}\right).$$

The model is closed, again, by specifying the additional parameters

$$\alpha = \frac{13}{25}, \quad \beta = \beta_o f_\beta, \quad \beta^* = 0.09, \quad \sigma = 0.5, \quad \sigma^* = 0.6, \quad \sigma_{do} = 0.125,$$

$$\sigma_d = \begin{cases} 0, & \frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j} \leq 0, \\ \sigma_{do}, & \frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j} > 0. \end{cases}$$

2.1.4. Reynolds stress, anisotropy, invariance, and realizability

The Reynolds stress tensor will be one of the modelling targets in this project, so we will present important properties about it. In this section, the invariance properties and realizability constraints of this tensor are discussed.

From the definition of the Reynolds stress tensor and the ensemble average, it follows that R_{ij} is a semi-definite positive matrix, that is,

$$x_i R_{ij} x_j = x_i \overline{u_i' u_j'} x_j = \lim_{N\to\infty}\frac{1}{N}\sum_{\alpha=1}^{N} x_i u_i'^{(\alpha)} u_j'^{(\alpha)} x_j = \lim_{N\to\infty}\frac{1}{N}\sum_{\alpha=1}^{N}\left(x_i u_i'^{(\alpha)}\right)^2 \geq 0.$$

Therefore, the eigenvalues of the Reynolds stress will be real and non-negative. The determinant, trace, and diagonal entries of a matrix will also be non-negative. Furthermore, it can be shown that the Cauchy inequality must hold. In summary, we have that

$$R_{\alpha\alpha} \geq 0, \quad \det(R_{ij}) \geq 0, \quad R_{\alpha\beta}^2 \leq R_{\alpha\alpha}R_{\beta\beta}.$$

According to the spectral theorem, the Reynolds stress can be diagonalized since it is a semi-definite positive matrix. From this, it follows that

$$R_{ij} = 2k\left(\frac{1}{3}\delta_{ij} + V_{ik}\Lambda_{kl}V_{jl}\right),$$

where V_{ij} is a matrix containing the eigenvectors of the anisotropy tensor and Λ_{ij} is a diagonal matrix with the eigenvalues of the anisotropy tensor, λ_i. Moreover, the eigenvalues of the anisotropy tensor can be related to those of the Reynolds stress

$$\lambda_i = \frac{\phi_i}{2k} - \frac{1}{3},$$

where ϕ_i are the eigenvalues of the Reynolds stress. Using the previously mentioned properties, it follows that these eigenvalues must lie between 0 and $2k$ and that the eigenvalues of the anisotropy tensor are between $-1/3$ and $2/3$. Additionally, the components of b_{ij} must satisfy the following restrictions

$$-\frac{1}{3} \leq b_{\alpha\alpha} \leq \frac{2}{3}, \qquad -\frac{1}{2} \leq b_{\alpha\beta} \leq \frac{1}{2}, \qquad \forall \alpha \neq \beta.$$

There exist different ways to visualize the anisotropy tensor state The two most popular are using the invariant map and the barycentric map. The invariant map uses the second and third invariant of the anisotropy tensor, $\text{II} = b_{ij}b_{ji}$ and $\text{III} = b_{ij}b_{ik}b_{jk}$, to plot all possible states in the II-III plane. Using the restrictions the possible states fall within a triangle in this plane.

The barycentric map was introduced by [6] and uses the three limiting states of turbulence to represent the anisotropy tensor. These three limiting states are:

$$b_{ij1c} = \begin{pmatrix} \frac{2}{3} & 0 & 0 \\ 0 & -\frac{1}{3} & 0 \\ 0 & 0 & -\frac{1}{3} \end{pmatrix}, \qquad b_{ij2c} = \begin{pmatrix} \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{6} & 0 \\ 0 & 0 & -\frac{1}{3} \end{pmatrix}, \qquad b_{ij2c} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

- 1 component turbulence b_{ij1c}; only on eigenvalue of the Reynolds stress is non zero and turbulence occurs in the direction of the eigenvector associated to this eigenvalue.
- 2 component turbulence b_{ij2c}; two eigenvalues of the Reynolds stress are non zero, and turbulence occurs in the plane spanned by the eigenvectors associated with these eigenvalues.
- 3 component turbulence (isotropic turbulence) b_{ij3c}; the anisotropy tensor is zero.

Any realizable state of the anisotropy tensor can be obtained as a linear combination of the anisotropy tensor associated with the three limiting states,

$$b_{ij} = C_{1c}b_{ij1c} + C_{2c}b_{ij2c} + C_{3c}b_{ij3c},$$

where the coefficients are obtained directly from the eigenvalues of b_{ij}

$$C_{1c} = \lambda_1 - \lambda_2, \qquad C_{2c} = 2(\lambda_2 - \lambda_3), \qquad C_{3c} = 3\lambda_3 + 1.$$

Then each limiting state is placed in a vertex of a triangle with coordinates (x_{ic}, y_{ic}). The coordinates of the anisotropy tensor are simply obtained by

$$(x, y) = C_{ic}(x_{ic}, y_{ic}).$$

Thanks to the linearity of this representation, it is a simple tool to enforce relizeability of the anisotropy tensor by forcing the representation of this tensor to lie inside this triangle. Figure 2.1 shows the barycentric triangle representation of the anisotropy stress, where each corner of the triangle represents one of the limiting states. While all states inside this triangle are physically possible only the ones represented by the plane strain line are possible under the Boussinesq hypothesis.

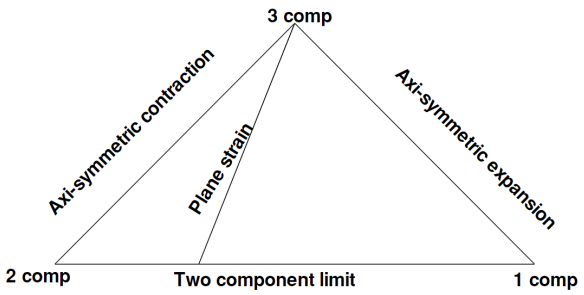


Figure 2.1: Barycentric triangle representation of the anisotropy stress, obtained from [6, p. 10]

While linear eddy viscosity models are the most commonly used, non-linear eddy viscosity models (NLEVM) can also be used. In [2] Pope presented a more general representation of the anisotropy tensor in which it depends on the normalized strain rate and rotation rate tensors

$$\hat{S}_{ij} = \tau \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right), \qquad \hat{\Omega}_{ij} = \tau \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} - \frac{\partial U_j}{\partial x_i}\right),$$

where $\tau = 1/\omega$ is a turbulent timescale. Then, with the use of the Cayley-Hamilton theorem, the anisotropy can be represented as

$$b_{ij} = \sum_{n=1}^{10} T_{ij}^{(m)} \left(\hat{S}_{ij}, \hat{\Omega}_{ij}\right) g^{(m)}\left(\lambda_1, \ldots, \lambda_5\right),$$

where $T_{ij}^{(m)}$ is a set of ten linearly independent symmetric tensors with zero trace, λ_i is a set of five invariants and $g^{(m)}$ some unknown functions. The basis tensors are

$$\begin{aligned}
T_{ij}^{(1)} &= \hat{S}_{ij,} & T_{ij}^{(6)} &= \hat{\Omega}_{ik}\hat{\Omega}_{kl}\hat{S}_{lj} + \hat{S}_{ik}\hat{\Omega}_{kl}\hat{\Omega}_{lj} - \frac{2}{3}\hat{S}_{pk}\hat{\Omega}_{kl}\hat{\Omega}_{lp}\delta_{ij},\\
T_{ij}^{(2)} &= \hat{S}_{ik}\hat{\Omega}_{kj} - \hat{\Omega}_{ik}\hat{S}_{kj}, & T_{ij}^{(7)} &= \hat{\Omega}_{ik}\hat{S}_{kl}\hat{\Omega}_{lp}\hat{\Omega}_{pj} - \hat{\Omega}_{ik}\hat{\Omega}_{kl}\hat{S}_{lp}\hat{\Omega}_{pj},\\
T_{ij}^{(3)} &= \hat{S}_{ik}\hat{S}_{kj} - \frac{1}{3}\hat{S}_{lk}\hat{S}_{kl}\delta_{ij}, & T_{ij}^{(8)} &= \hat{S}_{ik}\hat{\Omega}_{kl}\hat{S}_{lp}\hat{S}_{pj} - \hat{S}_{ik}\hat{S}_{kl}\hat{\Omega}_{lp}\hat{S}_{pj},\\
T_{ij}^{(4)} &= \hat{\Omega}_{ik}\hat{\Omega}_{kj} - \frac{1}{3}\hat{\Omega}_{lk}\hat{\Omega}_{kl}\delta_{ij}, & T_{ij}^{(9)} &= \hat{\Omega}_{ik}\hat{\Omega}_{kl}\hat{S}_{lp}\hat{S}_{pj} + \hat{S}_{ik}\hat{S}_{kl}\hat{\Omega}_{lp}\hat{\Omega}_{pj} - \frac{2}{3}\hat{S}_{qk}\hat{S}_{kl}\hat{\Omega}_{lp}\hat{\Omega}_{pq},\\
T_{ij}^{(5)} &= \hat{\Omega}_{ik}\hat{S}_{kl}\hat{S}_{lj} - \hat{S}_{ik}S_{kl}\hat{\Omega}_{lj}, & T_{ij}^{(10)} &= \hat{\Omega}_{ik}\hat{S}_{kl}\hat{S}_{lp}\hat{\Omega}_{pq}\hat{\Omega}_{qj} - \hat{\Omega}_{ik}\hat{\Omega}_{kl}\hat{S}_{lp}\hat{S}_{pq}\hat{\Omega}_{qj},
\end{aligned}$$

and the invariants are

$$\begin{aligned}
\lambda_1 &= \hat{S}_{ij}\hat{S}_{ji}, & \lambda_4 &= \hat{\Omega}_{ij}\hat{\Omega}_{jk}\hat{S}_{ki},\\
\lambda_2 &= \hat{\Omega}_{ij}\hat{\Omega}_{ji}, & \lambda_5 &= \hat{\Omega}_{ij}\hat{\Omega}_{jk}\hat{S}_{kl}\hat{S}_{li}.\\
\lambda_3 &= \hat{S}_{ij}\hat{S}_{jk}\hat{S}_{ki}, &&
\end{aligned}$$

This formulation has the advantage of meeting the invariance properties of the Navier Stokes equations regardless of the form the functions $g^{(m)}$ have.

2.2. Multiphase flow

Multiphase flows refer to fluid systems in which two or more phases exist and interact with each other. These combinations of phases can take various forms, e.g., gas-liquid, liquid-solid, liquid-liquid, or other combinations. In this work, we are particularly interested in stratified gas-liquid two-phase flows that often occur in nuclear reactors. The equations governing these systems are similar to those of single-phase fluids, with some additional complexities.

The conservation of mass equations for immiscible incompressible two-phase flows is the same as for incompressible single-phase flows:

$$\frac{\partial u_j}{\partial x_j} = 0.$$

The conservation of momentum equation is also similar but with some additional terms

$$\frac{\partial u_i}{\partial t} + u_j\frac{\partial u_i}{\partial x_j} = \frac{1}{\rho}\left(-\frac{\partial p}{\partial x_i}I + \frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right) + f_i^{\sigma}\right) + f_i.$$

The term

$$f_i^{\sigma} = \sigma\kappa\delta(x_j - x_j')n_i,$$

models the surface tension, where σ is the surface tension coefficient which is a material property, κ is the curvature of the interface between the two fluids, n_i is the normal direction of the interface and

$\delta(x_j - x'_j)$ is the Dirac function, used because the surface tension is a force that only appears in the interface [7].

It is also important to note that, since the viscosity is no longer constant (due to the two phases having different properties), a new term arises from the viscous term.

$$\frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)\right)=\frac{\partial}{\partial x_j}\left(\mu\frac{\partial u_i}{\partial x_j}\right)+\frac{\partial \mu}{\partial x_j}\frac{\partial u_j}{\partial x_i}. \tag{2.8}$$

Additionally, it is necessary to keep track of where the interface between the two phases is located. While different options exist for this purpose, such as level set methods [8], a volume of fluid solver is used in this project. Volume of fluid methods will be explained in chapter 5.

2.3. Turbulence modelling in two-phase flows

Similarly to single-phase flows, the computational cost of DNS makes these simulations unfeasible for two-phase turbulence. Therefore, it is necessary to employ turbulence models. The idea behind multiphase turbulence modelling is the same as for single-phase flows; however, the presence of multiple species and an interface between them pose additional complexities.

Two widely employed approaches in multiphase flow turbulence modelling are the mixture model and the two-fluid Euler-Euler model [5]. The mixture model treats the multiphase system as a single homogenized mixture with averaged properties. A single continuity and momentum equation is used for the whole system. Furthermore, only one turbulent model is used. On the other hand, the two-fluid Euler-Euler model considers each phase on its own, with separate sets of governing equations for both phases and separate turbulence models. While this model is more adequate for two-phase flows with large density ratios, the density ratios of the cases in this project are not too large (of order 10), and therefore, the mixture model has been selected for its simplicity.

The details and derivation of the mixture model and the two-fluid Euler-Euler model can be found in [5]; for conciseness, a small overview will be presented here.

Two-fluid Euler-Euler model

The k-phase function, φ^k, of a general function φ is obtained by

$$\varphi^k = \alpha^k \varphi,$$

where α^k takes the value of one if the point is occupied by the k-phase at a given time and zero otherwise. The phase average is defined as

$$\tilde{\varphi}^k = \frac{\bar{\varphi}^k}{\bar{\alpha}^k},$$

and the weighted sum of all the k-phase averages is equal to the total average

$$\bar{\varphi} = \sum \bar{\alpha}^k \tilde{\varphi}^k.$$

The k-phase mass-weighted mean value is given by

$$\hat{\varphi}^k = \frac{\overline{\rho^k \varphi^k}}{\bar{\rho}^k},$$

and the general mean value, $\hat{\varphi}$, can be obtained as the mass-weighted sum of the k-phase mass-weighted mean values

$$\hat{\varphi} = \frac{\sum \bar{\alpha}^k \tilde{\rho}^k \hat{\varphi}^k}{\rho}.$$

The mass conservation equation per phase is

$$\frac{\partial}{\partial t}\left(\bar{\alpha}^k \tilde{\rho}^k\right) + \frac{\partial}{\partial x_j}\left(\bar{\alpha}^k \tilde{\rho}^k \hat{u}_j^k\right) = \Gamma^k,$$

where Γ^k is the interfacial source of mass and must satisfy that $\sum_k \Gamma^k = 0$. The term Γ^k represents the amount of mass that it is being transferred through the interface, for example, due to the evaporation of one of the phases. Since we are interested in flows with no phase mass transfer, this term would be zero for this work. The momentum conservation equation is

$$\frac{\partial}{\partial t}\left(\bar{\alpha}^k\tilde{\rho}^k\hat{u}_i^k\right) + \frac{\partial}{\partial x_j}\left(\bar{\alpha}^k\tilde{\rho}^k\hat{u}_i^k\hat{u}_j^k\right) = -\frac{\partial}{\partial x_i}\left(\bar{\alpha}^k\tilde{p}^k\right) + \frac{\partial}{\partial x_j}\left[\bar{\alpha}^k\left(\tilde{\tau}_{ij}^k + R_{ij}^k\right)\right] + \bar{\alpha}^k\tilde{\rho}^k\hat{f}_i^k + M_i^k,$$

where R_{ij}^k is the k-phase Reynolds stress tensor and M_i^k is the k-phase interfacial momentum source and it is usually given.

As was the case in single-phase flows, the Reynolds stress tensor is an unclosed term which needs additional modelling. The development of turbulence models for multiphase flows is an active area of research, and while some models for particular cases are available, single-phase models are often used. However, single-phase models are inadequate due to the presence of the interface between phases, causing an overestimation of the turbulent kinetic energy [9]. Some authors have mitigated this issue with source terms in the turbulent quantities equations [10], [11], [9].

Mixture model

As mentioned before, the mixture model treated both phases as a mixture and uses the same equation for the whole mixture. These equations are formulated in terms of the total averages instead of the phase averages. The continuity equation of this model for an incompressible fluid is simply given by

$$\frac{\partial\hat{u}_j}{\partial x_j} = 0.$$

The momentum equation has the form

$$\frac{\partial}{\partial t}\left(\tilde{\rho}\hat{u}_i\right) + \frac{\partial}{\partial x_j}\left(\tilde{\rho}\hat{u}_i\hat{u}_j\right) = -\frac{\partial}{\partial x_i}\left(\tilde{p}\right) + \frac{\partial}{\partial x_j}\left[\left(\tilde{\tau}_{ij} + R_{ij}\right)\right] + \tilde{\rho}\hat{f}_i,$$

where R_{ij} is the Reynolds stress tensor.

3 Machine learning for turbulence modeling

This chapter provides an overview of the machine learning algorithms that will be used in this project, as well as the current state of the art in machine learning for turbulence modeling.

3.1. Introduction to machine learning

Machine learning is a field within artificial intelligence that focuses on the development of algorithms and statistical models, allowing computer systems to improve their performance on a specific task by learning from and analyzing data. Unlike traditional programming, where explicit instructions are provided, machine learning enables systems to learn and make predictions or decisions without being explicitly programmed for each scenario.

In [12], three general categories for machine learning techniques are given: supervised learning, unsupervised learning and semi-supervised learning.

- **Supervised learning**: In supervised learning, the algorithm is trained on a labelled dataset, where each input is associated with a corresponding output or target. The model learns to map the inputs to the correct outputs by generalizing from the labelled examples provided during training. The goal is to make accurate predictions on new, unseen data.
- **Unsupervised learning**: Unsupervised learning involves training models on unlabeled data where the algorithm tries to find hidden patterns or structures within the data. The machine learning algorithm aims to explore the data and identify relationships or groupings without explicit guidance on the output.
- **Semisupervised learning**: In semi-supervised learning, the algorithm undergoes training with partial supervision, utilizing either a restricted set of labelled training data or incorporating corrective information from the environment. This class of machine learning includes reinforcement learning, which is a type of machine learning where an intelligent agent learns to make decisions by receiving rewards or penalties for actions taken.

This project will concentrate on the use of supervised learning algorithms, with a specific emphasis on regression algorithms, which aim to predict the value of some quantity based on input data by modelling the relationship between the input variables and the output variable. While numerous supervised learning algorithms are available in the next section, a short description of the two chosen for this project is given: artificial neural networks and symbolic regression.

3.1.1. Neural networks

Neural networks (NN) are machine learning computational models inspired by the structure and functioning of the human brain. In the context of this work, they will be employed for supervised machine learning, though they can also be utilized in other applications. NNs consist of interconnected nodes, often referred to as neurons. The output of a single neuron is computed by

$$p(x) = \alpha(w_i x_i + b_i),$$

where $x_i \in \mathbb{R}^{n_{\text{inputs}}}$ are the inputs, $w_i \in \mathbb{R}^{n_{\text{inputs}}}$ are the weights of those inputs, $b \in \mathbb{R}$ is referred as the bias and $\alpha(\cdot)$ is a non-linear function called the activation function. Different examples of activation functions include the rectified linear unit (ReLU)

$$\alpha(y) = \max(0, y),$$

or the sigmoid function

$$\alpha(y) = \frac{1}{1 + \exp^{-y}}.$$

In classical feed-forward neural networks, neurons are organized in layers, which then are connected. Other types of neural networks include, for example, recurrent neural networks (RNN), in which the output of a node can be used as input for the same or previous layers. For feed-forward NN, the first layer, or input layer, receives the input data, then the activation of this layer is used as input for the next layer, and so on, until the output layer is reached, which output is the target of the regression. The intermediate layers are known as hidden layers. Employing multiple hidden layers is known as deep learning.

The weights and biases of neural networks are unknowns that are learned during training. In a supervised machine-learning framework, these parameters are optimized by minimizing the error between the data labels and the predicted output of the network. This process is usually done by gradient descent algorithms such as Adam or stochastic gradient descent (SDG) [13]. For fully connected NNs, the gradient of the error with respect to the network parameters can be computed using the backpropagation algorithm.

3.1.2. Symbolic regression

Symbolic regression is a type of regression analysis where the goal is to find a mathematical expression or formula that accurately represents the relationship between input variables and the target output. In this work, we will make use of sparse symbolic regression. However, there are other available techniques, such as gene expression programming [14].

Sparse symbolic regression, as used in [15], is a supervised regression technique in which a symbolic expression with only a few nonlinear relevant terms is discovered to fit a given dataset. The expression is obtained by creating a library of candidate functions and then solving a linear optimization problem with sparsity promotion techniques.

Let $\boldsymbol{y} \in \mathbb{R}^{n_y}$ be the target of the regression and $\boldsymbol{x} \in \mathbb{R}^{n_x}$ be the input features. Let $\boldsymbol{\Theta}(\boldsymbol{x}) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_\Theta}$ be the candidate functions library

$$\boldsymbol{\Theta}(\boldsymbol{x}) = (\Theta_1(\boldsymbol{x}), \Theta_2(\boldsymbol{x}), \dots, \Theta_{n_\Theta}(\boldsymbol{x}))$$

where in general each function $\Theta_i(\boldsymbol{x})$ is a nonlinear function of the input features. Note that these functions are given to the algorithm and are not discovered through this process. The aim is to find the linear coefficients, $\Xi \in \mathbb{R}^{n_\Theta \times n_y}$ $\Xi = \left(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \dots, \boldsymbol{\xi}_{n_y}\right)$, for each of the candidate functions such that

$$\boldsymbol{y} = \boldsymbol{\Theta}(\boldsymbol{x})\Xi,$$

and Ξ is sparse. The coefficient vectors are found solving an optimization problem in which the error is minimized

$$\Xi = \arg\min_{\tilde{\Xi}} \|\Theta(X)\tilde{\Xi} - Y\|, \tag{3.1}$$

where $Y \in \mathbb{R}^{n \times n_y}$ and $X \in \mathbb{R}^{n \times n_x}$ are the outputs and inputs of all the observations, n is the number of different observations and $\Theta(X) : \mathbb{R}^{n \times n_x} \to \mathbb{R}^{n \times n_\Theta}$ is the candidate library for all observations. In general, the least square error is used and additional regularization terms are included in equation (3.1) to promote sparsity of vectors $\boldsymbol{\xi_i}$, more details can be found in chapter 4. For clarity

$$\Xi = \arg\min_{\tilde{\Xi}} \left\| \begin{pmatrix} \boldsymbol{\Theta}(\boldsymbol{x^1}) \\ \boldsymbol{\Theta}(\boldsymbol{x^2}) \\ \vdots \\ \boldsymbol{\Theta}(\boldsymbol{x^n}) \end{pmatrix} \left(\tilde{\boldsymbol{\xi}}_1, \tilde{\boldsymbol{\xi}}_2, \dots, \tilde{\boldsymbol{\xi}}_{n_y}\right) - \begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_n \end{pmatrix} \right\|_2^2 .$$

3.2. Machine learning for fluid dynamics

In recent years, the use of machine learning techniques in scientific domains, including fluid dynamics, has gained considerable attention. This progress is aided by the large increase in the computational resources and data that we have available. While these techniques have applications in many areas of fluid mechanics, such as optimization, reduced-order modelling, and control, in this project, we are interested in data-driven turbulence modelling.

Duraisamy et al. [16] identify four levels of modelling in which uncertainties are introduced in turbulence modelling:

- **L1**: Uncertainties are introduced when the ensemble average procedure is applied to the Navier Stokes equations. There exists an infinite number of flow fields that are compatible with the average flow field, and it is impossible to recover the microscopic flow field from the average one. Therefore, information is lost, which is unrecoverable.
- **L2**: In order to close the averaged equations, the unclosed terms are modelled as a function of the mean flow variables. Assumptions in the dependencies of this functional form are level 2 uncertainties.
- **L3**: With the variable dependencies already set, the third level of uncertainties involves the particular form the functional takes.
- **L4**: Lastly, once a functional form is set, the model parameters need to be chosen or calibrated. Level 4 uncertainties refer to this process.

The inherent assumptions in the RANS approach and the process of formulating closure models introduce potential accuracy limitations and reduced predictive ability. For example, the Boussinesq assumption is an example of (L2) uncertainty, the form of the turbulent transport equations in the $\kappa - \epsilon$ model is a (L3) uncertainty, and the choice of the parameters for the same model is a (L4) uncertainty.

3.2.1. Previous work

While linear eddy viscosity models are widely used, it is well known that the Boussinesq hypothesis is not valid for many flows [2], such as those involving strong streamline curvature, separation, or significant anisotropy. Moreover, it is believed that the inadequacy of this assumption is to blame for the inaccuracies of LEVM in some applications [16]. Hence, in recent years, research has focused on circumventing this hypothesis through the use of data-driven turbulence modelling.

Ling et al. [17] proposed tensor basis neural networks to fully model the Reynolds stress; see figure 3.1. They used the invariants of the deformation and rotation of the velocity as regressors for the coefficients of the tensorial decomposition, this way the model was Galilean invariant. They show the improvement in the prediction of the Reynolds stresses compared to conventional NNs and propagated the predicted Reynolds stress into the RANS model equations to improve the flow field. However, they showed that even if the DNS Reynolds stress data was used in a RANS simulation, the DNS flow field was not exactly recovered, and they don't provide details about how the Reynolds stress is propagated.

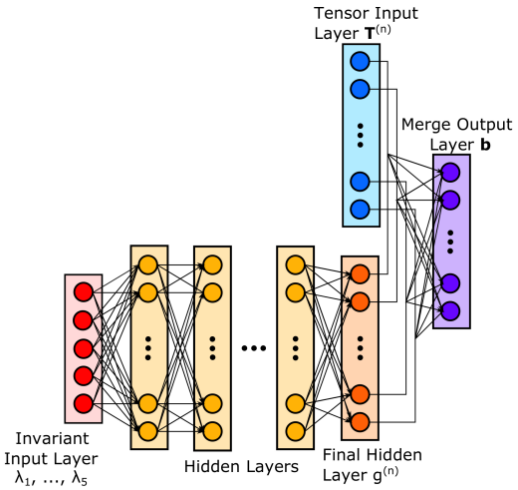


Figure 3.1: Tensor basis neural network diagram. Obtained from [17, p. 6]

In [18], the authors describe the Reynolds stress by its intensity, k, its orientation (three angles) and its coordinates in the barycentric triangle. Then, these six scalars are regressed with random forests using a set of physically motivated chosen invariants as inputs. While the results in predictability and generalizability of the Reynolds stress of this approach are encouraging, they do not propagate this prediction into the RANS equations. In [19], the same set of invariants is used to create a tensor basis random forest to predict the Reynolds stress. However, the author notes the challenges involved with propagating the predictions in a RANS solver due to the lack of robustness.

One avenue to address this robustness issue is to involve the RANS solver in the training process. In [20], the Reynolds stress is modelled with a tensor basis NN, but instead of training the model with Reynolds stress data from DNS, they use the velocity and pressure of the converged flow field to train the model coupled with the RANS solver. To compute the derivative of the RANS equations with respect to the Reynolds stress, they solve an additional set of adjoining equations. They show the capabilities of the model to learn NLEVM from synthetic data and perform better than conventional LEVM when using DNS data. Figure 3.2 shows a summary diagram for this approach. The tensor basis NN is used to predict the Reynolds stress, which is propagated into the RANS equations. Then, The derivative of the cost function with respect to the Reynolds stress is obtained by the adjoint equation and the derivative of the cost function with respect to the NN parameters is obtained using the chain rule.

In [21] [22], the Reynolds stress is again modelled with a tensor basis NN, and in this case, it is trained coupled with the RANS solver through an ensemble filter. The authors observe that this training method results in less computational time of the training as well as better accuracy and robustness compared to the adjoint method presented by [20]. This method is also more flexible because obtaining the analytical expression of the adjoint equations (which change depending on the RANS model and the cost function) is not necessary. Zhang et al. [23] extended the ensemble Kalman training method by including the error of the Reynolds stress model compared to the DNS into the loss function.

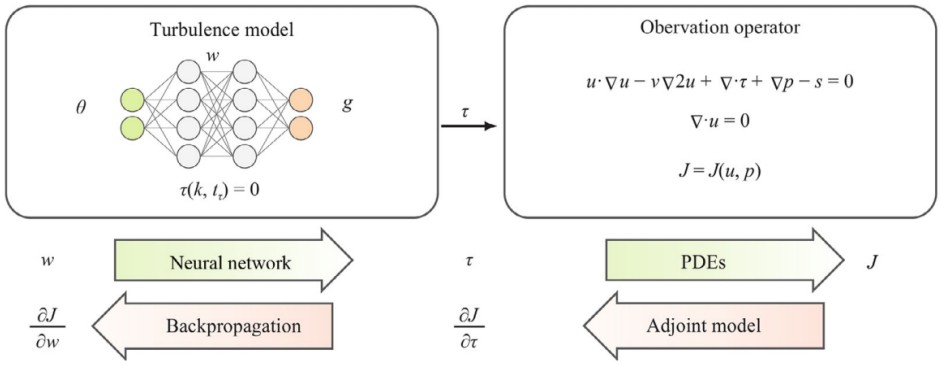


Figure 3.2: Diagram of the adjoin learning method. Obtained from [24, p. 2]

Another way to use model-consistent data for the machine learning process is field inversion. In [25], a multiplicative correction is introduced in the turbulent kinetic energy equation to quantify model uncertainties. The multiplicative field is then obtained by solving a discrete optimization problem. This approach was later extended to compute corrections for RANS models, which can be regressed using machine learning.

In [26], the authors present SpaRTA (Sparse Regression of Turbulent Stress Anisotropy). Sparse symbolic regression is used to model the coefficients of corrections to the Reynolds stress and the turbulent kinetic energy equation. The value of these corrections is obtained through a field inversion of the DNS data, which they named k-corrective frozen-RANS. Their field inversion consists of solving the additional turbulent transport equations of the RANS model for ω and the correction of the k equation using the DNS data of the flow field, Reynolds stress and k. This way, model inconsistencies for the turbulent quantities are avoided. They show that propagating these corrections in the RANS model results in the accurate recovery of the DNS flow field.

Mandler et al. [27] investigated the effect of using corrections in the k-equation to obtain model-consistent turbulent scales. They concluded that while it did not have a noticeable impact in a priory prediction of Reynolds stresses, models with k-corrections outperformed those that did not have it when the Reynolds stress was propagated in the RANS solvers. They also developed a series of realizability restrictions and limiters to the machine learning models, which resulted in improved robustness [28].

Another (L2) assumption that is usually employed is that closure models only depend on local data. This is derived from the local turbulence equilibrium hypothesis. However, as was the case with the Boussinesq hypothesis, this assumption is not valid for many flows [2]. Although to a lesser extent than with the Boussinesq hypothesis, recent research has also tried to address this issue through non-local data-driven turbulence model.

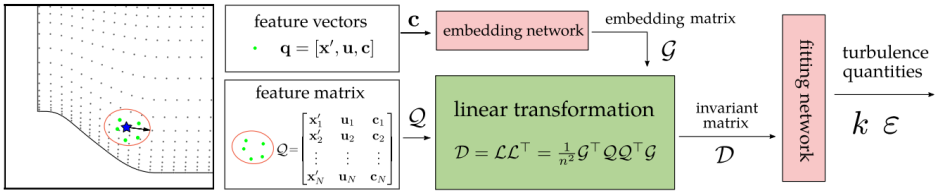


Figure 3.3: Vector cloud neural network diagram. Obtained from [29, p. 3]

In [30], the authors introduce vector cloud neural networks (VCNN), a new NN architecture which can use non-local data, such as the flow variables of a different point in the domain, while preserving the frame invariance. This model consists of two neural networks and an embedding transformation. The data is sampled in different points, and for each point, the data goes through the first network to extract a given number of features. These features are then used to construct the embedding matrix, a linear transformation that is applied to the original sampled data. With this procedure, frame invariance and order of sampling invariance is achieved. Lastly, the fitting network is used to regress the target using the transformed data as input. They then show the potential of this new approach by solving a convection equation using this machine learning model. A scheme of this network architecture can

be seen in figure 3.3. Later, Vector Clouds NN were used in [29] to completely remove the additional transport equations used in RANS models by predicting the turbulent quantities directly using VCNNs.

4 Methodology

This chapter explains the framework for data-driven turbulence modelling. It begins by describing how the RANS equations can be simplified for one-dimensional cases, setting the stage for subsequent developments. We then present two methods for inverting the turbulent viscosity field. Finally, we outline the framework for discovering machine learning models, discuss how corrections are introduced into the model to improve its accuracy and explain the regression techniques.

4.1. Simplified model

This section explains how the 3D RANS equation of the $k-\omega$ model can be simplified for statistically 1D cases. The $k-\omega$ model equations are

$$\begin{aligned}
\frac{\partial U_i}{\partial x_i} &= 0, \\
\frac{\partial \left(\rho U_i\right)}{\partial t} + \frac{\partial \left(\rho U_i U_j\right)}{\partial x_j} &= -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left((\nu + \nu_t)\frac{\partial U_i}{\partial x_j}\right) + \rho g_i, \\
\frac{D\left(\rho k\right)}{Dt} &= \frac{\partial}{\partial x_i}\left[\rho\left(\nu + \alpha_k \nu_t\right)\frac{\partial k}{\partial x_i}\right] + \rho\nu_t \frac{\partial U_i}{\partial x_j}\frac{\partial U_i}{\partial x_j} - \beta^* \rho\omega k, \qquad (4.1) \\
\frac{D\left(\rho\omega\right)}{Dt} &= \frac{\partial}{\partial x_i}\left[\rho\left(\nu + \alpha_\omega \nu_t\right)\frac{\partial \omega}{\partial x_i}\right] + \rho\gamma \frac{\partial U_i}{\partial x_j}\frac{\partial U_i}{\partial x_j} - \beta\rho\omega^2, \qquad (4.2) \\
\nu_t &= \frac{k}{\omega}.
\end{aligned}$$

However, we are interested in the stationary state of a homogeneous solution in the x and z directions. This means that the time derivatives and derivatives by x and z will be zero. Moreover, only the streamwise velocity U_1 is non-zero. Therefore, the model can be simplified to one-dimensional, stationary partial differential equations.

$$\begin{aligned}
\frac{d}{dy}\left[\rho\left(\nu + \nu_t\right)\frac{dU}{dy}\right] &= \frac{dp}{dx}, \qquad (4.3) \\
\frac{d}{dy}\left[\rho\left(\nu + \alpha_k \nu_t\right)\frac{dk}{dy}\right] &= \beta^* \rho\omega k - \rho\nu_t \left(\frac{dU}{dy}\right)^2, \\
\frac{d}{dy}\left[\rho\left(\nu + \alpha_\omega \nu_t\right)\frac{d\omega}{dy}\right] &= \beta\rho\omega^2 - \rho\gamma \left(\frac{dU}{dy}\right)^2, \\
\nu_t &= \frac{k}{\omega}.
\end{aligned}$$

In these reduced equations, the pressure is no longer unknown, and only the known pressure drop dp/dx appears in the momentum equation. Moreover, the continuity equation is automatically satisfied, so it is no longer considered.

4.2. Turbulent viscosity field inversion

From equation (4.3), it is clear that for a given case, the turbulent viscosity ν_t fully determines the velocity profile in the streamwise direction. We would like to know the turbulent viscosity field that

the turbulence model should predict such that the velocity profile is accurate. This problem is known as the field inversion of the turbulent viscosity. For this purpose, two approaches are proposed:

- **Using the DNS Reynolds stress tensor.** Comparing the momentum equation of our model 4.3 with the averaged Navier-Stokes equation (2.7), it can be seen that for both equations to be equal, the following condition must be met.

$$\begin{aligned} \frac{\partial}{\partial x_j}\left(-\rho R_{1j}\right) &= \frac{\partial}{\partial x_j}\left[\rho\left(-\frac{2k}{3}\delta_{1j} + \nu_t \frac{\partial U_1}{\partial x_j}\right)\right] \\ -\frac{\partial(\rho R_{11})}{\partial x} - \frac{\partial(\rho R_{12})}{\partial y} - \frac{\partial(\rho R_{13})}{\partial z} &= \frac{\partial}{\partial y}\left(\rho\nu_t \frac{\partial U_1}{\partial y}\right) \\ -\frac{\partial(\rho R_{12})}{\partial y} &= \frac{\partial}{\partial y}\left(\rho\nu_t \frac{\partial U_1}{\partial y}\right), \end{aligned}$$

  This can be achieved by simply setting the turbulent viscosity to

$$\nu_t = -\frac{R_{12}}{\frac{\partial U_1}{\partial y}}. \tag{4.4}$$

  This is the same definition of ν_t given in [31]. However, this definition of the turbulent viscosity suffers from some issues. The derivative of the streamwise velocity appears in the denominator of the equation (4.4), which means that when there is a critical point in the velocity ν_t will tend to infinity. Moreover, if R_{12} and $\frac{\partial U_1}{\partial y}$ have the same sign, the turbulent viscosity is negative. These two issues lead to unphysical values of ν_t. To avoid this, instead of using the definition (4.4) directly, an optimization process is set to obtain a turbulent viscosity without unphysical values. The loss function for this problem is

$$\mathcal{L}(\boldsymbol{\nu}_t) = \left\| \frac{\partial U_1^{DNS}}{\partial y}\left(\boldsymbol{\nu}_t \frac{\partial U_1^{\text{DNS}}}{\partial y} + R_{12}^{\text{DNS}}\right)\right\|_2^2 + \lambda_1 \sum_{i=0}^{n-1} \left\| \frac{\nu_{t,i+1} - \nu_{t,i}}{h} \right\|_2^2.$$

  The first term minimizes the difference between ν_t and the definition given by (4.4), while the second one is a regularization term that minimizes the discretized first derivative of ν_t. The first term is multiplied by the velocity gradient so that the regularization term gains importance near the critical points of the velocity. This prevents very large values for the turbulent viscosity. Additionally, the values of ν_t are constrained to be positive. The full optimization problem is

$$\arg\min_{\boldsymbol{\nu}_t} \quad \mathcal{L}(\boldsymbol{\nu}_t), \qquad \text{s.t. } \nu_{t,i} \geq 0, \qquad \forall i.$$

- **Using the DNS velocity profile.** Another approach is to solve an optimization problem to minimize the residual of the discretization of the momentum equation (4.3). The discretized momentum equation has the form

$$A(\boldsymbol{\nu}_t)\boldsymbol{u} = \boldsymbol{b}.$$

  Details about the discretization of this equation can be found in chapter 5. The loss function of the optimization problem is

$$\mathcal{L}(\boldsymbol{\nu}_t) = \left\|\boldsymbol{u}_{\text{DNS}} - A(\boldsymbol{\nu}_t)^{-1}\boldsymbol{b}\right\|_2^2 + \lambda_1 \sum_{i=0}^{n-1} \left\| \frac{\nu_{t,i+1} - \nu_{t,i}}{h} \right\|_2^2,$$

  where $\boldsymbol{u}_{\text{DNS}}$ is the velocity profile from the DNS simulations, and a regularization term for the first derivative is added again.

  The gradient of the first term of the loss function with respect to each component of $\boldsymbol{\nu}_t$ is

$$\frac{\partial \left\| \boldsymbol{u}_{\text{DNS}} - A(\boldsymbol{\nu}_t)^{-1}\boldsymbol{b} \right\|_2^2}{\partial \nu_{t,i}} = 2\left(\boldsymbol{u}_{\text{DNS}} - A^{-1}\boldsymbol{b}\right)^T \left(A^{-1}\frac{\partial A}{\partial \nu_{t,i}}A^{-1}\boldsymbol{b}\right).$$

The final optimization problem is

$$\arg\min_{\boldsymbol{\nu}_t} \quad \mathcal{L}(\boldsymbol{\nu}_t), \qquad \text{s.t. } \nu_{t,i} \geq 0, \qquad \forall i.$$

Both optimization problems are solved using the coordinate descent algorithm 1. This algorithm randomly cycles through all the degrees of freedom of the turbulent viscosity, in this case, the value in each mesh node, and performs a gradient descent step using Armijo's rule for backtracking.

Algorithm 1 `coordinate_descent`

for $i \in \{1, \ldots,$ `max_iter`$\}$ **do**
  Initialize `indices` $\leftarrow \{1, \ldots,$ `mesh`$.n - 1\}$
  Suffle `indices`
  while lenght(`indices`) > 0 **do**
    $j \leftarrow$ pop(`indices`)
    `grad` $\leftarrow \frac{\partial f}{\partial \nu_{t,i}}$
    $d \leftarrow -\frac{\texttt{grad}}{\|\texttt{grad}\|}$
    `step` $\leftarrow s$
    for $j \in \{1, \ldots,$ `max_iter_ls`$\}$ **do**
      $\nu^*_{t,j} \leftarrow \max(\nu_{t,j} + \texttt{step} * d, \nu_{\min})$
      if $f(\boldsymbol{\nu_t^*}) < f(\boldsymbol{\nu_t}) + \alpha * \texttt{step} * d *$ `grad` **then**
        $\nu_t \leftarrow \nu_t^*$
        break
      else
        `step` $\leftarrow \beta *$ `step`
  if Stopping criterium **then**
    break

4.3. Machine learning framework

This section details the machine learning framework used in our project. In this framework, we use the velocity and Reynolds stress tensor obtained from the DNS and the turbulent viscosity field from the field inversion to find machine learning models that improve the accuracy of the $k - \omega$ turbulent model. Figure 4.1 shows an overview of this process. The high-fidelity data obtained from the DNS simulations in Briscola is used for the turbulent viscosity field inversion, then the same data and the obtained viscosity are introduced in the model equations to obtain the targets of the machine learning as well as the input features that will be used for the regressions. Finally, the regression techniques are used to discover models of which the best performing are selected and propagated in OpenFOAM. The numerical solvers involved in this framework are introduced in chapter 5.

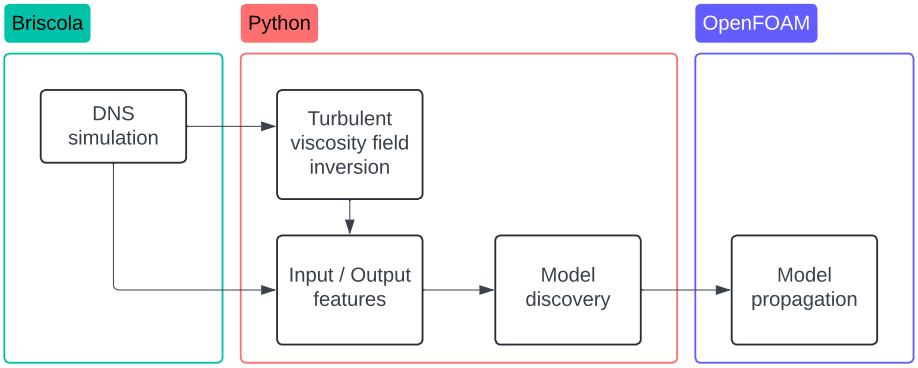


Figure 4.1: Model discovery framework.

The first step is understanding the model's shortcomings and what needs improvement. After some investigations, the most promising approach is introducing two different corrections in the k and ω turbulent transport equations. The corrected model equations are

$$\frac{d}{dy}\left[\rho\left(\nu+\nu_t\right)\frac{dU}{dy}\right]=\frac{dp}{dx},$$

$$\frac{d}{dy}\left[\rho\left(\nu+\alpha_k\nu_t\right)\frac{dk}{dy}\right]=\beta^*\rho\omega k-\rho\left(\nu_t\left(\frac{dU}{dy}\right)^2\right)+\rho\Delta_k, \tag{4.5}$$

$$\frac{d}{dy}\left[\rho\left(\nu+\alpha_\omega\nu_t\right)\frac{d\omega}{dy}\right]=\beta\rho\omega^2-\rho\left(\gamma\left(\frac{dU}{dy}\right)^2\right)+\rho\Delta_\omega, \tag{4.6}$$

$$\nu_t=\frac{k}{\omega}.$$

Where the terms Δ_k and Δ_k are the machine learning modeled corrections. Different forms for these terms are explored, shown in equations (4.7) and (4.8).

$$\Delta_k=f(\boldsymbol{q})\bar{\Delta}_k g_k(\boldsymbol{q}),\quad \Delta_\omega=f(\boldsymbol{q})\bar{\Delta}_\omega g_\omega(\boldsymbol{q}). \tag{4.7}$$

$$\Delta_k=f(\boldsymbol{q})\bar{\Delta}_k e^{g_k(\boldsymbol{q})},\quad \Delta_\omega=f(\boldsymbol{q})\bar{\Delta}_\omega e^{g_\omega(\boldsymbol{q})}. \tag{4.8}$$

Here $\boldsymbol{q}$ are the input features, g_k and g_ω are the machine learning functions (whose output is nondimensional), $\bar{\Delta}_k$ and $\bar{\Delta}_\omega$ are dimensional factors and $f(\boldsymbol{q})$ is a filter function used to only propagate the corrections in certain areas of the domain.

4.3.1. Targets

Once the turbulent viscosity field ν_t^{opt} has been obtained by a field inversion, the next step is to modify the k and ω equations such that they predict ν_t^{opt}.

We would like the model to predict U^{DNS}, k^{DNS}, ν_t^{opt}, $\omega^{\text{opt}}:=\frac{k^{\text{DNS}}}{\nu_t^{\text{opt}}}$.

The machine learning targets Δk and $\Delta\omega$ are then obtained by introducing U^{DNS}, k^{DNS}, ν_t^{opt} and ω^{opt} in equations (4.5) and (4.6)

$$\Delta_k=\frac{\frac{d}{dy}\left[\rho\left(\nu+\alpha_k\nu_t^{\text{opt}}\right)\frac{dk^{\text{DNS}}}{dy}\right]-\beta^*\rho\omega^{\text{opt}}k^{\text{DNS}}+\rho\left[\nu_t^{\text{opt}}\left(\frac{dU^{\text{DNS}}}{dy}\right)^2\right]}{\rho}. \tag{4.9}$$

$$\Delta_\omega=\frac{\frac{d}{dy}\left[\rho\left(\nu+\alpha_\omega\nu_t^{\text{opt}}\right)\frac{d\omega^{\text{opt}}}{dy}\right]-\beta\rho(\omega^{\text{opt}})^2+\rho\left[\gamma\left(\frac{dU^{\text{DNS}}}{dy}\right)^2\right]}{\rho}. \tag{4.10}$$

4.3.2. Input features

The input features for the model regression are chosen based on previous work in the literature. These input features must be non-dimensional and Galilean invariant. Three different feature sets are used:

- The first feature set is taken from the work of Ling et al. [17]. These features are obtained from the invariants of $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$, which were originally proposed in the tensor basis decomposition of the Reynolds Stress Tensor by [2]. Only the features that are non-zero for 1D flows are included.
- The second set is an extension of the previous set using also the normalized gradient of the turbulent kinetic energy [18].
- The last set is composed of features obtained by physical intuition and which are aimed to have an easy physical interpretation [32]. An additional feature is included in this set to include for the effect of the interface.

The features are then normalized using the function

$$q_\beta = \frac{\tilde{q}_\beta}{|\tilde{q}_\beta| + |q^*_\beta|},$$

ensuring that all features lie between -1 and 1, $q_\beta \in (1, -1)$. Normalization is also necessary for the features of the third set to obtain non-dimensional inputs. Table 4.1 summarises the input features and their normalization factor q^*_β.

Table 4.1 shows a summary

Feature Set	Index	Feature	Normalization Factor
FS1	q_1	$\frac{1}{2\omega^2}\left(\frac{dU}{dy}\right)^2$	1
	q_2	$\frac{-1}{8\omega^4}\left(\frac{dU}{dy}\right)^4$	1
FS2	q_3	$\frac{-2k}{\epsilon^2}\left(\frac{dk}{dy}\right)^2$	1
	q_4	$\frac{-k}{4\epsilon^2\omega^2}\left(\frac{dU}{dy}\right)^2\left(\frac{dk}{dy}\right)^2$	1
FS3	q_5	$\min\left(\frac{\sqrt{k}d_{\text{wall}}}{50\nu}, 2\right)$	-
	q_6	$\min\left(\frac{\sqrt{k}d_{\text{int}}}{50\nu}, 2\right)$	-
	q_7	k	$\frac{1}{2}U^2$
	q_8	$U\frac{dp}{dx}$	$\sqrt{U^2\left(\frac{dp}{dx_i}\frac{dp}{dx_i}\right)}$
	q_9	$\frac{k}{\epsilon}$	$\frac{1}{\sqrt{2}}\left\lvert\frac{dU}{dy}\right\rvert$

Table 4.1: Features used for the machine learning algorithms

4.3.3. Sparse symbolic regression

The initial technique employed to model the corrections is sparse symbolic regression. Originally introduced by Brunton et al. [15] for uncovering the governing laws of dynamical systems, a similar framework is utilized here to derive symbolic expressions for the model's corrections using supervised machine learning. The goal is to ensure that these expressions are not only accurate but also simple and comprehensible, thereby minimizing the risk of overfitting and enhancing the model's interpretability.

We consider that the corrections can be expressed as a linear combination of functions of the input features

$$\Delta_k = \Theta_k \xi_k, \quad \Delta_\omega = \Theta_\omega \xi_\omega,$$

Where $\Theta_k \in \mathbb{R}^{1\times n_{f_k}}$ and $\Theta_\omega \in \mathbb{R}^{1\times n_{f_\omega}}$ are the function libraries, n_{f_k} and n_{f_ω} are the number of functions in the library and $\xi_k \in \mathbb{R}^{n_{f\,k}}$ and $\xi_\omega \in \mathbb{R}^{n_{f\,\omega}}$ are the coefficients associated to each of the functions in the function library. The symbolic regression aims to find the coefficient vectors so that they are sparse.

Another model we consider is

$$\log(\Delta_k) = \Theta_k \xi_k, \quad \log(\Delta_\omega) = \Theta_\omega \xi_\omega,$$

in which instead of predicting the correction, the logarithm of the correction is predicted instead.

Two crucial components are required to find a model: a library of candidate functions and the regression method to obtain the coefficient vectors. The candidate functions library serves as a repository of potential mathematical expressions from which the model can select to represent the underlying relationships in the data. Meanwhile, the regression method guides the optimization process by minimizing the error between the predicted and actual values.

In general, the optimization problem consists of a least squares problem where additional terms may be added to the loss function to promote parsimony and simplicity. For example, if there are available n_p point samples of data for the corrections, the loss function would be

$$\|\Delta_i - \Theta_{ij}\xi_j\|_2^2,$$

plus the regularization terms, where $\Delta_i \in \mathbb{R}^{n_p}$, $\Theta_{ij} \in \mathbb{R}^{n_p \times n_f}$ and $\xi_j \in \mathbb{R}^{n_f}$. Four different regression methods will be used: LASSO [33], elastic net [34], STLSQ [15] and SR3 [35].

Candidate Function Library

The selection of the candidate function library is a critical step in using sparse symbolic regression. This library is composed of a set of potential mathematical expressions that the model can use to represent the corrections accurately. However, the functions themselves need to be set manually, and therefore, expertise about potential relationships between the target and the inputs is needed.

For this work, the candidate function libraries are composed of polynomial combinations of the input features up to order N.

$$\mathcal{B} = \left[1, q_1, q_1^2, q_1 q_2, \ldots\right].$$

An additional dimensional factor, $\bar{\Delta}_k$ is multiplied with the library so the regressed expression dimensions match those of the equation that is going to be added to

$$\Theta_k = \bar{\Delta}_k \mathcal{B}, \qquad \Theta_\omega = \bar{\Delta}_\omega \mathcal{B},$$

or in the case, the logarithm of the correction is being regressed

$$\Theta_k = \log\left(\bar{\Delta}_k\right) + \mathcal{B}, \qquad \Theta_\omega = \log\left(\bar{\Delta}_\omega\right) + \mathcal{B},$$

where

$$\bar{\Delta}_k \in \left\{k\omega, \left(\frac{dU}{dy}\right)^2\right\}, \quad \text{and} \quad \bar{\Delta}_\omega \in \left\{\omega^2, \left(\frac{dU}{dy}\right)^2\right\}.$$

LASSO

LASSO, which stands for least absolute shrinkage and selection operator, is a regression method in which the objective is to solve the minimization problem

$$\xi = \arg\min_{\tilde{\xi}} \quad \|\Theta\tilde{\xi} - \Delta\|_2^2 + \lambda\|\tilde{\xi}\|_1,$$

where the term $\lambda\|\tilde{\xi}\|_1$ is used for regularization with a weight λ. This l^1-norm penalty term promotes the shrinkage of some coefficients to exactly zero, resulting in sparse expressions in which only some of the functions of the candidate function library are present. Thus improving the interpretability of the model and reducing the risk of overfitting.

The LASSO problem is solved using the Python package Scikit-learn [36], and the parameter space of λ is set to

$$\lambda = \left[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\right].$$

Elastic net

Elastic net is a regression method that combines the l^1-norm regularization term of LASSO with an additional l^2-norm regularization term.

$$\xi = \arg\min_{\tilde{\xi}} \quad \|\Theta\tilde{\xi} - \Delta\|_2^2 + \lambda\rho\|\tilde{\xi}\|_1 + 0.5\lambda(1-\rho)\|\tilde{\xi}\|_2.$$

On one hand, the l^1 has a similar effect as in the LASSO problem. On the other hand, the l^2 term (which, when used alone, is known as Ridge regression) distributes the shrinkage across all coefficients, improving the predictive accuracy when the predictors are highly correlated. The parameter ρ represents the degree of mixture between the two of the penalty terms, with $\rho = 0$ being equal to the Ridge regression and $\rho = 1$ to LASSO.

The optimization problem is solved again using the Python package Scikit-learn. The search space for λ and ρ are

$$\lambda = \left[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\right],$$

$$\rho = [0.01, 0.1, 0.2, 0.5, 0.7, 0.9, 0.95, 0.99, 1.0].$$

Sequential thresholding least squares

Sequential thresholding least squares (STLSQ) is an iterative regression method which uses a different strategy to enforce sparsity compared to the two previous methods [15]. In each iteration, the Ridge regression is solved first

$$\xi = \arg\min_{\tilde{\xi}} \quad \|\Theta\tilde{\xi} - \Delta\|_2^2 + \lambda\|\tilde{\xi}\|_2^2.$$

Then, all the coefficients below a certain threshold are set to zero, excluding the associated terms for the rest of the regression. After that, the process is repeated until convergence.

The STLSQ problem is solved using the Python package PySINDy [37]. The search spaces for λ and the threshold value are

$$\lambda = [0.01, 0.05, 0.1, 0.5],$$

$$\texttt{threshold} = \left[10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\right].$$

Sparse relaxed regularised regression

Sparse relaxed regularised regression (SR3) is yet another regression method. In this case, the loss function is given by

$$\xi = \arg\min_{\tilde{\xi}, u} \quad \|\Theta\tilde{\xi} - \Delta\|_2^2 + \lambda R(u) + \frac{1}{2\nu}\|\tilde{\xi} - u\|_2^2,$$

where u is an auxiliary variable and $R(u)$ its is associated regularization term. Following the work in [38], we will not use a regulariser for u but instead perform thresholding as in STLSQ. The SR3 problem is solved using the Python package PySINDy, and the search space for the hyperparameters is

$$\nu = [0.01, 0.1, 1.0, 10.0],$$

and

$$\texttt{threshold} = \left[10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\right].$$

4.3.4. Neural network

Another powerful and popular supervised machine learning technique is neural networks (NN). These networks have been proven to be highly effective for regression due to their ability to learn complex, non-linear relationships from data. However, these systems constitute black boxes, which hinders the interpretability of the model.

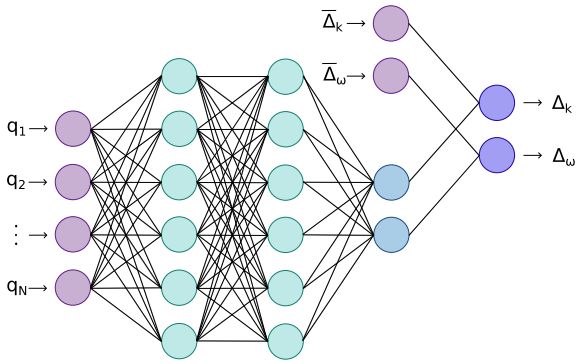


Figure 4.2: Diagram of the neural network architecture.

Many neural network architectures exist, each designed to address specific types of problems and data structures, such as convolutional neural networks (CNNs) for image data and recurrent neural networks (RNNs) for sequential data. However, for this work, we utilize simple, fully connected neural networks with an additional final layer to merge the output with a dimensional factor.

Figure 4.2 shows a diagram of the NN architecture used for this work. The network is composed of an input layer for the input features, two hidden layers, an output layer and an additional input layer for a dimensional factor, which gets combined with the outputs such that the dimensions of the correction match those of the equation introduced into. The number of neurons in each of the hidden layers, as well as the type of activation functions used in those layers, are hyperparameters which are modified to find different models. The two hidden layers can be shared for both corrections, having a single network for both of them or separate, having a different network for each of the corrections.

More rigorously, the mathematical description of the network is:

- **Joint network**

  Let $q \in \mathbb{R}^{n_i}$, $\bar{\Delta} := [\bar{\Delta}_k, \bar{\Delta}_\omega] \in \mathbb{R}^2$ be the inputs of the network and $\Delta := [\Delta_k, \Delta_\omega] \in \mathbb{R}^2$ the output. Then, the network is given by

  $$h_1 = \alpha_1 \left(W_1 q + b_1 \right),$$

  $$h_2 = \alpha_2 \left(W_2 h_1 + b_2 \right),$$

  $$h_3 = W_3 h_3 + b_3,$$

  where $h_i \in \mathbb{R}^{n_{h_i}}$, $W_i \in \mathbb{R}^{n_{h_i} \times n_{h_{i-1}}}$ and $b_i \in \mathbb{R}^{n_{h_i}}$. The dimensions of the layers n_{h_1} and n_{h_2} are hyperparameters, as also are the type of activation functions for these layers $\alpha_1(\cdot)$ and $\alpha_2(\cdot)$ and the dimension of the third layer is $n_{h_3} = 2$. Note that $n_{h_0} = n_i$. The last layer combines the output of the third layer with a dimensional factor depending on whether the correction or its logarithm is being regressed.

  $$\Delta = \bar{\Delta} \odot h_3, \qquad \log\left(\Delta\right) = \log\left(\bar{\Delta}\right) + h_3,$$

  where $\odot$ is the Hadamard product that denotes element-wise multiplication. The weights, W_i, and the biases, b_i, of the network are learnt during training.

- **Separated networks**

  The architecture of the separated networks remains the same as in the previous case, except that the first two hidden layers are no longer shared. This means the two networks only share the initial input layer and can thus be described separately.

  Let $\phi \in \{k, \omega\}$, each network is described as

$$h_{1,\phi} = \alpha_{1,\phi} \left(W_{1,\phi} q + b_{1,\phi} \right),$$

$$h_{2,\phi} = \alpha_{2,\phi} \left(W_{2,\phi} h_{1,\phi} + b_{2,\phi} \right),$$

$$h_{3,\phi} = W_{3,\phi} h_{3,\phi} + b_{3,\phi},$$

  where $h_{i,\phi} \in \mathbb{R}^{n_{h_{i,\phi}}}$, $W_{i,\phi} \in \mathbb{R}^{n_{h_{i,\phi}} \times n_{h_{i-1,\phi}}}$ and $b_{i,\phi} \in \mathbb{R}^{n_{h_{i,\phi}}}$. As in the previous case, the dimension of the third layer is fixed $n_{h_{3,\phi}} = 1$ and the dimension of the hidden layers and their activation functions are hyperparameters. The output of the third layer is again combined with a dimensional factor depending on whether the correction or its logarithm is being regressed.

$$\Delta_\phi = \bar{\Delta}_\phi h_{3,\phi}, \qquad \log\left(\Delta_\phi\right) = \log\left(\bar{\Delta}_\phi\right) + h_{3,\phi}.$$

As mentioned before, the weights and biases (we will refer to both as weights) of the neural networks are unknowns that are learned during training. Initially, the weights are initialized randomly and then, using a gradient descent method and data, the weights are iteratively updated.

More specifically during training, the data is fed through the network in batches, and the output predictions are compared to the actual target values using a loss function, which quantifies the prediction error, for this work the mean squared error is used as the loss function. The backpropagation algorithm is then employed to calculate gradients of the loss function with respect to the network's weights. These gradients are used to update the weights through an optimization algorithm, in the case of this work Adam is used as an optimizer. The hyperparameters of the training are the batch size, the number of epochs and the learning rate of the Adam optimizer, which for this work are set to 32, 100 and 10^{-3} respectively.

For the implementation of the NNs, the TensorFlow Pyhton package is used. TensorFlow also has a C API available, which we use to infer the learnt models in OpenFOAM.

4.3.5. Model discovery

In the previous, different regression techniques were introduced to learn models for the corrections. However, each of these techniques had their own set of hyperparameters, which leads to the question, how do we choose these hyperparameters, or how do we select a model?

Hyperparameter optimization involves systematically searching for the best set of hyperparameters, which are parameters that define the model structure and training process, but are not learned from the data. As a hyperparameter optimization technique, we will use grid search. Grid search is a method where a predefined set of hyperparameters is searched over a specified range. Each combination of hyperparameters is evaluated using cross-fold validation.

Training samples

Earlier, we discussed how a filtering function could be used to propagate the correction only in the desired areas of the domain. Should the correction model then be trained with data samples of those areas only? We use two options to select which data samples need to be used:

- All data samples are used.
- Data samples are only used if for that particular sample

$$f(\boldsymbol{q}) > \texttt{zone\_tol},$$

  where we set $\texttt{zone\_tol} = 10^{-2}$.

Cross-fold validation

Cross-fold validation is a robust technique for assessing a model's generalizability. It involves dividing the dataset into several folds and training the model multiple times, each time using a different fold as the validation set and the remaining folds as the training set. This approach ensures that every data point is used for both training and validation, evaluating the model's performance and reducing the risk of overfitting.

If there are n_t training cases and we perform k-fold cross-validation with k folds, the data is split into k parts, keeping all the data corresponding to the same training case together. For the remaining of this work we will use $k = 5$.

Discarding and selecting models

Once the k-fold cross-validation has been performed for all models, the next step is to select which models to propagate into the RANS simulations. This step is necessary as good data fits in the training stage, but it does not always mean good results when the corrections are introduced in the RANS model. However, propagating all discovered models is computationally expensive; therefore, some selection must be done. These steps vary slightly depending on the regression technique used:

- **Sparse symbolic regression**

  For a given regression technique with n_p total hyperparameter combinations, we have n_p models for Δ_k and n_p models for Δ_ω. Out of those models we discard the ones that have more that `n_max_terms` terms (or equivalently, non zero coefficients). From the remaining models, we select the n_k Δ_k models and n_ω Δ_ω models with the best validation score. Finally, we combine all selected models for k and ω together yielding a total of $n_k \times n_\omega$ models
- **Joint neural network**

  For this case, we simply select the $n_k \times n_\omega$ models with the highest validation score.
- **Separate neural networks**

  The procedure is equal to the symbolic regression case, but no models are discarded in this case.

Model propagation

Lastly, the selected models are propagated into the RANS model, first in the cases used for the training and then in the cases whose data has not been used in the training process. The full framework for the model discovery is summarised in figure 4.3.

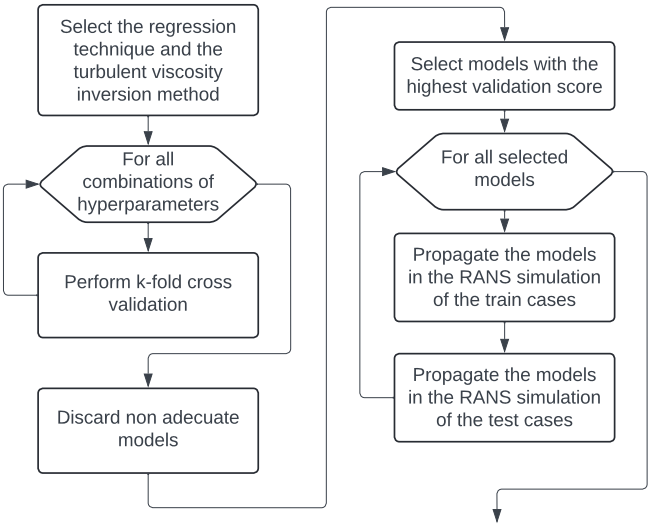


Figure 4.3: Flow chart of the model discovery algorithm.

5 Numerical solvers

This chapter will discuss the numerical solvers used in this project. These solvers are essential for the different stages of the data-driven modelling framework. First, we will look at the solver used for Direct Numerical Simulations (DNS) of two-phase flows, which is used to generate high-fidelity data. Next, we will examine the OpenFOAM solver used for Reynolds-averaged Navier-Stokes simulations. We will present a Python solver for the simplified model, which allows for a simple integration of the solver in the training process of the model discovery. Finally, the test case used throughout the project is introduced. This chapter will provide a clear overview of the numerical methods used in the project.

5.1. Direct numerical simulations of two-phase flows

This section gives an overview of the numerical methods used to perform the two-phase DNS. These simulations were done using Briscola, an NRG in-house finite volume solver, the two-phase part of which was developed throughout this project. The Navier Stokes equations are solved by a pressure correction scheme, and the location of the two fluids is tracked using a VoF solver.

5.1.1. The finite volume method

The finite volume method (FVM) is a numerical technique used to solve partial differential equations. The core idea behind FVM is to divide the domain of the problem into a finite number of small control volumes (or cells) and apply the integral form of the governing equations over these control volumes [39]. Some of the advantageous properties this method has is that it is inherently conservative and that it is well-suited for problems involving discontinuities.

As a very brief example. To solve the following PDE

$$\frac{\partial u_i}{\partial t} + \frac{\partial F_{ij}(\boldsymbol{u})}{\partial x_j} = 0, \tag{5.1}$$

the domain would be discretized in a finite number of cells C_i with volume V_i and equation (5.1) is integrated in each of the cells

$$\int_{C_k} \frac{\partial u_i}{\partial t} dV + \int_{C_k} \frac{\partial F_{ij}(\boldsymbol{u})}{\partial x_j} dV = 0.$$

Using the divergence theorem, the second term can be turned into a surface integral.

$$\int_{C_k} \frac{\partial u_i}{\partial t} dV + \oint_{\partial C_k} \partial F_{ij}(\boldsymbol{u}) n_j dS = 0. \tag{5.2}$$

The average value of a cell is defined as

$$U_{i,k}(t) = \frac{1}{V_k} \int_{C_k} u_i(\boldsymbol{x}, t) dV.$$

Therefore, equation (5.2) can be rewritten as

$$\frac{\partial U_{i,k}}{\partial t} + V_k \oint_{\partial C_k} \partial F_{ij}(\boldsymbol{u}) n_j dS = 0.$$

Finally, to solve this equation, the temporal derivative gets discretized with a time-marching scheme, and the fluxes between cells (the surface integral term) get reconstructed using the average cell values. Different methods will depend on the way these fluxes get reconstructed.

5.1.2. Volume of fluid methods

Volume of Fluid (VOF) methods are numerical techniques used in computational fluid dynamics to simulate and analyze multiphase flows. The fundamental idea behind VOF methods is to track the volume fractions of each fluid phase within each computational cell or grid element. In other words, for two-phase flows, VOF methods provide a way to determine how much of a cell is occupied by one fluid phase and how much is occupied by another. This information allows for the accurate calculation of fluid properties, flow velocities, and pressure gradients at the interfaces between different fluids.

From now on, we will denote the volume fraction α, which ranges from 0 (the cell is filled with one of the phases) to 1 (the cell is filled with the other phase): and the fluid which corresponds to a volume fraction equal to one will be called the liquid. The equation that describes how α evolves is the advection equation

$$\frac{\partial \alpha}{\partial t} + \frac{\partial u_j \alpha}{\partial x_j} = 0.$$

This equation can be solved algebraically or geometrically. Algebraic methods use numerical schemes that discretize the equation into algebraic expressions. Geometric methods, however, focus on explicitly reconstructing the interface geometry to compute the fluid that gets advected. In this work, it will be solved geometrically as these methods result in advantageous properties such as the monotonicity of the solution and less artificial diffusion [40]. For every time step two main steps are performed:

- **Interface reconstruction**: This step consists of constructing an interface that separates the two phases.

  One class of interface reconstruction algorithms are PLIC (piecewise linear interface calculation) schemes. In these schemes, the interface is represented as a plane in each interface cell (those with $0 < \alpha < 1$) of the mesh, which follows the equation

  $$n_i x_i + C = 0, \tag{5.3}$$

  where x_i is the position vector. Thus the problem is reduced to finding n_i and C for each cell.

  This problem is usually solved in two steps. First we obtain n_i for every cell, called the normal reconstruction. Then, we find a value of C for every cell so that the volume of the truncated cell equals the fraction of volume given by α. This problem is called the local volume enforcement (LVE) problem.
- **Convection**: The equation (5.3) is updated via finite volume method. Through geometric techniques, the amount of liquid that enters and leaves every cell is computed and used to update α.

5.1.3. Briscola's numerical scheme

In each time step, the first step is to solve the advection equation of the VoF solver

$$\frac{\alpha^{n+1/2} - \alpha^{n-1/2}}{\Delta t^n} + \frac{\partial}{\partial x_i}\left(\alpha^n u_i^n\right) = 0. \tag{5.4}$$

Equation 5.4 is solved by split advection to impose monotonicity of the solution [40]. The interface normal is reconstructed with the mixed Young central scheme [41], and the LVE problem is solved by an efficient analytical formula [42] valid for parallelepiped cells.

The fraction of the volume field is used to update the fluid properties

$$\varphi^{n+1/2} = \alpha^{n+1/2}\varphi_1^{n+1/2} + (1-\alpha^{n+1/2})\varphi_2^{n+1/2}, \qquad \varphi \in \{\rho, \mu\},$$

Then, a pressure correction scheme is used

$$\frac{u_i^* - u_i^n}{\Delta t^n} = \mathcal{C}_i^{n+1}(u) + \frac{1}{\rho^{n+1/2}}\mathcal{M}_i^{n+1}(\mu, u) - \frac{1}{\rho^{n+1/2}}\frac{\partial p^{n-1/2}}{\partial x_i} + \sigma^{n+1/2}\frac{\kappa^{n+1/2}}{\langle\rho\rangle}\frac{\partial \alpha^{n+1/2}}{\partial x_i},$$

$$u_i^{n+1} = u_i^* - \frac{\Delta t}{\rho^{n+1/2}}\left(\frac{\partial p^{n+1/2}}{\partial x_i} - \frac{\partial p^{n-1/2}}{\partial x_i}\right), \tag{5.5}$$

$$\frac{\partial u_i^{n+1}}{\partial x_i} = 0,$$

where $\mathcal{C}_i^{n+1}(u) \approx u_j(\boldsymbol{x}, t^n)\frac{\partial u_i(\boldsymbol{x},t^n)}{\partial x_j}$ is the time discretization of the convective term, $\mathcal{M}_i^{n+1}(\mu, u) \approx \frac{\partial}{\partial x_j}\left(\mu(\boldsymbol{x}, t^n)\left(\frac{\partial u_i(\boldsymbol{x},t^n)}{\partial x_j} + \frac{\partial u_j(\boldsymbol{x},t^n)}{\partial x_i}\right)\right)$ the one of the viscous term and the last term in the right-hand side is the surface tension. For the convective term, a second-order extrapolation formula is used

$$\mathcal{C}_i^{n+1}(u) = \left(1 + \frac{\Delta t^n}{2\Delta t^{n-1}}\right) u_j^n \frac{\partial u_i^n}{\partial x_j} - \frac{\Delta t^n}{2\Delta t^{n-1}} u_j^{n-1} \frac{\partial u_i^{n-1}}{\partial x_j}.$$

For the viscous term, the first term on the hand side of equation (2.8) is discretized using the Crank-Nicolson scheme and the second one using second-order extrapolation formula again

$$\mathcal{M}_i^{n+1}(\mu, u) = \frac{\partial}{\partial x_j}\left[\frac{\mu^{n+1/2}}{2}\left(\frac{\partial u_i^{n+1}}{\partial x_j} + \frac{\partial u_i^n}{\partial x_j}\right)\right]\left(1 + \frac{\Delta t^n}{2\Delta t^{n-1}}\right)\frac{\partial \mu^{n+1/2}}{\partial x_j}\frac{\partial u_j^n}{\partial x_i} - \frac{\Delta t^n}{2\Delta t^{n-1}}\frac{\partial \mu^{n+1/2}}{\partial x_j}\frac{\partial u_j^{n-1}}{\partial x_i}.$$

Therefore, the prediction step is implicit in the velocity. The resulting system is solved using a multigrid solver. Using the Brackbill formulation [7], the surface tension term has the form

$$\mathcal{F}_i^{n+1/2}(\alpha) = \sigma^{n+1/2}\frac{\kappa^{n+1/2}}{\langle\rho\rangle}\frac{\partial \alpha^{n+1/2}}{\partial x_i},$$

where $\sigma^{n+1/2}$ is the surface tension coefficient, $\langle\rho\rangle$ is the mean density and $\kappa^{n+1/2}$ is the interface curvature. The curvature κ is computed using the standard height function technique with the improvements proposed by Lopez et al. [43]. For stability and accuracy reasons, the pressure gradients and the surface tension must be discretized in the same location [7]. Therefore, the gradients of the α field are computed in the same location as the gradients of the pressure and the rest of the terms necessary for the surface tension are interpolated to these locations if needed.

Solving the projection step (5.5) for the pressure yields a non-constant coefficient Poisson problem, which is well known to be the bottleneck for this type of scheme. Following the work in [7], the pressure term is split into a constant and non-constant coefficient term

$$\frac{1}{\rho^{n+1/2}}\frac{\partial p^{n+1/2}}{\partial x_i} \to \frac{1}{\rho_0}\frac{\partial p^{n+1/2}}{\partial x_i} + \left(\frac{1}{\rho^{n+1/2}} - \frac{1}{\rho_0}\right)\frac{\partial \hat{p}}{\partial x_i},$$

where the constant coefficient term is treated implicitly and the non-constant term explicitly by using a linear extrapolation of the pressure

$$\hat{p} = -\frac{\Delta t^n}{\Delta t^{n-1}} p^{n-1} + \left(1 + \frac{\Delta t^n}{\Delta t^{n-1}}\right) p^n.$$

This allows us to solve the resulting system using a Fast Fourier solver, which can result in speed-ups of one order of magnitude compared to using multigrid [7]. The resulting Poisson equation is

$$\frac{\partial^2 p^{n+1/2}}{(\partial x_i)^2} = \frac{1}{\Delta t^n}\frac{\partial u_i}{\partial x_i} + \frac{\rho_0}{\partial p^{n+1/2}}\frac{\partial^2 p^{n-1/2}}{(\partial x_i)^2} + \left(1 - \frac{\rho_0}{\partial p^{n+1/2}}\right)\frac{\partial^2 \hat{p}}{(\partial x_i)^2},$$

and the correction of the velocity is

$$u_i^{n+1} = u_i^* - \Delta t^n\left[\frac{1}{\rho_0}\left(\frac{\partial p^{n+1/2}}{\partial x_i} - \frac{\partial \hat{p}}{\partial x_i}\right) + \frac{1}{\rho^{n+1/2}}\left(\frac{\partial \hat{p}}{\partial x_i} - \frac{\partial p^{n-1/2}}{\partial x_i}\right)\right].$$

5.1.4. Validation of Briscola

The newly implemented two-phase solvers have been validated with two 2-D rising bubble simulation cases [44]. In these cases, a fluid bubble is immersed in a denser fluid column, which causes the bubble to rise due to the action of gravity. Figure 5.1 shows a description of the simulation setup, and Table 5.2 shows the different parameters for each simulation

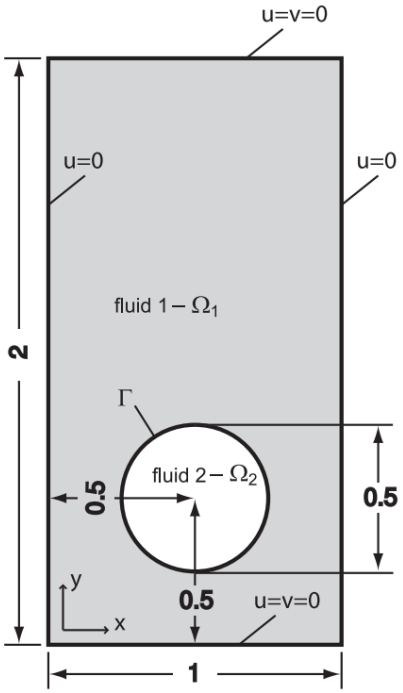


Figure 5.1: Description of the simulation set up, obtained from [44, p. 1262].

Test case	ρ_1	ρ_2	μ_1	μ_2	g	σ	Re	Eo	ρ_1/ρ_2	μ_1/μ_2
1	1000	100	10	1	0.98	24.5	35	10	10	10
2	1000	1	10	0.1	0.98	1.96	35	125	1000	100

Figure 5.2: Simulation parameters for each case.

The characteristic non-dimensional numbers of the problem are the Reynolds number, which describes the ratio of inertial to viscous effects, and the Eotvos number, which gives the ratio of gravitational forces to surface tension effects

$$\mathrm{Re} = \frac{\rho_1 U_g L}{\mu_1}, \qquad \mathrm{Eo} = \frac{\rho_1 U_g L}{\sigma},$$

where $U_g = \sqrt{2gR}$ and R is the radius of the bubble. Four Briscola solver configurations have been tested, using a colocated or a staggered mesh with and without using the split in the pressure equation explained in the previous section. The simulations with the Briscola solvers have been performed on a mesh with cell size $h = 1/128$ and with a fixed timestep $\Delta t = 0.0004$. The reference data is obtained with the solver TP2D [44], which is a FEM-level set solver (Briscola is a finite volume VoF solver) in a mesh with $h = 1/320$ and with a fixed timestep $\Delta t = h/16 \approx 0.0002$. The end time of the simulation is $t = 3$.

The results are compared using the centre of mass height

$$Y_c = \frac{\int_{\Omega_2} y \, dx \, dy}{\int_{\Omega_2} dx \, dy},$$

the average rising velocity

$$V = \frac{\int_{\Omega_2} u_y \, dx \, dy}{\int_{\Omega_2} dx \, dy},$$

and the shape of the bubble at the end time.

Figure 5.3 shows the bubble's shape at the end of the simulation for case 1. It can be seen that the shape is extremely similar for the different solvers; the only noticeable difference is that for the TP2D solver, the bubble is slightly higher.

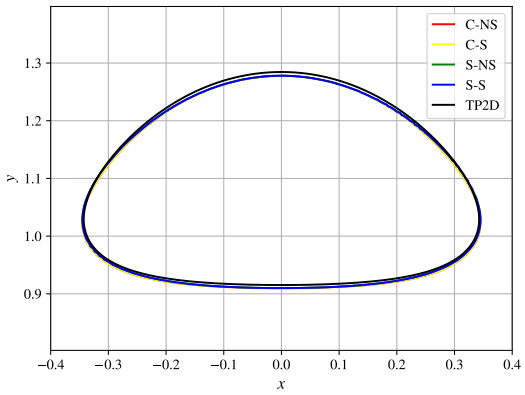


Figure 5.3: Shape of the bubble for case 1 at $t = 3$. Legend: *C-NS: Colocated solver, no split for pressure equation. C-S: Colocated solver, split for pressure equation. S-NS: Staggered solver, no split for pressure equation. S-S: Staggered solver, split for pressure equation. TP2D: Benchmark results from [44]*

In figure 5.4, the vertical position of the centre of mass and the rising velocity of the bubble for case 1 is shown. Again, the results are very similar except for the bubble of TP2D, which rises a little bit faster after $t = 1$. After this time, Briscola solvers marginally under-predict the rising velocity, which leads to a lower final height.

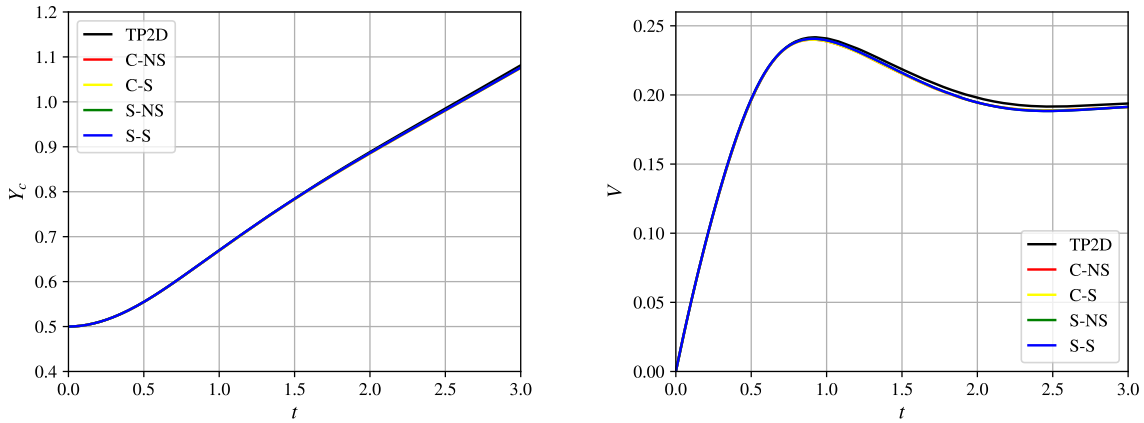


Figure 5.4: Height of the centre of mass (a) and mean rising velocity (b) of the bubble for case 1. Legend: *C-NS: Colocated solver, no split for pressure equation. C-S: Colocated solver, split for pressure equation. S-NS: Staggered solver, no split for pressure equation. S-S: Staggered solver, split for pressure equation. TP2D: Benchmark results from [44]*

Case 2 is a more challenging case for multi-phase solvers. Among the causes for this is the much larger density ratio than in case 1 and the appearance of very thin fluid structures that need fine grids to be captured. This case could not be simulated with the colocated solver using the split for the pressure equation. However, this is not particularly alarming since the split was introduced by [7] for staggered solvers. The end shape of the bubbles is shown in figure 5.5. While the main bulk of the shape is similar, there are differences, mainly in the lower perimeter. The colocated solver predicts a smaller breakage than the reference code, and the staggered solvers don't predict breakage at all. In [44], the performance of different solvers for this case is compared, showing a lot of variation between

each other. Some predict thin filaments, while others break into smaller bubbles. This phenomenon is dominated by surface tension; thus, the main differences can probably be traced to how this force is handled. In the case of Briscola, the curvature of the interface is computed with a height function technique, which is known not to handle very coarse resolutions well. This could explain the difference in the bottom edges.

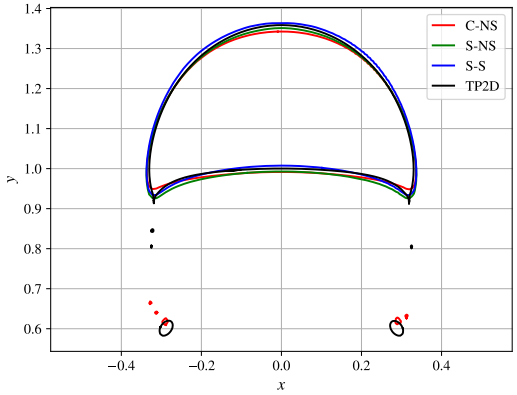
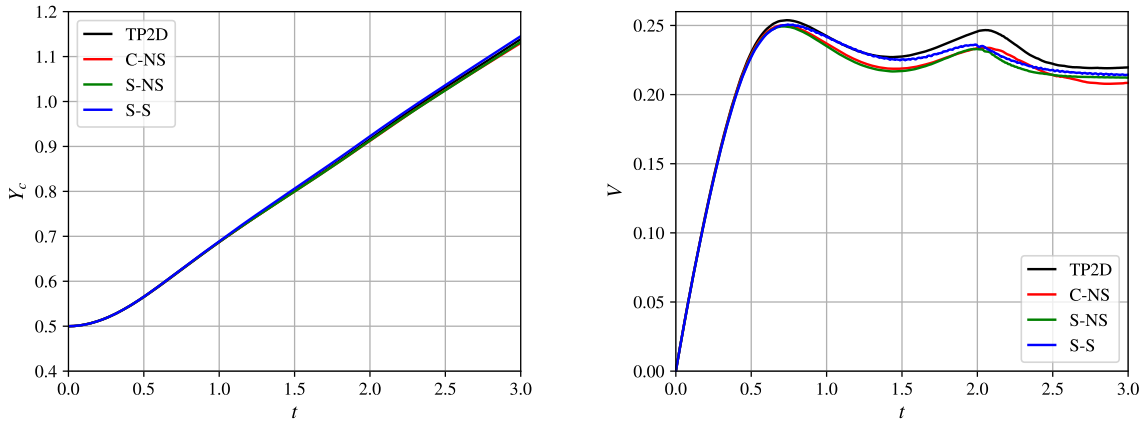


Figure 5.5: Shape of the bubble for case 2 at $t = 3$. Legend: *C-NS: Colocated solver, no split for pressure equation. C-S: Colocated solver, split for pressure equation. S-NS: Staggered solver, no split for pressure equation. S-S: Staggered solver, split for pressure equation. TP2D: Benchmark results from [44]*

In figure 5.6, the vertical position of the centre of mass and the rising velocity of the bubble for case 1 is shown. While the results between solvers are not equal, the variances align with the differences shown between solvers in [44].



(a) **(b)**

Figure 5.6: Height of the centre of mass (a) and mean rising velocity (b) of the bubble for case 2. Legend: *C-NS: Colocated solver, no split for pressure equation. C-S: Colocated solver, split for pressure equation. S-NS: Staggered solver, no split for pressure equation. S-S: Staggered solver, split for pressure equation. TP2D: Benchmark results from [44]*

5.2. RANS solver

The correction models discovered for the $k-\omega$ turbulence model are implemented in the open-source CFD software OpenFOAM. For the simulations, the incompressibleVof module of OpenFOAMv11 is used. The turbulence model kOmega is selected, and the forcing pressure gradient is implemented as an explicit source term in the momentum equation using the momentumSource functionality.

5.2.1. PIMPLE algorithm

OpenFOAM uses the PIMPLE algorithm to solve the incompressible RANS equations [45]. The PIMPLE method combines the SIMPLE (Semi-Implicit-Method for Pressure Linked Equations) and the PISO (Pressure-Implicit with Splitting of Operators) algorithms. This family of numerical algorithms uses splittings such that the momentum and continuity equations can be uncoupled, reducing the total computational cost of solving the system [46].

$$\underbrace{\frac{\partial\left(\rho U_i\right)}{\partial t} + \frac{\partial\left(\rho U_i U_j\right)}{\partial x_j}}_{=:\mathcal{C}_i(U)} = -\underbrace{\frac{\partial p}{\partial x_i}}_{=:\mathcal{G}_i(p)} + \underbrace{\frac{\partial}{\partial x_j}\left[(\nu+\nu_t)\frac{\partial U_i}{\partial x_j}\right]}_{=:\mathcal{L}_i(U)} + \underbrace{\rho g_i}_{=:\mathcal{F}_i}, \tag{5.6}$$

$$\underbrace{\frac{\partial \rho U_i}{\partial x_i}}_{=:\mathcal{D}(\rho U)} = 0, \tag{5.7}$$

The algorithm works as follows. Let $\mathcal{C}_i(U)$, $\mathcal{G}_i(p)$, $\mathcal{L}_i(U)$, $\mathcal{F}_i$ and $\mathcal{D}(\rho U)$ be the convective, gradient, viscous, forcing and divergence operators as shown in equations (5.6) and (5.7). In this case, the momentum equation is discretized implicitly in time using the backward Euler scheme, as shown in (5.8).

$$\frac{(\rho U_i)^{n+1} - (\rho U_i)^n}{\Delta t} + \mathcal{C}_i(U^{n+1}) = -\mathcal{G}_i(p^{n+1}) + \mathcal{L}_i(U^{n+1}) + \mathcal{F}_i^{n+1}. \tag{5.8}$$

Then equation (5.8) is linearized and discretized in space, yielding a system of equations for each velocity component. These equations are solved in an iterative process for each timestep; these iterations are called outer iterations, and we denote by m the index of these iterations. For each time step, the velocity and pressure from the previous time step are used as the starting point for the first outer iteration. The linear system for each velocity component reads as follows.

$$A^{m-1}U_i^m = Q_i^{m-1} - G_i(p^m),$$

where A^{m-1} is a matrix, Q_i^{m-1} includes explicit terms or other terms that may arise from the linearization, and G_i is the ith component of the gradient operator. Note that A and Q depend on the previous solution, but for conciseness, the superindex will be omitted. Matrix A can be split into a diagonal part A_D and an off-diagonal part A_{OD}. For each outer iteration, a predicted velocity is obtained by solving the momentum equation (5.9) using the pressure from the previous iteration.

$$(A_D + A_{OD})U_i^* = Q_i - G_i(p^{m-1}), \tag{5.9}$$

This velocity may not satisfy the continuity equation, so the preliminary velocity and the pressure are corrected.

$$U_i^{**} = U_i^* + U_i', \qquad p^* = p^{m-1} + p'.$$

The first condition we impose on the corrected fields is to satisfy the following equation

$$A_D U_i^{**} + A_{OD} U_i^* = Q_i - G_i(p^*). \tag{5.10}$$

Subtracting equation (5.9) from (5.10), a relationship between the velocity and the pressure correction can be obtained. Note that because A_D is diagonal, it is trivial to obtain its inverse.

$$U_i' = -(A_D)^{-1}G_i(p')$$

The second condition imposed is that the corrected velocity satisfies the continuity equation, which yields a Poisson equation for the pressure correction.

$$D(\rho(A_D)^{-1}G_i(p')) = D(\rho U^*)$$

This process can be repeated by correcting the pressure and velocity again. Each additional correction will be denoted by an extra symbol in the super index.

$$A_D U_i^{***} + A_{OD} U_i^{**} = Q_i - G_i(p^{**}), \tag{5.11}$$

Subtracting equation (5.10) from (5.11) a relationship can be obtained between corrections

$$U_i'' = -(A_D)^{-1}(A_{OD} U_i' + G_i(p'')). \tag{5.12}$$

Forcing the corrected velocity to satisfy the continuity equation, a Poisson equation is obtained for the pressure correction.

$$D(\rho(A_D)^{-1} G_i(p'')) = D(\rho(A_D)^{-1} A_{OD} U_i') \tag{5.13}$$

The process can be repeated, and all that needs to be changed is to add a super index in the corrections of equations (5.12) and (5.13). At the end of each outer iteration, the transport equations of the turbulence model are solved.

$$\frac{(\rho\omega)^{n+1} - (\rho\omega)^n}{\Delta t} = \frac{\partial}{\partial x_i}\left[\rho\left(\nu + \alpha_\omega \nu_t\right)\frac{\partial \omega^{n+1}}{\partial x_i}\right] + \rho\gamma \frac{\partial U_i^{n+1}}{\partial x_j}\frac{\partial U_i^{n+1}}{\partial x_j} - \beta\rho\omega^n\omega^{n+1},$$
$$\frac{(\rho k)^{n+1} - (\rho k)^n}{\Delta t} = \frac{\partial}{\partial x_i}\left[\rho\left(\nu + \alpha_k \nu_t\right)\frac{\partial k^{n+1}}{\partial x_i}\right] + \rho\nu_t \frac{\partial U_i^{n+1}}{\partial x_j}\frac{\partial U_i^{n+1}}{\partial x_j} - \beta^*\rho\omega^{n+1}k^{n+1},$$

Figure 5.7 shows the flow chart of the PIMPLE algorithm. There are three main controls per time step: how many outer iterations are done, whether the momentum equation is solved to obtain a prediction of the velocity (or the previous velocity is used instead) and how many corrections are done. For this work, 2 outer iterations are used, the momentum equation is not solved, and 3 corrections are used.

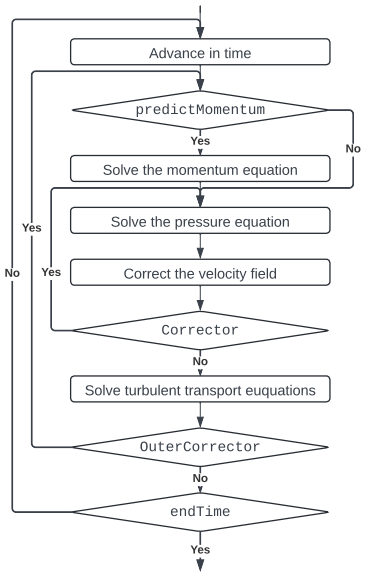


Figure 5.7: Flow chart of the PIMPLE algorithm.

5.2.2. Variable density turbulence model

OpenFOAM's incompressibleVof solver uses a single turbulence model for the whole mixture with a constant density. However, the mixture density is not constant. This issue was addressed in [10] for the multiphaseEuler solver, which uses a turbulent model for each phase but also fails to account for the effect of the variable density. The authors of the paper conclude that including the variable density effect improves the solver's accuracy. Therefore, we also address this issue in our model by introducing a source term in the turbulence transport equations as in [10].

$$\frac{D\left(\rho k\right)}{Dt} = \frac{\partial}{\partial x_i}\left[\rho\left(\nu + \frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_i}\right] + \rho\nu_t \frac{\partial U_i}{\partial x_j}\frac{\partial U_i}{\partial x_j} - \rho\beta^*\omega k + \rho\Delta_k, \tag{5.14}$$

$$\frac{D(\rho\omega)}{Dt} = \frac{\partial}{\partial x_i}\left[\rho\left(\nu + \frac{\nu_t}{\sigma_\omega}\right)\frac{\partial\omega}{\partial x_i}\right] + \rho\gamma\frac{\partial U_i}{\partial x_j}\frac{\partial U_i}{\partial x_j} - \rho\beta\omega^2 + \rho\Delta_\omega, \tag{5.15}$$

$$\frac{Dk}{Dt} = \frac{\partial}{\partial x_i}\left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_i}\right] + \nu_t\frac{\partial U_i}{\partial x_j}\frac{\partial U_i}{\partial x_j} - \beta^*\omega k + \Delta_k + S_k, \tag{5.16}$$

$$\frac{D\omega}{Dt} = \frac{\partial}{\partial x_i}\left[\left(\nu + \frac{\nu_t}{\sigma_\omega}\right)\frac{\partial\omega}{\partial x_i}\right] + \gamma\frac{\partial U_i}{\partial x_j}\frac{\partial U_i}{\partial x_j} - \beta\omega^2 + \Delta_\omega + S_\omega, \tag{5.17}$$

The equations we want the solver to use are (5.14) and (5.15). However, the equations implemented in OpenFOAM are (5.16) and (5.17), where S_k and S_ω are the source terms to be included.

$$\frac{Dk}{Dt} + \frac{k}{\rho}\frac{D\rho}{Dt} = \frac{\partial}{\partial x_i}\left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_i}\right] + \frac{1}{\rho}\frac{\partial\rho}{\partial x_i}\left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_i}\right] + \nu_t\frac{\partial U_i}{\partial x_j}\frac{\partial U_i}{\partial x_j} - \beta^*\omega k + \rho\Delta_k, \tag{5.18}$$

$$\frac{D\omega}{Dt} + \frac{\omega}{\rho}\frac{D\rho}{Dt} = \frac{\partial}{\partial x_i}\left[\left(\nu + \frac{\nu_t}{\sigma_\omega}\right)\frac{\partial\omega}{\partial x_i}\right] + \frac{1}{\rho}\frac{\partial\rho}{\partial x_i}\left[\left(\nu + \frac{\nu_t}{\sigma_\omega}\right)\frac{\partial\omega}{\partial x_i}\right] + \gamma\frac{\partial U_i}{\partial x_j}\frac{\partial U_i}{\partial x_j} - \beta\omega^2 + \rho\Delta_\omega, \tag{5.19}$$

Expanding equations (5.14) and (5.15) we obtain (5.18) and (5.19) and comparing them with (5.16) and (5.17) yields the following source terms

$$S_k = \frac{1}{\rho}\frac{\partial\rho}{\partial x_i}\left[\left(\nu + \frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_i}\right] - \frac{k}{\rho}\frac{D\rho}{Dt}.$$

$$S_\omega = \frac{1}{\rho}\frac{\partial\rho}{\partial x_i}\left[\left(\nu + \frac{\nu_t}{\sigma_\omega}\right)\frac{\partial\omega}{\partial x_i}\right] - \frac{\omega}{\rho}\frac{D\rho}{Dt}.$$

Note that the material derivative of the density is zero, $\frac{D\rho}{Dt} = 0$; thus, that term can be omitted.

5.3. Simplified model solver

This section introduces the Python solver for the simplified model. We first discuss the details of the spatial discretizations and the implementation of the boundary conditions. Then, we follow with the iterative methods to solve the non-linear system of equations. Finally, a comparison is made between the simplified solver and OpenFOAM.

The simplified model equations are

$$\frac{d}{dy}\left[\rho(\nu + \nu_t)\frac{dU}{dy}\right] = \frac{dp}{dx}, \tag{5.20}$$

$$\frac{d}{dy}\left[\rho(\nu + \alpha_k\nu_t)\frac{dk}{dy}\right] = \beta^*\rho\omega k - \rho\nu_t\left(\frac{dU}{dy}\right)^2 - \rho\Delta_k, \tag{5.21}$$

$$\frac{d}{dy}\left[\rho(\nu + \alpha_\omega\nu_t)\frac{d\omega}{dy}\right] = \beta\rho\omega^2 - \rho\gamma\left(\frac{dU}{dy}\right)^2 - \rho\Delta_\omega, \tag{5.22}$$

$$\nu_t = \frac{k}{\omega}.$$

These equations are spatially discretized using finite differences. The one-dimensional domain $\Omega = [-H, H]$ is divided into N cells. The cell centres are given by

$$x_i = -H + (0.5 + i)h, \qquad i = 0, 1, \ldots, N-1,$$

and the face centres of the cells are given by

$$x_{i-\frac{1}{2}} = -H + ih, \qquad i = 0, 1, \ldots, N,$$

where $h = 2H/N$ is the cell size. The equations are discretized in the cell centres. For the Laplacian, the following second-order central difference formula is used

$$\frac{d}{dy}\left[\kappa\frac{d\phi}{dy}\right]_{x=x_i} = \frac{1}{h^2}\left(\kappa_{i-\frac{1}{2}}\phi_{i-1} - \left(\kappa_{i-\frac{1}{2}} + \kappa_{i+\frac{1}{2}}\right)\phi_i + \kappa_{i+\frac{1}{2}}\phi_{i+1}\right).$$

For the first derivative terms, second-order central differences are also used

$$\left.\frac{d\phi}{dy}\right|_{x=x_i} = \frac{1}{2h}\left(\phi_{i+1} - \phi_{i-1}\right).$$

Fields are interpolated between the cell centres and the cell faces using linear interpolation.

5.3.1. Boundary conditions and wall functions

The same boundary conditions must be used for the simplified model to be consistent with OpenFOAM. These boundary conditions are used in discretising the non-linear equations and for the interpolation of fields to the faces of the cells. In OpenFOAM, wall functions are used for turbulent quantities; these functions are algebraic relationships that provide the value of a field either in the wall boundary or the first cell centre. The implementation needs to be different depending on where the boundary conditions are provided. For U and k, the boundary condition is given in the wall; then, a linear extrapolation is used to obtain the value in the first cell centre outside the domain

$$\phi_{-1} = 2\phi_{-\frac{1}{2}} - \phi_0, \qquad \phi_N = 2\phi_{N-\frac{1}{2}} - \phi_{N-1}.$$

For ω, the value is given in the first cell centre, which can be directly used as the boundary condition. For ν_t the boundary value is given in the wall but it is only used to set the value in the boundary when the field is interpolated to the cell faces. The wall functions used are:

- **kLowReWallFunction**

C	C_ϵ	C_μ	`tol`
11	1.9	0.09	10^{-6}

Table 5.1: kLowReWallFunction parameters.

This wall function provides the value of the turbulent kinetic energy k in the wall boundaries. The formulas used are as follows.

$$\begin{aligned} k &= \max\left(k_{\text{vis}}u_\tau^2, \texttt{tol}\right), \\ k_{\text{vis}} &= \frac{2400C_f}{C_\epsilon^2}, \\ u_\tau &= C_\mu^{0.25}\sqrt{k}, \end{aligned} \qquad \begin{aligned} C_f &= \frac{1}{(y^+ + C)^2} + \frac{2y^+}{C^3} - \frac{1}{C^2}, \\ y^+ &= \frac{u_\tau y}{\nu}. \end{aligned}$$

The parameters used are given in table 5.1, and y is the distance to the wall of the centre of the first cell.

- **omegaWallFunction**

  This wall function provides the value of ω in the centre of the first cell. The formula used is

$$\omega = \frac{6\nu}{\beta_1 y^2},$$

  where $\beta_1 = 0.09$.

- **nutLowReWallFunction**

  This wall function simply sets the value of ν_t to zero in the wall boundaries.

These wall functions are valid as long as the first cell centre is in the viscous sublayer of the boundary layer, which can be formulated as

$$y^+ < y^+_{\text{lam}},$$

where $y^+_{\text{lam}} \approx 3.886$ and

$$y^+ = y\frac{\sqrt{(\nu + \nu_t)\frac{dU}{dy}}}{\nu}.$$

5.3.2. Fixed-point iteration and time-dependent solver.

The model equations are solved in two ways: the first is to perform a fixed-point iteration, and the second is to solve the time-dependent version of the equations until the stationary state is reached. For each method, we will first discuss the discretization of the equations and then present a pseudocode of the algorithm.

Fixed-point iteration solver

We start by dicretizing the ω equation (5.21) into the form

$$A_\omega \omega = b_\omega,$$

where

$$A^n_\omega = \frac{d}{dy}\left[\rho\left(\nu + \alpha_\omega \nu^n_t\right)\frac{d}{dy}\right] - \beta\rho\omega^n,$$

and

$$b^n_\omega = -\rho\gamma\left(\frac{dU^n}{dy}\right)^2 - \rho\Delta^n_\omega.$$

For a given iteration, the residual can be computed as

$$r^n_\omega = b^n_\omega - A^n_\omega \omega^n.$$

The same is done for the k equation (5.20) with

$$A^n_k = \frac{d}{dy}\left[\rho\left(\nu + \alpha_k \nu^n_t\right)\frac{d}{dy}\right] - \beta^*\rho\omega^{n+1},$$

and

$$b^n_k = -\rho\nu^n_t\left(\frac{dU^n}{dy}\right)^2 - \rho\Delta^n_k.$$

Again, the residual is computed as

$$r^n_k = b^n_k - A^n_k k^n.$$

Lastly, the momentum equation (5.22) is also dicretized, with

$$A^n_U = \frac{d}{dy}\left[\rho\left(\nu + \nu^{n+1}_t\right)\frac{d}{dy}\right],$$

and

$$b^n_U = \frac{dp}{dx}.$$

The residual is

$$r^n_U = b^n_U - A^n_U U^n.$$

Algorithm 2 `stationary_solver`

for $i \in \{1, \ldots, \texttt{max\_iter}\}$ **do**
  ▷ Solve the ω equation
  $r_\omega^n \leftarrow b_\omega^n - A_\omega^n \omega^n$
  $\delta\omega^{n+1} \leftarrow (A_\omega^n)^{-1} \alpha_\omega r_\omega^n$
  $\omega^{n+1} \leftarrow \max(\omega^n + \delta\omega^{n+1}, \omega_{\min})$
  ▷ Solve the k equation
  $r_k^n \leftarrow b_k^n - A_k^n k^n$
  $\delta k^{n+1} \leftarrow (A_k^n)^{-1} \alpha_k r_k^n$
  $k^{n+1} \leftarrow \max(k^n + \delta k^{n+1}, k_{\min})$
  ▷ Update ν_t
  $\nu_t^{n+1} \leftarrow k^{n+1}/\omega^{n+1}$
  ▷ Solve the U equation
  $r_U^n \leftarrow b_U^n - A_U^n U^n$
  $\delta U^{n+1} \leftarrow (A_U^n)^{-1} \alpha_U r_U^n$
  $U^{n+1} \leftarrow U^n + \delta U^{n+1}$
  ▷ Check the stopping criterion
  if $\|r^n\|/\|b^n\| < \texttt{tol}$ **then**
    break

Algorithm 2 presents the solving procedure where the stopping criterium is computed using the norm of the residual normalized by the norm of the right-hand side of the equations.

$$r^n = \begin{pmatrix} r_\omega^n \\ r_k^n \\ r_U^n \end{pmatrix}, \qquad b^n = \begin{pmatrix} b_\omega^n \\ b_k^n \\ b_U^n \end{pmatrix}.$$

The solver parameters include relaxation parameters α_ω, α_k, α_U set to 0.5, the lower bounds for ω and k, $\omega_{\min}$ and $k_{\min}$ set to 10^{-16}, and the tolerance of the stopping criterion set to $\texttt{tol} = 10^{-10}$.

Time-dependent solver

The discretization of the equations for the time-dependent solver is the same as for the fixed point iteration, but adding the corresponding time derivative term. For the ω equation (5.21) we have

$$\rho\frac{\omega^{n+1} - \omega^n}{\Delta t} = \frac{d}{dy}\left[\rho\left(\nu + \alpha_\omega \nu_t^n\right)\frac{d\omega^{n+1}}{dy}\right] + \rho\gamma\left(\frac{dU^n}{dy}\right)^2 - \beta\rho\omega^n\omega^{n+1} + \rho\Delta_\omega^n,$$

which can be rewritten as

$$\left(\frac{\rho}{\Delta t}I + A_\omega^n\right)\omega^{n+1} = b_\omega^n + \frac{\rho}{\Delta t}\omega^n.$$

Similarly for the k equation (5.20)

$$\rho\frac{k^{n+1} - k^n}{\Delta t} = \frac{d}{dy}\left[\rho\left(\nu + \alpha_k \nu_t^n\right)\frac{dk^{n+1}}{dy}\right] + \rho\nu_t^n\left(\frac{dU^n}{dy}\right)^2 - \beta^*\rho\omega^{n+1}k^{n+1} + \rho\Delta_k^n,$$

which can be rewritten as

$$\left(\frac{\rho}{\Delta t}I + A_k^n\right)k^{n+1} = b_k^n + \frac{\rho}{\Delta t}k^n.$$

And for the momentum equation (5.22)

$$\rho\frac{u^{n+1} - u^n}{\Delta t} = \frac{d}{dy}\left[\rho\left(\nu + \nu_t^{n+1}\right)\frac{dU^{n+1}}{dy}\right] - \frac{dp}{dx},$$

rewritten as

$$\left(\frac{\rho}{\Delta t}I + A_U^n\right)U^{n+1} = b_U^n + \frac{\rho}{\Delta t}U^n.$$

Instead of the residual, the difference in the solution compared to the last step is used as a stopping the criterium.

$$\delta^{n+1} = \begin{pmatrix} \delta_\omega^{n+1} \\ \delta_k^{n+1} \\ \delta_U^{n+1} \end{pmatrix} = \begin{pmatrix} \delta_\omega^{n+1} - \delta_\omega^n \\ \delta_k^{n+1} - \delta_k^n \\ \delta_U^{n+1} - \delta_U^n \end{pmatrix}.$$

Algorithm 3 summarizes the solving procedure of the time-dependent solver.

Algorithm 3 `time_dependent_solver`

for $t \in \{t_0, \ldots, t_{\max}\}$ **do**
  ▷ Update ω
  $\omega^{n+1} \leftarrow (\rho/\Delta t + A_\omega^n)^{-1} (b_\omega^n + \omega^n \rho/\Delta t)$
  $\delta\omega^{n+1} \leftarrow \omega^{n+1} - \omega^n$
  $\omega^{n+1} \leftarrow \max(\omega^{n+1}, \omega_{\min})$
  ▷ Update k
  $k^{n+1} \leftarrow (\rho/\Delta t + A_k^n)^{-1} (b_k^n + k^n \rho/\Delta t)$
  $\delta k^{n+1} \leftarrow k^{n+1} - k^n$
  $k^{n+1} \leftarrow \max(k^{n+1}, k_{\min})$
  ▷ Update ν_t
  $\nu_t^{n+1} \leftarrow k^{n+1}/\omega^{n+1}$
  ▷ Update U
  $U^{n+1} \leftarrow (\rho/\Delta t + A_U^n)^{-1} (b_U^n + U^n \rho/\Delta t)$
  $\delta U^{n+1} \leftarrow U^{n+1} - U^n$
  ▷ Check the stopping criterion
  if $\|\delta^{n+1}\|/\|b^n\| <$ `tol` **then**
    break

5.3.3. Validation of the simplified model solver.

Figure 5.8 shows the converged solution for a particular simulation of both solvers, where we can see that both solutions are equivalent.

Finally, figure 5.9 shows the solution of the simplified model and OpenFOAM's solver using constant and variable density for the turbulent transport equations. As we can see, both models are equivalent when the density is treated the same way, but there are clear differences between using constant density and variable density.

5.4. Test case: stratified flow with non-deforming interface

We will use a stratified flow channel with a non-deforming interface as a test case to model the interface's effect. This setup allows us to isolate and examine the interface's interactions without deformation-related complications. By focusing on a simplified yet representative scenario, we can gain valuable insights into the fundamental dynamics at the interface and how they influence the overall flow behaviour.

The channel consists of two parallel walls that extent infinitely in the x and z directions. However, such a domain can not be used for simulations, so the channel is reduced to a rectangular prism in which the boundaries in the x and z directions are cyclic. To be able to perform the necessary simulations to generate the data, the density ratio has been kept in the order of 10^1 using the scenario developed in [47] as a starting point.

ν_L	ν_G	Height	Length	Width
$1.886 \cdot 10^{-5}$	$1.886 \cdot 10^{-5}$	0.1	0.4	0.2

Table 5.2: Fixed parameters.

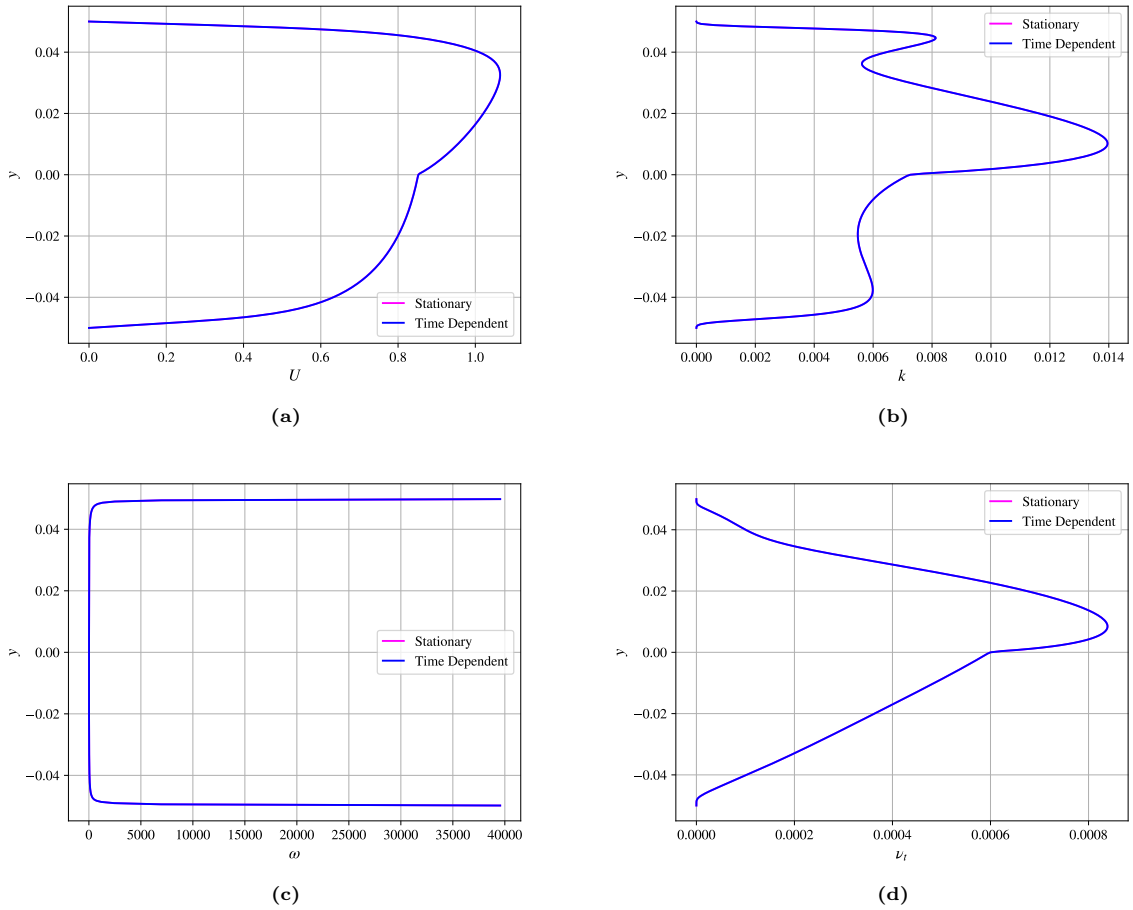


Figure 5.8: Solution of the simplified model (U (a), k (b), ω (c), ν_t (d). Simulation parameters: $\rho_l = 10$, $\rho_g = 1$, $\nu_l = 1.886 \cdot 10^{-5}$, $\nu_g = 1.886 \cdot 10^{-5}$, $dp/dx = 0.5$.

Table 5.2 shows the common parameters for all the stratified flow with non-deforming interface simulations. These are the dimensions of the channel and the kinematic viscosity of both fluids. On the other hand, table 5.3 shows the parameters that define each of the specific simulations. These are obtained by varying the density ratio and the forcing of the channel.

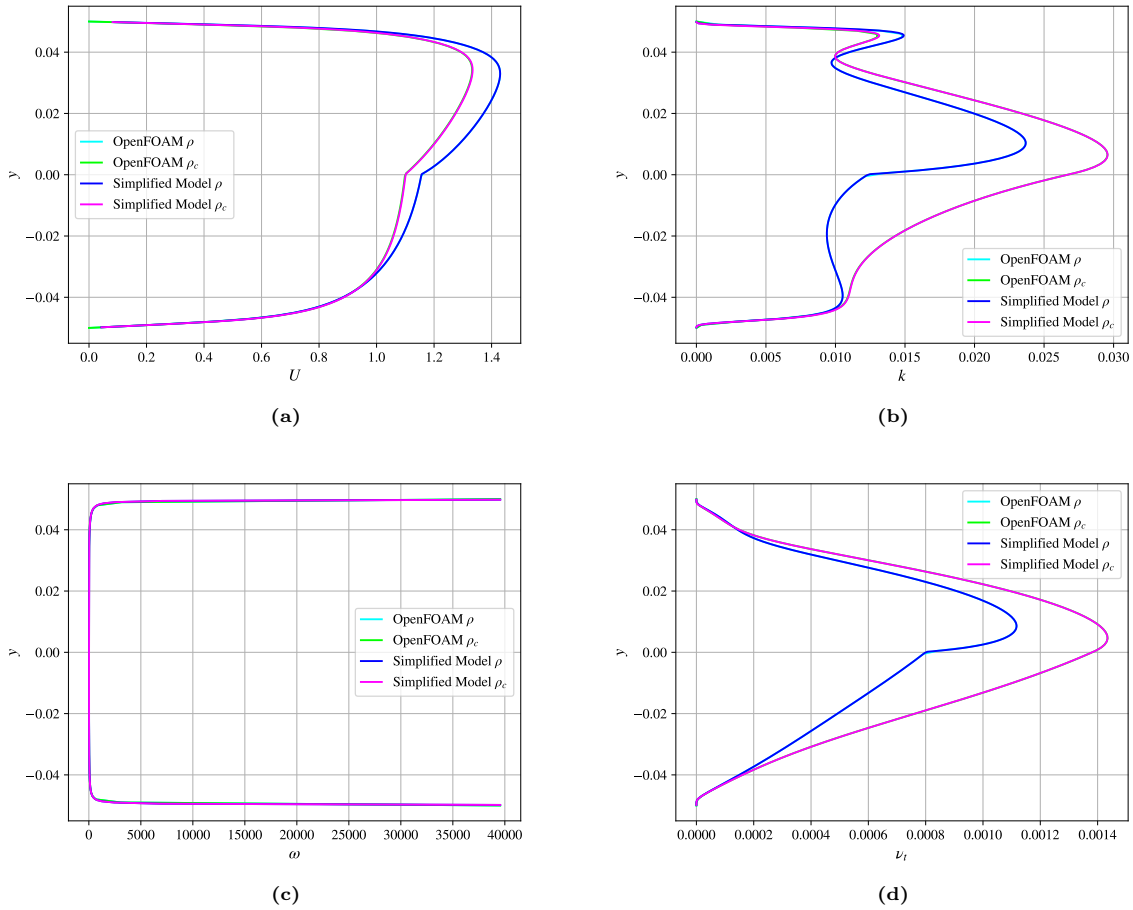


Figure 5.9: Solution of the simplified and RANS model (U (a), k (b), ω (c), ν_t (d). Simulation parameters: $\rho_l = 10$, $\rho_g = 1$, $\nu_l = 1.886 \cdot 10^{-5}$, $\nu_g = 1.886 \cdot 10^{-5}$, $dp/dx = 0.5$.

Case	ρ_L	ρ	$\frac{dp}{dx}$
1	10	1	0.3
2	10	1	0.5
3	10	5	0.3
4	10	5	0.5
5	11	1	0.3
6	11	1	0.5
7	10	0.9	0.3
8	10	0.9	0.5
9	10	2	0.3
10	10	2	0.5
11	10	3	0.3
12	10	3	0.5
13	9	1	0.3
14	9	1	0.5
15	7	1	0.3
16	7	1	0.5

Table 5.3: Case parameters.

Figure 5.10 shows a schematic of the different domains that are used in the DNS, RANS and simplified model simulations. While the DNS setup uses a 3D domain, the RANS simulations are done

in a 2D domain, and the simplified model uses a 1D domain.

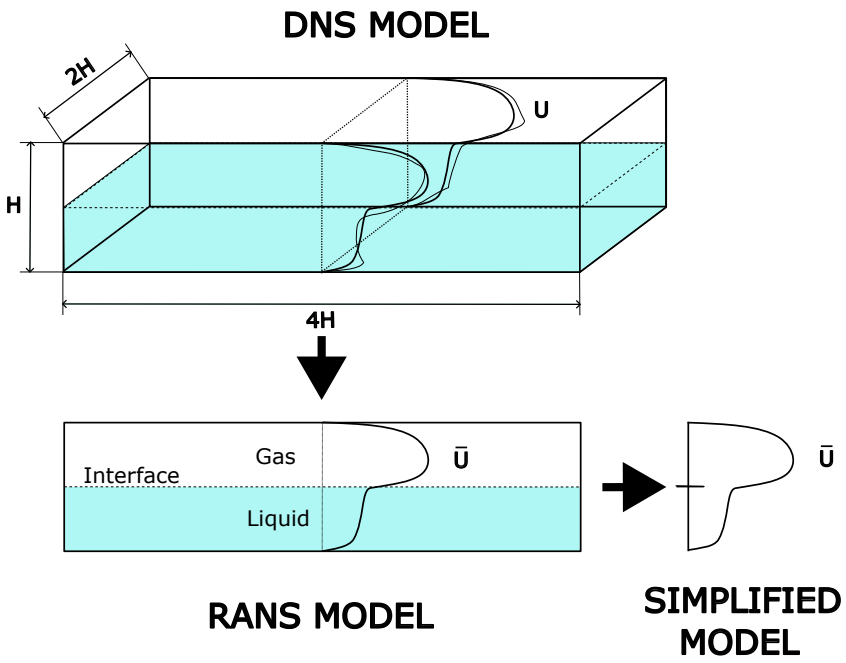
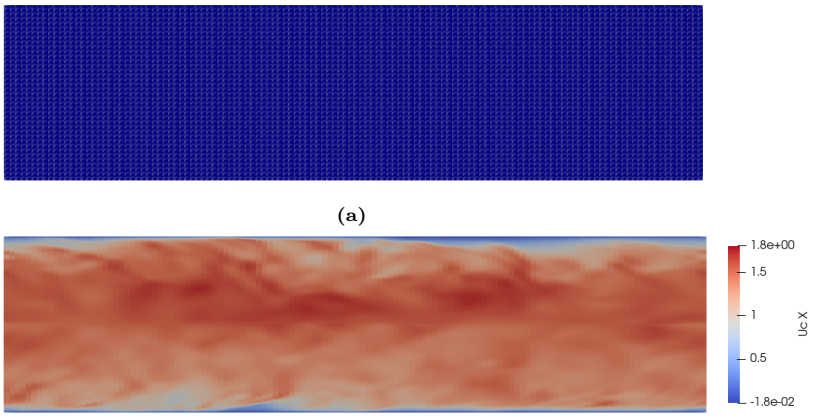


Figure 5.10: Diagram of the different mathematical models of the channel.

DNS

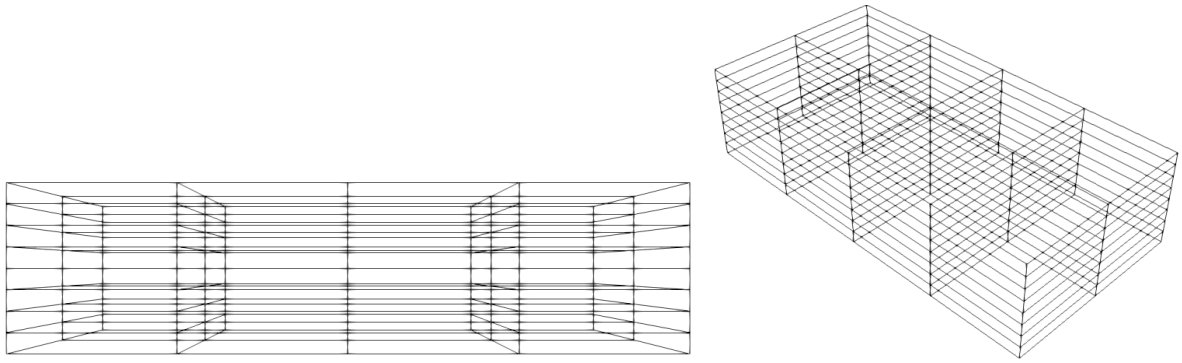
The DNS simulations are performed using Briscola. Figure 5.11 shows the mesh used for the simulations. It is a uniform structured mesh composed of hexahedral cells. There are 128 cells in the x direction, 512 in the y direction and 64 cells in the z direction. The mesh is decomposed in 64 patches for parallelization as shown in figure 5.12.



(a)

(b)

Figure 5.11: Briscola mesh for the stratified flow channel with fixed interface case (a) and streamline velocity (b) of case 1 at time $t = 5s$.



(a) (b)

Figure 5.12: Briscola mesh decomposition.

The boundary conditions of the velocity, u_i, pressure, p, and volume of fluid field, α, are given in table 5.4.

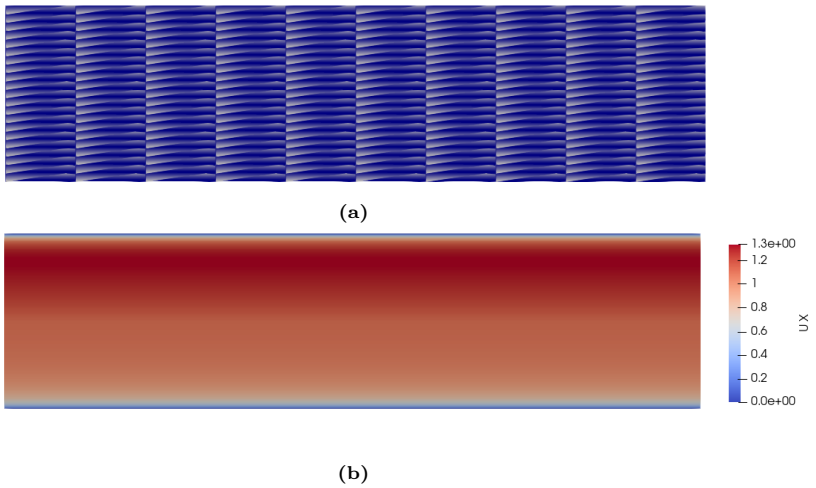
	Inlet	Outlet	Walls	Sides
u_i	Periodic	Periodic	No Slip	Periodic
p	Periodic	Periodic	Zero Gradient	Periodic
α	Periodic	Periodic	Zero Gradient	Periodic

Table 5.4: Boundary conditions used in the DNS simulations of the stratified flow FI case.

The boundary condition "No Slip" refers to the homogeneous Dirichlet boundary condition, where the fluid velocity at the boundary is set to zero, indicating that the fluid has no relative motion with respect to the boundary. The "Zero Gradient" boundary condition corresponds to the homogeneous Neumann boundary condition, where the gradient of the field variable is set to zero, implying no flux across the boundary. Periodic boundary conditions mean that the field variables at one boundary are equal to the field variables at the opposite boundary, creating a repeating or cyclic pattern in the simulation domain.

RANS

The 2D RANS simulations are performed using OpenFOAM. Figure 5.13 shows the mesh used for the simulations. It is a uniform structured mesh composed of hexahedral cells. There are 10 cells in the x direction and 256 in the y direction.



(a)

(b)

Figure 5.13: OpenFOAM mesh for the stratified flow channel with fixed interface case (a) and streamline velocity (b) of case 1 at time $t = 30s$.

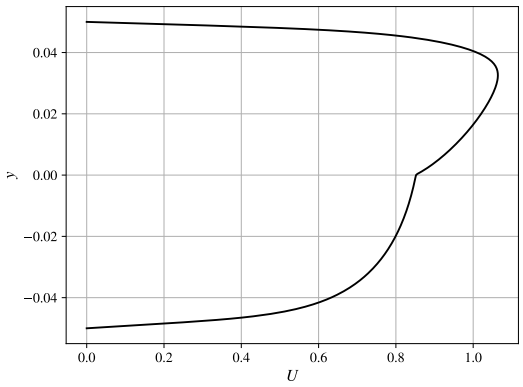
The boundary conditions for the average velocity, U_i, the pressure, p, the volume of fluid field, α, the turbulent kinetic energy, k, the turbulent dissipation rate, ω, and the turbulent viscosity, ν_t are given in table 5.5.

	Inlet	Outlet	Walls	Sides
U_i	Periodic	Periodic	No Slip	Empty
p	Periodic	Periodic	Zero Gradient	Empty
α	Periodic	Periodic	Zero Gradient	Empty
k	Periodic	Periodic	kLowReWallFunction	Empty
ω	Periodic	Periodic	omegaWallFunction	Empty
ν_t	Periodic	Periodic	nutLowReWallFunction	Empty

Table 5.5: Boundary conditions used in the RANS simulations of the stratified flow FI case.

Simplified model

The simplified model simulations are performed in Python. Figure 5.14 shows the result of a simulation. A uniform one-dimensional mesh composed of 256 cells is used.



(a)

Figure 5.14: Streamline velocity of the stratified flow channel with fixed interface simplified model case 1 at time $t = 30s$.

The boundary conditions for the average velocity, U, the turbulent kinetic energy, k, the turbulent dissipation rate, ω, and the turbulent viscosity, ν_t are given in table 5.6.

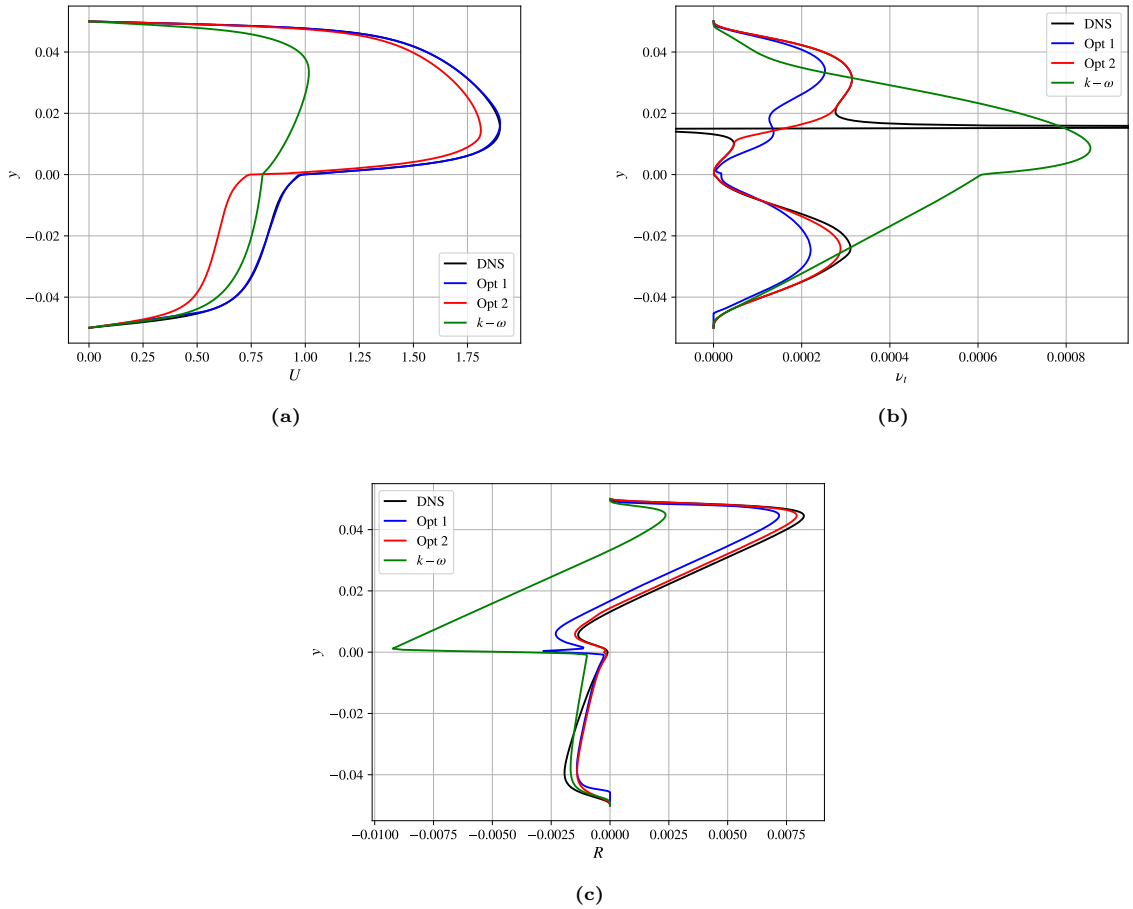
	Walls
U	No Slip
k	kLowReWallFunction
ω	omegaWallFunction
ν_t	nutLowReWallFunction

Table 5.6: Boundary conditions used in the simplified model simulations of the stratified flow FI case.

6 Results

In this section, we will discuss the results of the framework presented in chapter 4. First, we will examine the outcomes of the turbulent viscosity inversion and the resulting velocity fields. Next, we will explore the propagation of the corrections into the simplified model. Finally, we will present the regression of the corrections and the integration of these regressed models into the RANS solver, assessing the overall accuracy, robustness, and generalizability of the models found.

6.1. Turbulent viscosity field inversion



(a) **(b)** **(c)**

Figure 6.1: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 1. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*

Figure 6.1 shows the results of the turbulent viscosity inversion for case 1. The top right plot shows the turbulent viscosity obtained with both inversion methods as well as the turbulent viscosity obtained from the DNS data using formula (4.4) and the turbulent viscosity from the baseline $k-\omega$ model. It can be seen that both procedures result in similar viscosity fields where the value goes to zero near the interface. Furthermore, the profile obtained using the Reynolds stress optimization process closely matches the viscosity from the DNS, successfully smoothing the field in regions where the results from the DNS lead to unphysical values for ν_t. The plot on the top left shows the velocity profile obtained propagating the turbulent viscosity into the simplified momentum equation. The velocity profile obtained from the momentum equation field inversion better matches the results of the DNS, with the velocity profile obtained from the Reynolds stress optimization showing larger differences with the DNS profile. Lastly, the bottom plot shows the xy component of the Reynolds stress tensor, R_{12}. We can see that while both methods yield an improvement in the Reynolds stress tensor compared to the baseline model, the second approach results in a better approximation. Moreover, the smoothing of the unphysical peak in the turbulent viscosity does not result in a worse prediction of R_{12}.

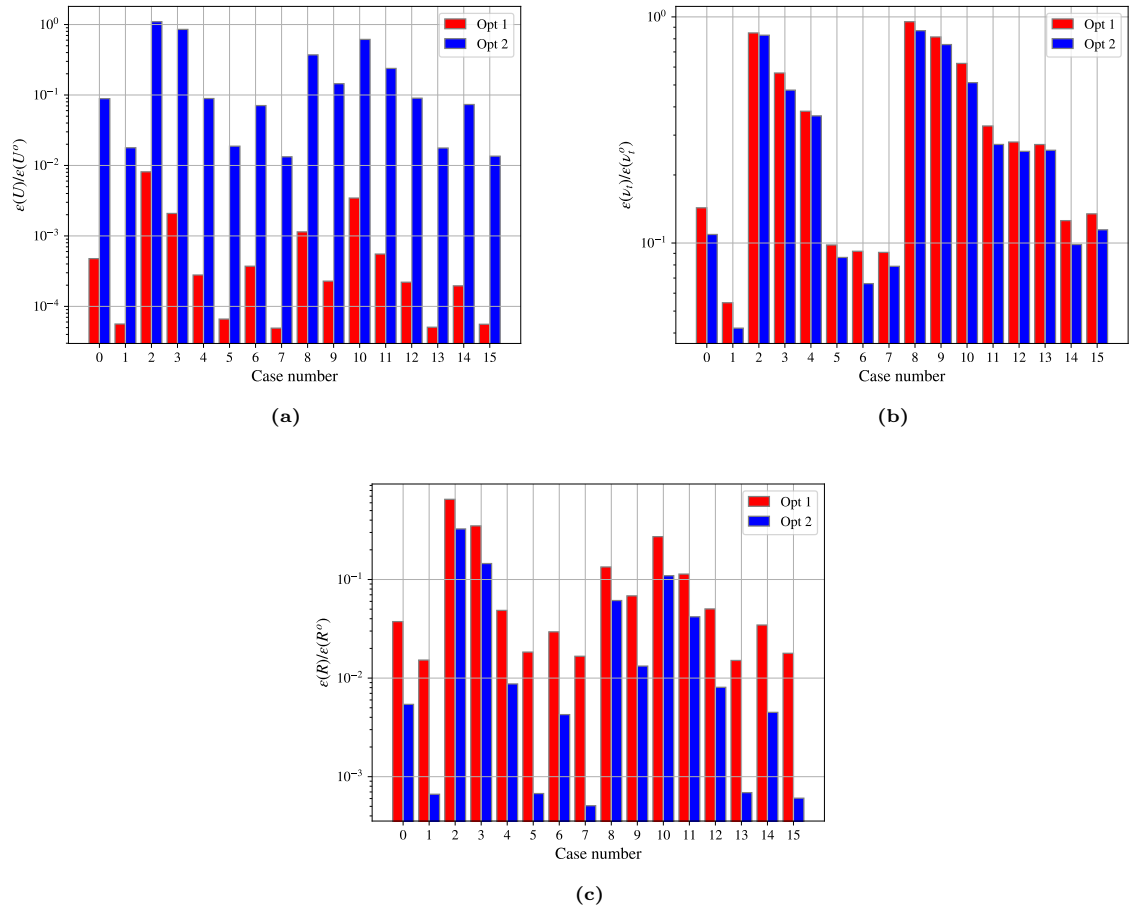


Figure 6.2: MSE error of the mean velocity (a), turbulent viscosity field (b) and xy component of the Reynolds stress tensor (c) of every case compared to the velocity profile obtained from the DNS and turbulent viscosity obtained from formula (4.4), normalized by the error of the baseline $k-\omega$ model. Legend: *Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*

Figure 6.2 summarizes the MSE error of the mean velocity, the turbulent viscosity and the Reynolds stress tensor for all cases. The plot on the top left shows the MSE of the velocity profile normalized by the MSE error of the baseline model. While both methods provide improvements compared to the baseline $k-\omega$ model, the momentum equation approach consistently yields lower errors. In the plot on the right, the MSE errors of the turbulent viscosity are shown. Here no significant differences can be drawn between both methods. On the bottom plot, we can see that the second viscosity optimization

approach yields lower errors in the Reynolds stress tensor.

From these results, it appears that the large velocity errors of the baseline model can be avoided under the Boussinesq approximation if the turbulence model predicts the adequate turbulent viscosity field. Moreover, using the momentum equation field inversion to obtain the viscosity yields lower errors in the velocity than obtaining the turbulent viscosity field from the Reynolds stress optimation problem and vice-versa for the error in the Reynolds stress.

6.2. Target correction propagation

It is also interesting to understand what happens when the corrections are propagated into the RANS model and what is sufficient to address the shortcomings of the model.

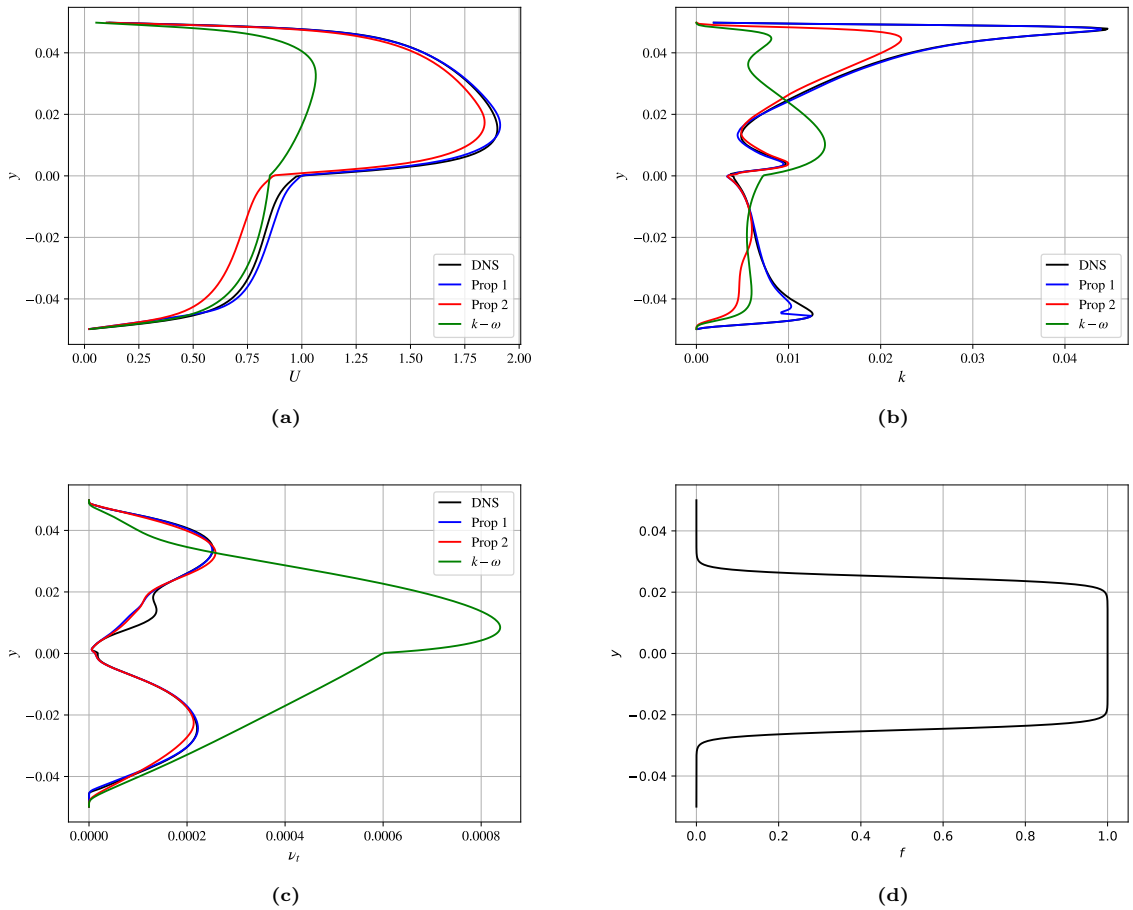


Figure 6.3: Solution of the velocity (a), turbulent kinetic energy (b) and turbulent viscosity (c) fields of the simplified model. $f(\boldsymbol{q}$ values (d). Legend: *DNS: velocity profile obtained from the DNS, turbulent kinetic energy of the DNS and turbulent viscosity obtained from formula* (4.4). *Prop 1: the full correction field is applied. Prop 2: the correction field is multiplied by* $f(\boldsymbol{q})$. $k-\omega$*: baseline model.*

The full correction fields obtained by equations (4.9) and (4.10) are added as source terms in the turbulent transport equations. Figure 6.3 shows the solution obtained for case 1. Two different propagation methods are used, denoted in the legend as Prop 1 and Prop 2. The full correction is used in the first one, and in the second one, the correction is multiplied by a function $f(\boldsymbol{q})$. Multiplying by this function, we aim to propagate the correction only in a certain domain region, such as the interface. For the remainder of this work, we will use this filter function

$$f(\boldsymbol{q}) = \frac{\exp(\beta * q_5)}{\exp(\beta * q_5) + \exp(\beta * q_6)},$$

where $\beta = 500$. Nevertheless, in future work, $f(\boldsymbol{q})$ could be the output of another model that identifies regions with different behaviours, so different models are applied in each region. Function $f(\boldsymbol{q})$ is shown in figure 6.3 (d). It simply is a softmax function applied to the wall and interface distance. So when the interface distance gets smaller than the wall distance, it tends to 1. Plots (a), (b) and (c) show the velocity, turbulent kinetic energy and turbulent viscosity respectively. It can be seen that when propagating the full correction, the obtained profiles closely match the DNS targets. When the correction is applied only in the region near the interface, the velocity and the turbulent viscosity are still similar to those obtained using the full correction. However, the obtained turbulent kinetic energy is clearly off in the region where the correction is not applied.

We argue that this is not an issue for modelling the interface effect but rather a result of how the $k - \omega$ model works near the boundaries. The baseline model is not calibrated to obtain the exact k profile in this region but to capture the velocity profile accurately. Ultimately, this error is not caused by two-phase phenomena and will not be further investigated in this work.

Figure 6.4 summarizes the error of all cases when the full correction field is propagated and when it is applied only in the region specified by $f(\boldsymbol{q})$. The top left plot shows the MSE error in the velocity field, the top right plot shows the MSE error of the turbulent kinetic energy and the bottom one the MSE of the turbulent viscosity. For most cases, correcting only near the interface yields the same error in velocity as correcting the whole domain; however, it leads to larger errors in the turbulent kinetic energy. As argued before, this is just a consequence of how the $k - \omega$ model works in the boundaries and is unrelated to the interface effect.

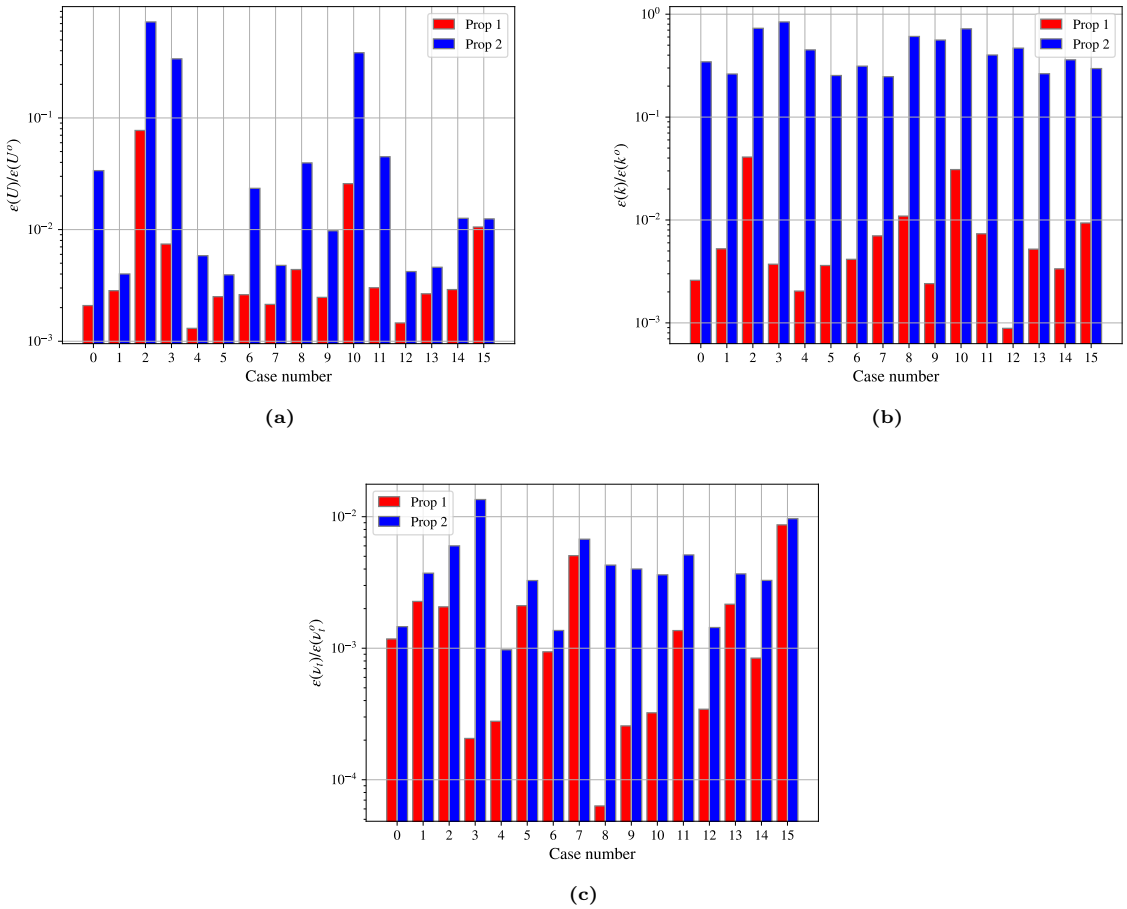


Figure 6.4: MSE error of the mean velocity (a), turbulent kinetic energy (b) and turbulent viscosity field (c) of every case compared to the velocity profile obtained from the DNS, turbulent kinetic energy of the DNS and turbulent viscosity obtained from formula (4.4), normalized by the error of the baseline $k - \omega$ model. Legend: *Prop 1: the full correction field is applied. Prop 2: the correction field is multiplied by* $f(\boldsymbol{q})$.

6.3. Model discovery

In this section, we present the results of the best-performing discovered models. We begin discussing the regression setup used. Then we continue by presenting the outcomes of the training phase, where the influence of the hyperparameters of the machine learning techniques is analysed. Following this, we examine the results of the propagation tests, assessing how well the discovered models perform when introduced in the system of non-linear PDEs.

Regression setup

As discussed in chapter 4, different choices can be made when setting up the regression problem. Here, we will introduce the most successful setup for the regressions as well as the influence of different choices made.

Linear or exponential model

Two different model forms were studied. In the exponential model, the logarithm of the corrections is regressed, while in the linear one, the corrections are directly regressed. Linear models led to worse data fits for Δ_ω than exponential models. Moreover, because of the difference in orders of magnitude of the corrections for different areas of the flow, linear models overfitted to the areas of the highest magnitude. This issue was mitigated with exponential models.

$$\Delta_k = f(\boldsymbol{q})\bar{\Delta}_k e^{g_k(\boldsymbol{q})}, \quad \Delta_\omega = f(\boldsymbol{q})\bar{\Delta}_\omega e^{g_\omega(\boldsymbol{q})},$$

Dimensional factors

Different choices for the dimensional factors $\bar{\Delta}_k$ and $\bar{\Delta}_\omega$ were investigated. For $\bar{\Delta}_k$, the available options are $k\omega$ and $\nu_t(dU/dy)^2$. The latter option is multiplied by ν_t, which for the studied cases tends to zero near the interface, the same location where the correction Δ_k has its biggest value. Therefore, $\nu_t(dU/dy)^2$ was deemed as inadequate; instead, $k\omega$ was chosen, providing good data fits in the training phase.

The investigated factors for $\bar{\Delta}_\omega$ are ω^2 and $(dU/dy)^2$. In this case, the former one provided better results in the training phase of the model discovery. However, the discovered models with this dimensional factor had poor performance when propagated in the RANS equations. Thus, $(dU/dy)^2$ was chosen as the dimensional factor.

Zone training

No noticeable differences in training data fit and propagation results were observed between using zone training and not using it. However, when no training was used, sparse symbolic regression models contained the wall distance in many of the expression terms. This hints at either some kind of overfitting or that the models are also learning about the effect of the boundary. Given that the model is supposed to correct near the interface, this is undesirable. Therefore, while no difference was noticeable in the results, zone training was used to try to prevent this issue. Moreover, If the objective is to address the effect of the interface, it appears more reasonable to exclude sample points in which the main source of error is due to different phenomena, such as the boundary layer.

6.3.1. Training

In this subsection, the results of the k-fold cross-validation of the model discovery are presented. For the sparse symbolic regressors, we show the validation R^2 scores, and the number of terms the expressions have, which is equivalent to the number of non-zero components of the regressed coefficient vector. For the NN models, only the R^2 scores are presented. The R^2 score is defined as

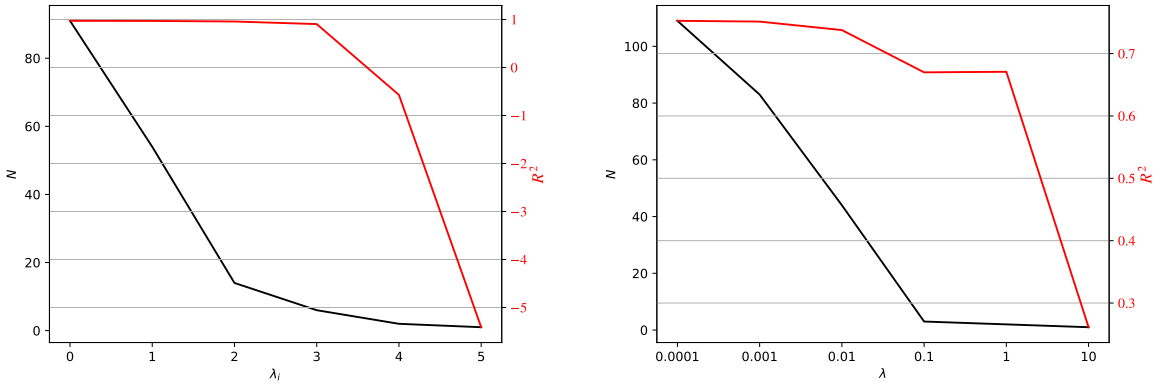
$$R^2 = 1 - \frac{\sum_i (\Delta_{i,true} - \Delta_{i,pred})^2}{\sum_i (\Delta_{i,true} - \Delta_{mean})^2},$$

where Δ_{mean} is the mean of all $\Delta_{i,true}$ observations. The final R^2 score is the average of the score in each k-fold. For conciseness, only the results of the LASSO regressor and the JNN models are presented here and the results of the other regressors are shown in appendix A.1.

LASSO

Figure 6.5 shows the R^2 validation scores and the number of terms of the discovered models using the LASSO regression. As expected, we see that for both Δ_k and Δ_ω, the number of non-zero terms decreases as the value of the regularization parameter λ becomes larger. Validation scores are in general higher for Δ_k than for Δ_ω, this trend will persist with all regression techniques. We also see that a larger value of the regularization parameter leads to lower validation scores. This could be due to different factors:

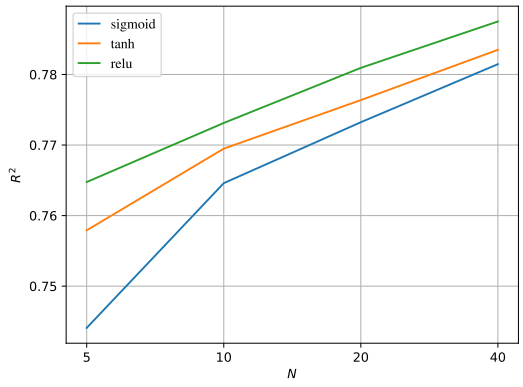
- A model that is general enough to model the effect of the interface can not be found with very few terms. As the regularization gets larger, the expressions become more sparse, and they can not capture all phenomena adequately.
- The training and validation cases may be too similar. Regularization aims to avoid overfitting; however, if the validation data is very similar to the validation data, overfitting will not lead to low validation scores.



(a) **(b)**

Figure 6.5: Number of terms (black) and R^2 validation score (red) of the discovered models for Δ_k (a) and Δ_ω (b) using the LASSO regressor depending on the value of the hyperparameter λ.

JNN



(a)

Figure 6.6: Validation R^2 score for $[\Delta_k, \Delta_\omega]$ using JNN.

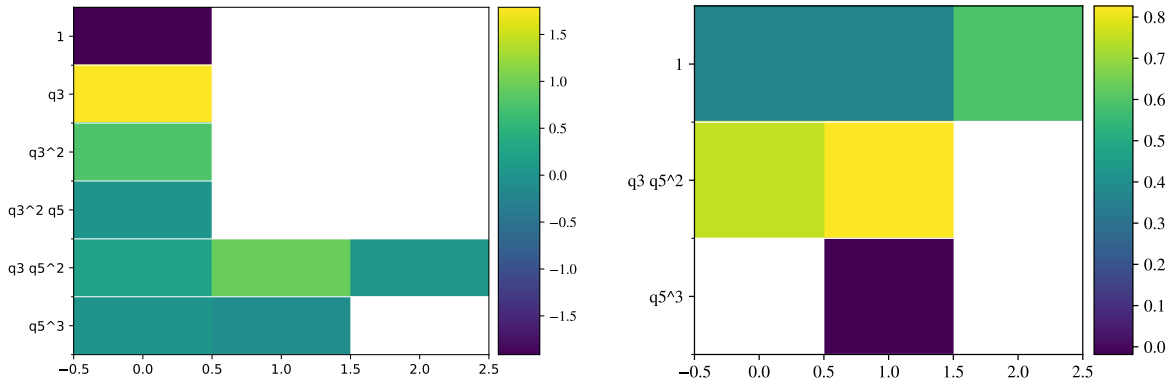
Figure 6.6 shows the R^2 validation scores of the trained joint NNs models for different types of activation functions and number of neurons per hidden layer. The same activation function and number

of neurons have been used in both hidden layers. The best-performing activation function is Relu, followed by the tanh function and the sigmoid activation function. The validation score increases as the number of neurons per layer increases. We would expect that as the number of parameters of the model increases, the probability of overfitting increases, too, but the results do not show a decrease in validation accuracy. As discussed before, this might not be because the model is not overfitting but instead, be a consequence of the training and validation cases being very similar.

6.3.2. Testing

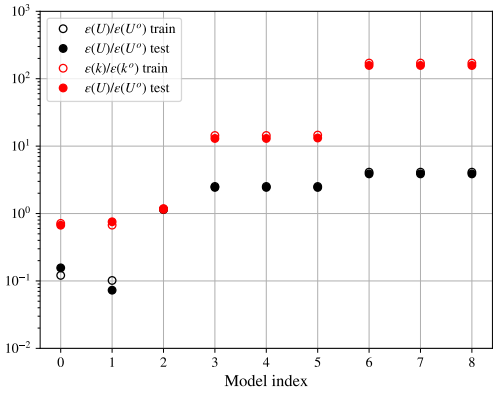
This section presents the results of the propagation of the discovered models for the training and test cases. For conciseness, only the results of the LASSO regressor and the SNN models will be discussed in detail here and the results of the other regressors are discussed in appendix A.2.

LASSO



(a) (b)

Figure 6.7: Non-zero coefficients of the propagated symbolic expressions for Δ_k (a) and Δ_ω (b) of the discovered models using the LASSO regressor. The x-axis is the model index (a lower index indicates a higher validation score), and the y-axis shows the terms associated with the non-zero coefficients.



(a)

Figure 6.8: Average MSE error of the velocity and the turbulent kinetic energy of the propagated LASSO models. The model index is $i = i_\omega \cdot n_\omega + i_k$, where the i_k and i_ω are the model indices for Δ_k and Δ_ω given in figure 6.7 and n_ω is the number of propagated Δ_ω models.

Figure 6.7 shows the Δ_k (left panel) and Δ_ω (right panel) models with the best validation scores discovered using the LASSO regression. The figure shows the values of the coefficients associated with

the terms on the y-axis for the different models x-axis. Here we can see that the correction only depends on features q_3 and q_5. Feature q_5 represents the normalized wall distance, which is a feature we might not want to include in a model that is supposed to work near the interface. One possible cause for the appearance in q_5, is that all cases have the same geometry, so the machine learning algorithm might be overfitting to this geometry. If we exclude q_5 from the input features, we obtain similar results, albeit with different terms in the regressed expressions.

Figure 6.8 shows the MSE error of the velocity and turbulent kinetic energy (normalized by the MSE error of the baseline model) for the propagated models in the training cases and in the testing cases. We can see that the velocity errors are lower than those of the turbulent kinetic energy. Moreover, the errors in the testing cases are, in general, equal to those in the training cases. This could mean that the learnt expression is general enough to make accurate predictions in unseen cases, but as we argued in the previous section, it is more likely that the cause is that the training and testing cases are very similar.

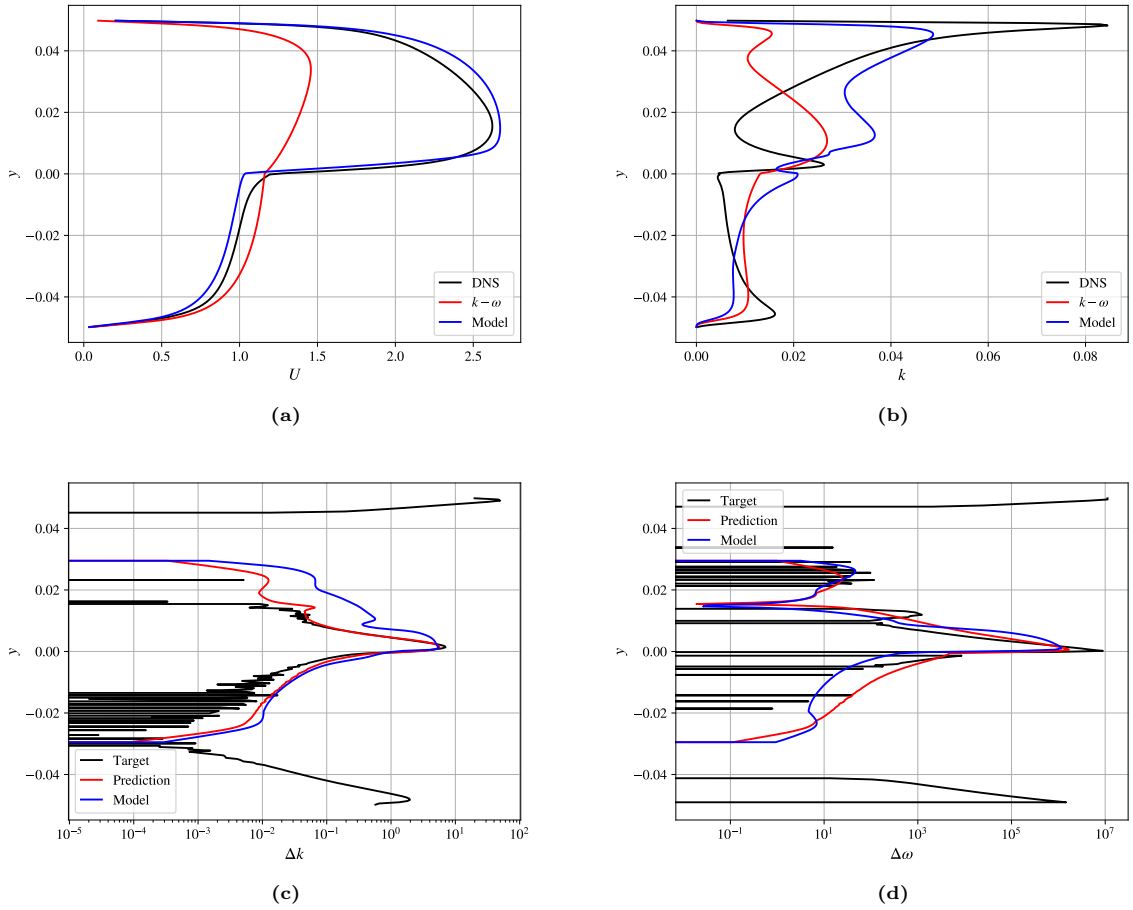


Figure 6.9: Velocity (a), turbulent kinetic energy (b), Δ_k (c) and Δ_ω (d) for case 8. Legend panels (a) and (b): *DNS: DNS results. $k-\omega$: baseline model. Model: LASSO regressed model.* Legend panels (c) and (d): *Target: Regression targets. Prediction: correction prediction in the training phase. Model: prediction field in the converged state of the simplified model.* (Following figure 6.7, model 0 for Δ_k and model 1 for Δ_ω)

Figure 6.9 shows the results of the simplified model for the best-performing LASSO discovered correction for one of the test cases. The figure compares the results of the DNS simulation, the baseline model and the discovered model. The two top panels show the velocity and turbulent kinetic energy. It can be seen that the discovered model greatly reduces the error compared to the baseline model. However, there are still errors, especially for the turbulent kinetic energy. The bottom two panels show the correction fields. These panels show that the data fit of the corrections is worse in the converged state of the simplified model equations than in the training phase, thus also obtaining worse predictions

of the velocity and turbulent kinetic energy. It is particularly noticeable in the Δ_k correction, where the overestimation of the correction in the gas phase directly leads to an overestimation of k. It is important to note that in this figure the results are shown for one of the cases with the lowest MSE errors, but in general the error might be larger.

SNN

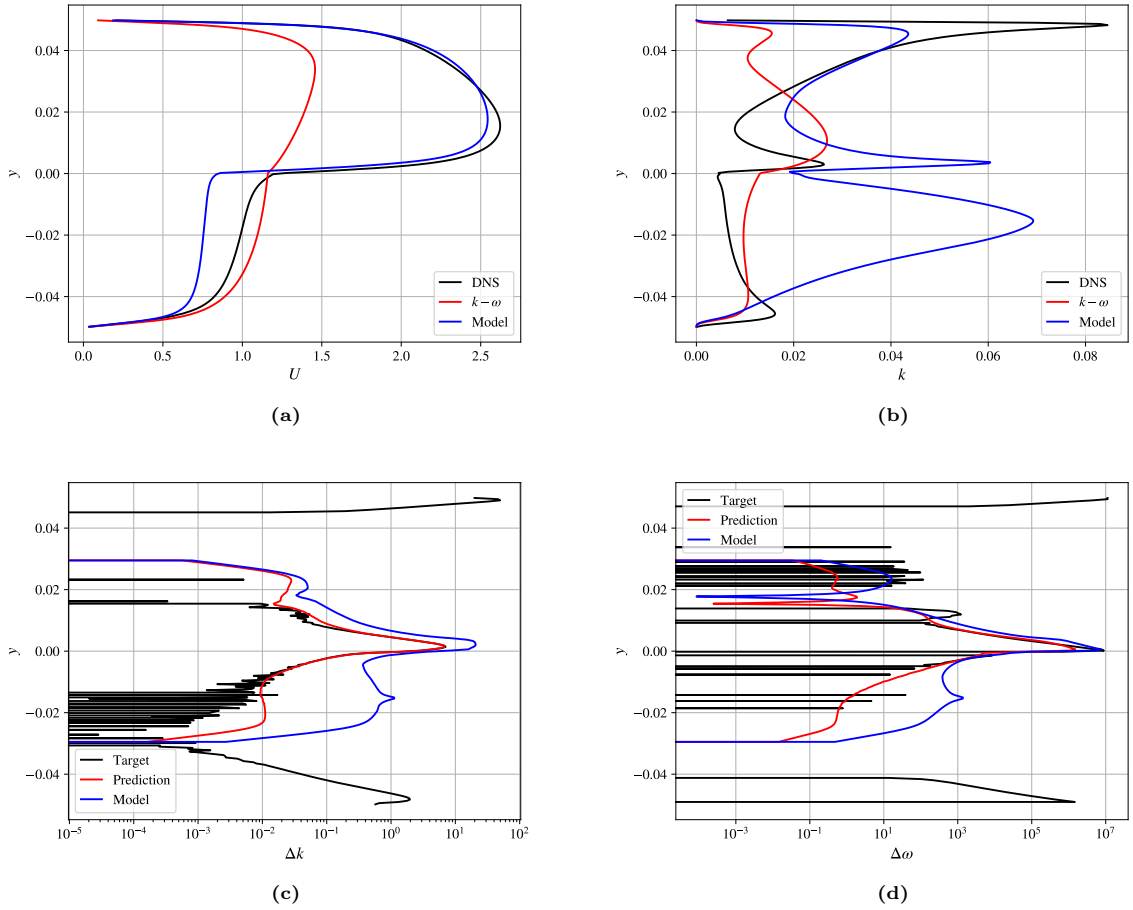
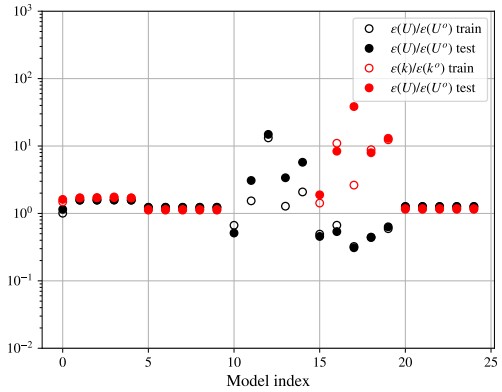


Figure 6.10: Velocity (a), turbulent kinetic energy (b), Δ_k (c) and Δ_ω (d) for case 8. Legend panels (a) and (b): *DNS: DNS results. $k-\omega$: baseline model. Model: SNN regressed model.* Legend panels (c) and (d): *Target: Regression targets. Prediction: correction prediction in the training phase. Model: prediction field in the converged state of the simplified model.*

Figure 6.11 shows the average MSE error of the velocity and turbulent kinetic energy for the propagated SNN models in the training cases and in the testing cases. No discovered model reduces the error of the turbulent kinetic energy compared to the baseline model and only a few models improve the velocity error. Moreover, for some cases, the turbulent kinetic energy is hugely overpredicted leading to very large MSE errors for k. Note that in order to not distort figure 6.11 too much the y-limit has been set to 10^3, but there are some models for which the k MSE error is larger than that. Also note, that many of the propagated models do not really alter the baseline too much and have MSE errors close to 1.

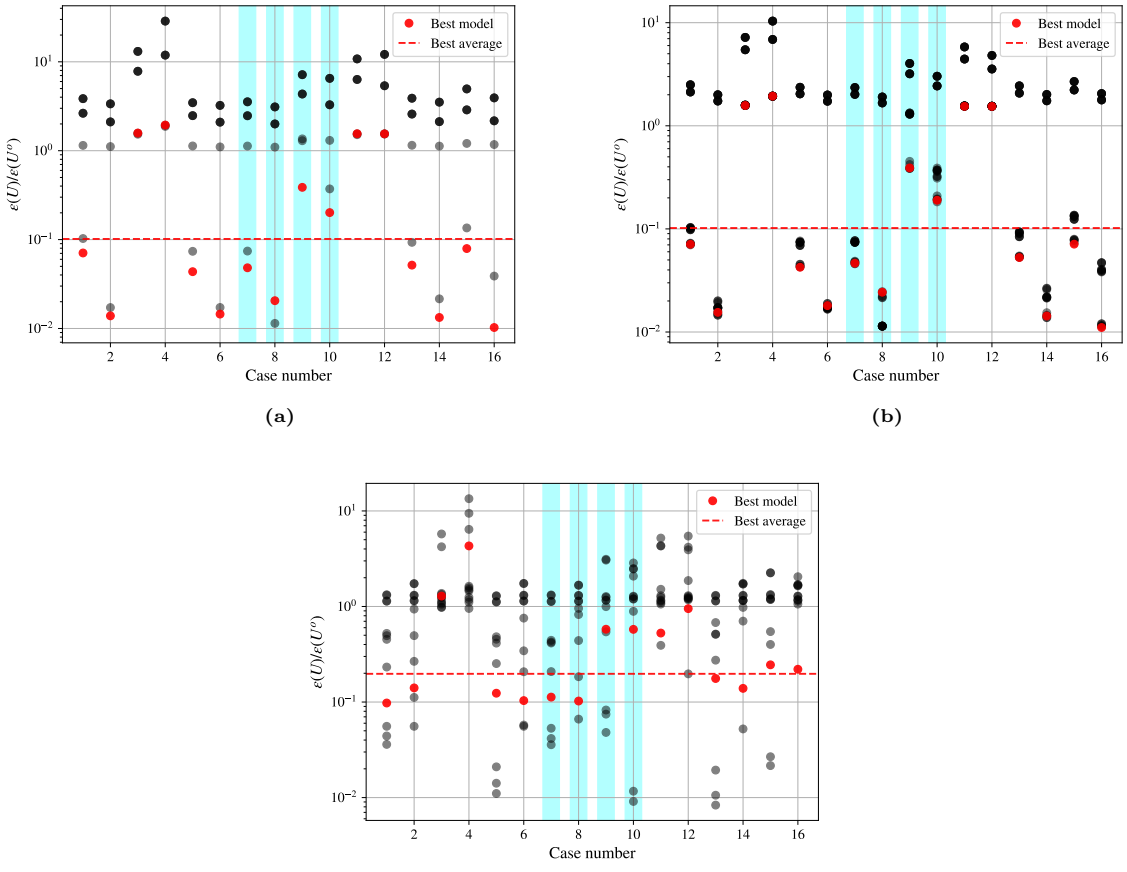
Figure 6.10 shows the results of the propagation of the best model discovered using SNN for case 8, which was not used for training. While the data fit is good in training, once the model is introduced in the RANS equations it overpredicts both corrections, especially in the liquid phase. This is then very noticeable in the turbulent kinetic energy profile, which is hugely overestimated in the liquid phase.



(a)

Figure 6.11: Average MSE error of the velocity and the turbulent kinetic energy of the propagated SNN models in the training and testing cases.

Comparison between all regressors



(a) **(b)**

(c)

Figure 6.12: Mean squared error of the propagation velocity field compared to the DNS and normalized by the MSE of the baseline model for different sparse symbolic regressors (LASSO (a), Elastic Net (b) and SLTSQ (c)). The x-axis is the number of the case, where the testing cases are highlighted in blue. Each grey dot corresponds to the propagation error of one model. The errors of the best model on average for each regression technique are shown in red, and the average error of that model is shown with a red dashed line.

Figures 6.12 and 6.13 show a summary of the MSE of the velocity for all discovered models using sparse symbolic regression and NNs respectively. The best performing methods are LASSO and elastic net, obtaining models that reduce the baseline error by one order of magnitude, however, models discovered with those methods struggle with cases 3, 4, 11 and 12 (those with the densest gas phase). While some NN models appear to be a clear improvement compared to the baseline model, we have seen before that the qualitative behaviour of these models is not that good.

Figure 6.12 shows the MSE errors of the velocity for each case for the models discovered with each of the sparse symbolic regression techniques. While 6.13 shows the same information for the NN models.

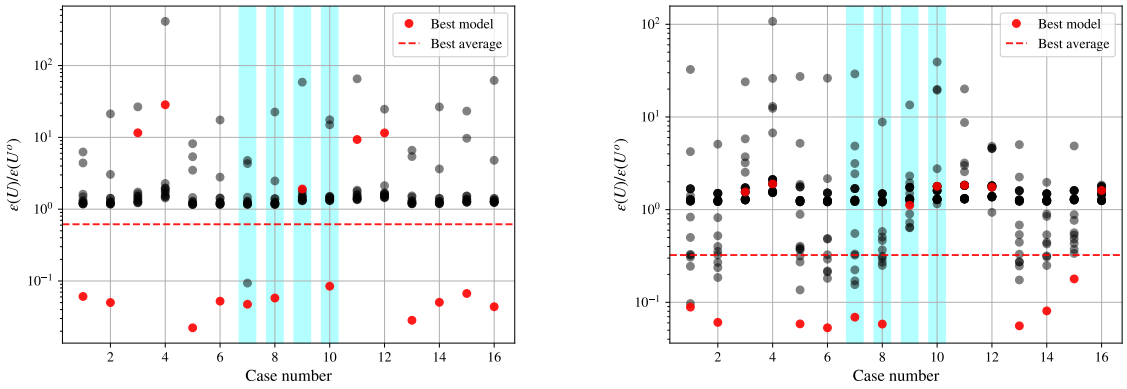


Figure 6.13: Mean squared error of the propagation velocity field compared to the DNS and normalized by the MSE of the baseline model for different NN methods (Joint NN (a) and Separated NNs (b)). The x-axis is the number of the case, where the testing cases are highlighted in blue. Each grey dot corresponds to the propagation error of one model. The errors of the best model on average for each regression technique are shown in red, and the average error of that model is shown with a red dashed line.

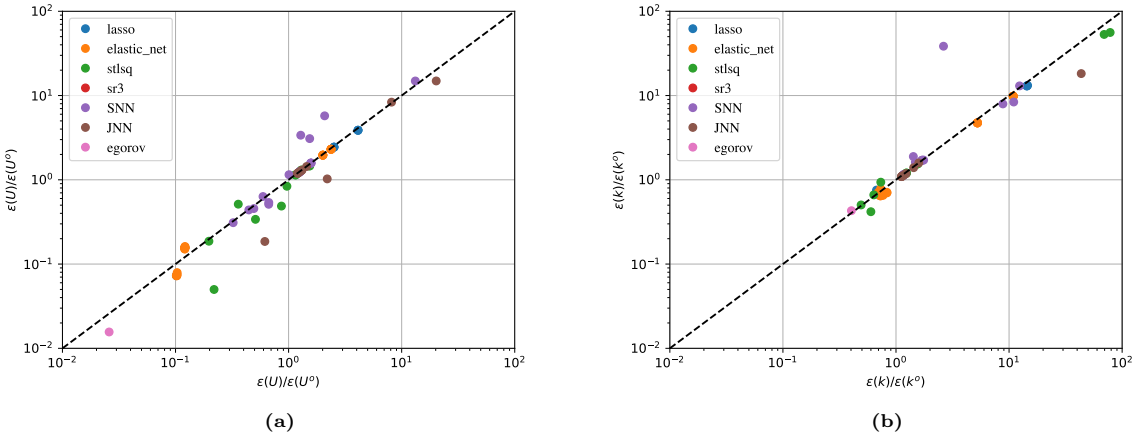


Figure 6.14: Comparison of the average MSE of the velocity (a) and turbulent kinetic energy (b) fields in training and testing cases for different discovered models.

Figure 6.14 shows the average MSE error of the velocity (left panel) and turbulent kinetic energy (right panel) of all found models using different regression techniques. For comparison, the average MSE error using Egorov damping [11] has also been included. Each point represents one discovered model, with the x coordinate being the MSE of the training cases and the y coordinate being the MSE of the testing cases. This means that the points that lie above the $x = y$ line represent models which generalize poorly, and points below this line represent models that perform unexpectedly well in unseen data. However, as we have discussed before, the testing and training cases used in this project are not

diverse enough to draw conclusions about the generalizability of the models.

The plots show that the correction models reduce the MSE of the velocity more than the MSE of the turbulent kinetic energy. Moreover, the model with the best accuracy is the Egorov damping model, in which a source is introduced in the omega equation. This means that Egorov simple approach is able to outperform all discovered models. We can also see that in general symbolic expression models perform better than NN models.

7

Conclusions and recommendations

A novel machine learning framework to discover correction models for RANS models in turbulent stratified gas-liquid flows while still employing the Boussinesq approximation is introduced. This framework includes two methods for performing the turbulent viscosity field inversion and introduces two correction terms in the turbulence model equations to ensure accurate prediction of the turbulent viscosity field. The framework is able to discover models which improve the accuracy of the baseline model, even in flow cases that have not been used for training, although the generalization capabilities for more varied flow regimes remain to be tested. In section 7.1, the conclusions of this research are presented, and in section 7.2, recommendations for future work are given.

7.1. Conclusions

The framework proposed in this thesis has shown significant potential in discovering correction models that improve the accuracy of baseline RANS models in turbulent stratified gas-liquid flows. The sparse symbolic regression models derived from the framework performed better than the NN models.

An important aspect of this research was developing a simplified model for inverting turbulent viscosity field. The proposed methods demonstrated that improving the turbulent viscosity field is sufficient to address most of the model's shortcomings while still using the Boussinesq approximation. Two methods were developed to perform the turbulent viscosity field inversion, one with a focus on minimizing the error of the velocity and the other one focusing on the error of the Reynold stress tensor. Both methods yielded turbulent viscosities, which resulted in better predictions of the velocity field than the baseline model.

The sparse symbolic regression models consistently improved the accuracy of the baseline model. Moreover, these models are easily interpretable and do not lead to robustness issues when propagated in the RANS equations. Out of all sparse symbolic regression techniques, LASSO and elastic net yielded the most successful models, reducing the baseline error by one order of magnitude. On the other hand, the SR3 regressor could not discover any model with only a small number of non-zero coefficients.

One of the proposed framework's key strengths is its ability to produce numerically stable models. These robust models ensured that the simulations converged without numerical issues, which is crucial for practical applications. This is in part achieved by staying under the Boussinesq approximation.

While the discovered models improved the baseline model, they were unable to outperform the Egorov damping approach [11]. The Egorov model, which introduces a source term in the omega equation, remained the superior model in terms of accuracy. This suggests that while the discovered models are a step forward, more improvement needs to be done before they can match the performance of the Egorov damping approach.

One of the main drawbacks of the proposed framework is that models that yield good data fits in training do not always result in good results when introduced in the RANS equations. This is an important limitation since the training of the models focuses on obtaining good data fits. The consequence of this is that discovered models need to be propagated in the RANS equations, which, as the discovered number of models increases, can become computationally costly.

While the discovered models were able to improve the results of the baseline model in cases not used for training, the variety of simulated cases for this project was very limited. Therefore, the generalizability of the approach for more varied flow regimes remains to be explored.

Neural network models performed worse than symbolic expression models. Not only were NNs unable to improve the accuracy achieved by the regressed symbolic expressions, but they also lack

interpretability. Moreover, the computational cost of training these models was higher. Therefore, the use of NNs is unjustified as they did not outperform sparse symbolic regressions in any aspect, at least within this research.

7.2. Recommendations

Based on the findings of this thesis, several recommendations can be made for future research. We will divide this between improvements to the current framework and topics that could be investigated outside the current framework.

Improvements to the current framework

- **Generating data for more flow regimes.** Performing DNS simulations requires a large amount of computational resources. Therefore, the number and, more importantly, the variety of simulated cases for this project was very limited. The less variety in the used data sets for the model discovery framework, the more likely the found models will not be generalizable enough, and the machine learning algorithms will overfit the available data. Therefore, we suggest performing more DNS simulations to have a richer data set and, in particular, to include cases in more varied flow regimes to increase the generalizability of the regressed models.
- **Improving the input features and the candidate function library.** As was explained in chapter 4, the input features for the regressions and the functions used in the candidate functions library must be manually set. Therefore, the model discovery is limited to this manual choice. Further work could be done to analyze the impact of including more features and different candidate functions.

Further research

- **Including the RANS equations in the training process.** One of the main shortcomings of the current framework is the apparent lack of correlation between good data fits in the training phase and good results in the propagation of the regressed models. This means that the posterior behaviour of the models can not be deduced by the apriori fit, and therefore, many models need to be propagated into the RANS equations to assess whether they are an improvement from the baseline model. While one of these simulations can be performed quickly, the process scales poorly when many models are propagated.

  A possible approach to address this issue is to include the model equations in the training phase. In [20, 21, 22, 23] the authors propose different methods to do this for single phase flows. For this purpose. the authors use a cost function that depends on the results of the propagation of the machine learning models for the training of the models and then use different techniques to compute the gradient of the cost function with respect to the model parameters. These techniques range from ensemble gradient methods to solving adjoint equations.

References

[1] J.H. Spurk and N. Aksel. *Fluid Mechanics*. Springer International Publishing, 2019. ISBN: 9783030302597. URL: https://books.google.nl/books?id=617BDwAAQBAJ.

[2] S.B. Pope. *Turbulent Flows*. Cambridge University Press, 2000. ISBN: 9780521598866.

[3] F.T.M. Nieuwstadt, J. Westerweel, and B.J. Boersma. *Turbulence: Introduction to Theory and Applications of Turbulent Flows*. Springer International Publishing, 2016. ISBN: 9783319315973.

[4] D.C. Wilcox. *Turbulence Modeling for CFD*. DCW Industries, Incorporated, 1994. ISBN: 9780963605108.

[5] M. Ishii and T. Hibiki. *Thermo-fluid Dynamics of Two-Phase Flow*. Smart energy systems–nanowatts to terawatts. Springer US, 2006. ISBN: 9780387291871.

[6] F. Durst S. Banerjee R. Krahl and Ch. Zenger. "Presentation of anisotropy properties of turbulence, invariants versus eigenvalue approaches". In: *Journal of Turbulence* 8 (2007), N32. DOI: 10.1080/14685240701506896.

[7] Michael S. Dodd and Antonino Ferrante. "A fast pressure-correction method for incompressible two-fluid flows". In: *Journal of Computational Physics* 273 (2014), pp. 416–434. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2014.05.024.

[8] Vinesh H. Gada and Atul Sharma. "On Derivation and Physical Interpretation of Level Set Method–Based Equations for Two-Phase Flow Simulations". In: *Numerical Heat Transfer, Part B: Fundamentals* 56.4 (2009), pp. 307–322. DOI: 10.1080/10407790903388258.

[9] P. Coste and J. Laviéville. "A turbulence model for large interfaces in high Reynolds two-phase CFD". In: *Nuclear Engineering and Design* 284 (2015), pp. 162–175. ISSN: 0029-5493. DOI: https://doi.org/10.1016/j.nucengdes.2014.12.004.

[10] Z. Dong, M. Bürgler, B. Hohermuth, and D.F. Vetsch. "Density-based turbulence damping at large-scale interface for Reynolds-averaged two-fluid models". In: *Chemical Engineering Science* 247 (2022), p. 116975. ISSN: 0009-2509. DOI: https://doi.org/10.1016/j.ces.2021.116975.

[11] E.M.A. Frederix, A. Mathur, D. Dovizio, B.J. Geurts, and E.M.J. Komen. "Reynolds-averaged modeling of turbulence damping near a large-scale interface in two-phase flow". In: *Nuclear Engineering and Design* 333 (2018), pp. 122–130. ISSN: 0029-5493. DOI: https://doi.org/10.1016/j.nucengdes.2018.04.010.

[12] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. "Machine Learning for Fluid Mechanics". In: *Annual Review of Fluid Mechanics* 52.1 (2020), pp. 477–508. DOI: 10.1146/annurev-fluid-010719-060214.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[14] Cândida Ferreira. "Gene Expression Programming: a New Adaptive Algorithm for Solving Problems". In: *CoRR* cs.AI/0102027 (2001). URL: https://arxiv.org/abs/cs/0102027.

[15] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (Mar. 2016), pp. 3932–3937. ISSN: 1091-6490. DOI: 10.1073/pnas.1517384113.

[16] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. "Turbulence Modeling in the Age of Data". In: *Annual Review of Fluid Mechanics* 51.1 (2019), pp. 357–377. DOI: 10.1146/annurev-fluid-010518-040547.

[17] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance". In: *Journal of Fluid Mechanics* 807 (2016), pp. 155–166. DOI: 10.1017/jfm.2016.615.

[18] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. "Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data". In: *Phys. Rev. Fluids* 2 (3 Mar. 2017), p. 034603. DOI: 10.1103/PhysRevFluids.2.034603.

[19] Mikael L.A. Kaandorp and Richard P. Dwight. “Data-driven modelling of the Reynolds stress tensor using random forests with invariance”. In: *Computers Fluids* 202 (2020), p. 104497. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2020.104497.

[20] Carlos Michelén Ströfer, Jinlong Wu, Heng Xiao, and Eric Paterson. “Data-Driven, Physics-Based Feature Extraction from Fluid Flow Fields”. In: *Communications in Computational Physics* 25 (Feb. 2018), pp. 625–650. DOI: 10.4208/cicp.OA-2018-0035.

[21] Xin-Lei Zhang, Heng Xiao, Xiaodong Luo, and Guowei He. “Ensemble Kalman method for learning turbulence models from indirect observation data”. In: *Journal of Fluid Mechanics* 949 (2022), A26. DOI: 10.1017/jfm.2022.744.

[22] Yu Zhang, Richard P. Dwight, Martin Schmelzer, Javier F. Gómez, Zhong-hua Han, and Stefan Hickel. “Customized data-driven RANS closures for bi-fidelity LES–RANS optimization”. In: *Journal of Computational Physics* 432 (2021), p. 110153. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2021.110153.

[23] Xin-Lei Zhang, Heng Xiao, Solkeun Jee, and Guowei He. “Physical interpretation of neural network-based nonlinear eddy viscosity models”. In: *Aerospace Science and Technology* 142 (2023), p. 108632. ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2023.108632.

[24] Carlos A. Michelén Ströfer and Heng Xiao. “End-to-end differentiable learning of turbulence models from indirect observations”. In: *Theoretical and Applied Mechanics Letters* 11.4 (2021), p. 100280. ISSN: 2095-0349. DOI: https://doi.org/10.1016/j.taml.2021.100280.

[25] Anand Pratap Singh and Karthik Duraisamy. “Using field inversion to quantify functional errors in turbulence closures”. In: *Physics of Fluids* 28.4 (Apr. 2016), p. 045110. DOI: 10.1063/1.4947045.

[26] Martin Schmelzer, Richard Dwight, and Paola Cinnella. “Discovery of Algebraic Reynolds-Stress Models Using Sparse Symbolic Regression”. In: *Flow Turbulence and Combustion* (Mar. 2020). DOI: 10.1007/s10494-019-00089-x.

[27] Hannes Mandler and Bernhard Weigand. “On frozen-RANS approaches in data-driven turbulence modeling: Practical relevance of turbulent scale consistency during closure inference and application”. In: *International Journal of Heat and Fluid Flow* 97 (2022), p. 109017. ISSN: 0142-727X. DOI: https://doi.org/10.1016/j.ijheatfluidflow.2022.109017.

[28] Hannes Mandler and Bernhard Weigand. “A realizable and scale-consistent data-driven non-linear eddy viscosity modeling framework for arbitrary regression algorithms”. In: *International Journal of Heat and Fluid Flow* 97 (2022), p. 109018. ISSN: 0142-727X. DOI: https://doi.org/10.1016/j.ijheatfluidflow.2022.109018.

[29] Ruiying Xu, Xu-Hui Zhou, Jiequn Han, Richard P. Dwight, and Heng Xiao. “A PDE-free, neural network-based eddy viscosity model coupled with RANS equations”. In: *International Journal of Heat and Fluid Flow* 98 (2022), p. 109051. ISSN: 0142-727X. DOI: https://doi.org/10.1016/j.ijheatfluidflow.2022.109051.

[30] Xu-Hui Zhou, Jiequn Han, and Heng Xiao. “Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids”. In: *Computer Methods in Applied Mechanics and Engineering* 388 (2022), p. 114211. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2021.114211.

[31] Akihiko Nakayama and Satoshi Yokojima. “Modeling Free-Surface Fluctuation Effects for Calculation of Turbulent Open-Channel Flows”. In: *Environ. Fluid Mech. (Dordr.)* 3.1 (2003), pp. 1–21.

[32] Jinlong Wu, Jian-Xun Wang, Heng Xiao, and Julia Ling. “A Priori Assessment of Prediction Confidence for Data-Driven Turbulence Modeling”. In: *Flow, Turbulence and Combustion* 99 (July 2017). DOI: 10.1007/s10494-017-9807-0.

[33] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[34] Hui Zou and Trevor Hastie. “Regularization and variable selection via the elastic net”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2 (2005), pp. 301–320.

[35] Peng Zheng, Travis Askham, Steven L. Brunton, J. Nathan Kutz, and Aleksandr Y. Aravkin. "A unified framework for sparse relaxed regularized regression: SR3". In: *IEEE Access* 7 (2019), pp. 1404–1423.

[36] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[37] Brian M. de Silva, Kathleen Champion, Markus Quade, Jean-Christophe Loiseau, J. Nathan Kutz, and Steven L. Brunton. *PySINDy: A Python package for the Sparse Identification of Nonlinear Dynamics from Data*. 2020. arXiv: 2004.08424 [math.DS].

[38] Elske van Leeuwen. "Data-Driven Turbulence Modeling, Discovering Turbulence Models using Sparse Symbolic Regression". MA thesis. 2022.

[39] R.J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. ISBN: 9781139434188.

[40] G.D. Weymouth and Dick K.-P. Yue. "Conservative Volume-of-Fluid method for free-surface simulations on Cartesian-grids". In: *Journal of Computational Physics* 229.8 (2010), pp. 2853–2865. ISSN: 0021-9991.

[41] E. Aulisa, S. Manservisi, R. Scardovelli, and S. Zaleski. "Interface reconstruction with least-squares fit and split advection in three-dimensional Cartesian geometry". In: *Journal of Computational Physics* 225.2 (2007), pp. 2301–2319. ISSN: 0021-9991.

[42] Ruben Scardovelli and Stephane Zaleski. "Analytical Relations Connecting Linear Interfaces and Volume Fractions in Rectangular Grids". In: *Journal of Computational Physics* 164.1 (2000), pp. 228–237. ISSN: 0021-9991.

[43] J. López, C. Zanzi, P. Gómez, R. Zamora, F. Faura, and J. Hernández. "An improved height function technique for computing interface curvature from volume fractions". In: *Computer Methods in Applied Mechanics and Engineering* 198.33 (2009), pp. 2555–2564. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2009.03.007.

[44] S. Hysing, S. Turek, D. Kuzmin, N. Parolini, E. Burman, S. Ganesan, and L. Tobiska. "Quantitative benchmark computations of two-dimensional bubble dynamics". In: *International Journal for Numerical Methods in Fluids* 60.11 (2009), pp. 1259–1288. DOI: https://doi.org/10.1002/fld.1934.

[45] Christopher Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles*. Reading, UK: CFD Direct Ltd, 2022.

[46] J.H. Ferziger, M. Perić, and R.L. Street. *Computational Methods for Fluid Dynamics*. Springer, 2020. ISBN: 9783319996929. URL: https://books.google.nl/books?id=Fy2izQEACAAJ.

[47] Yu Sheng. "Towards Data-Driven Modelling of Turbulent Two-Phase Flows with Large Interfaces". MA thesis. 2023.

[48] Andrea Beck and Marius Kurz. "A perspective on machine learning methods in turbulence modeling". In: *GAMM-Mitteilungen* 44.1 (2021), e202100002. DOI: https://doi.org/10.1002/gamm.202100002.

[49] Cornelia Grabe, Florian Jäckel, Parv Khurana, and Richard Dwight. "Data-driven augmentation of a RANS turbulence model for transonic flow prediction". In: *International Journal of Numerical Methods for Heat Fluid Flow* 33 (Apr. 2023). DOI: 10.1108/HFF-08-2022-0488.

[50] Florian Jäckel. "A Closed-form Correction for the Spalart-Allmaras Turbulence model for Separated Flows". In: *AIAA SCITECH 2022 Forum*. DOI: 10.2514/6.2022-0462. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2022-0462.

[51] Romit Maulik, Himanshu Sharma, Saumil Patel, Bethany Lusch, and Elise Jennings. *Deploying deep learning in OpenFOAM with TensorFlow*. Dec. 2020.

[52] Julia Steiner, Richard P. Dwight, and Axelle Viré. "Data-driven RANS closures for wind turbine wakes under neutral conditions". In: *Computers Fluids* 233 (2022), p. 105213. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2021.105213.

[53] Jinlong Wu, Heng Xiao, Rui Sun, and Qiqi Wang. "RANS Equations with Explicit Data-Driven Reynolds Stress Closure Can Be Ill-Conditioned". In: (Mar. 2019). DOI: 10.1017/jfm.2019.205.

[54] Jin-Long Wu, Heng Xiao, and Eric Paterson. "Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework". In: *Phys. Rev. Fluids* 3 (7 July 2018), p. 074602. DOI: 10.1103/PhysRevFluids.3.074602.

[55] Jin-Long Wu, Rui Sun, Sylvain Laizet, and Heng Xiao. "Representation of stress tensor perturbations with application in machine-learning-assisted turbulence modeling". In: *Computer Methods in Applied Mechanics and Engineering* 346 (2019), pp. 707–726. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2018.09.010.

[56] Heng Xiao, Jin-Long Wu, Sylvain Laizet, and Lian Duan. "Flows over periodic hills of parameterized geometries: A dataset for data-driven turbulence modeling from direct simulations". In: *Computers Fluids* 200 (2020), p. 104431. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2020.104431.

[57] Z. Wang and Weiwei Zhang. *A unified method of data assimilation and turbulence modeling for separated flows at high Reynolds numbers*. Nov. 2022.

[58] Xin-Lei Zhang, Heng Xiao, Xiaodong Luo, and Guowei He. "Combining direct and indirect sparse data for learning generalizable turbulence models". In: *Journal of Computational Physics* 489 (2023), p. 112272. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2023.112272.

A

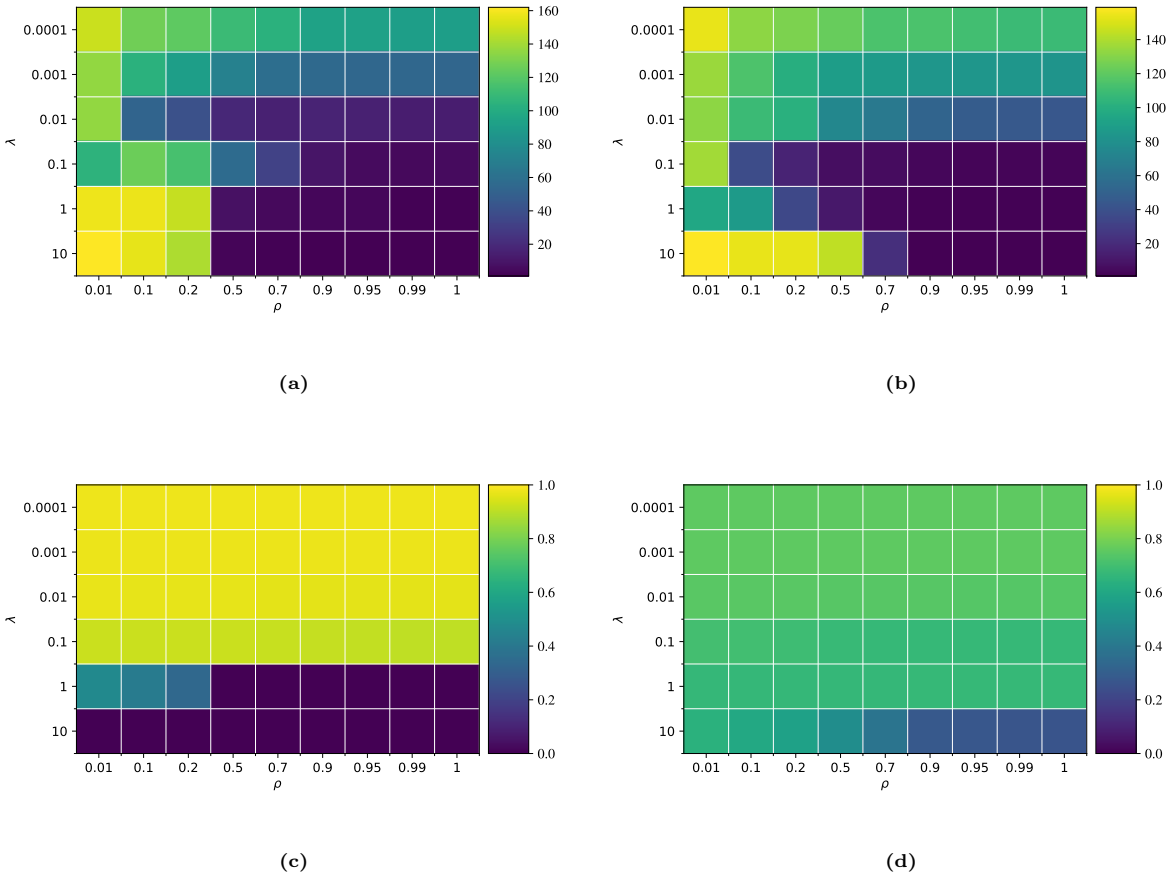
Additional results

This appendix presents results that were omitted from chapter 6 and serves as a continuation of sections 6.3.1 and 6.3.2.

A.1. Training

This section includes the results omitted from section 6.3.1.

Elastic net



(a) **(b)**

(c) **(d)**

Figure A.1: Number of terms (Δ_k (a) and Δ_ω (b)) and R^2 validation score (Δ_k (c) and Δ_ω (d)) of the discovered models using the elastic net regressor depending on the value of the hyperparameters λ and ρ.

Figure A.1 shows the R^2 validation scores and the number of terms of the discovered models, in this case, using the Elastic net as a regression method. For this regressor, a larger value of λ implies stronger regularization, while a larger value of ρ means more l^1 norm regularization compared to l^2 regularization.

We can see that for a given value of λ as ρ gets larger, the number of non-zero coefficients reduces. We can also see that in general, for a given value of ρ a larger value of λ leads to less term. However, this is not always the case. For low values of ρ (more l^2 norm regularization), increasing the value of λ leads to more terms. This might be due to the l^1 regularization not having a significant impact for low values of ρ.

The R^2 validation score results are similar to the Lasso results. The more regularization, the lower the score. Moreover, the value of ρ also seems to have a large impact on the validation score. The more relative importance the l^1 norm has, the lower the score. The causes of these results were already discussed with the LASSO regressor, but the results of this regressor show that there is some correlation between very sparse expressions and low validation scores. This may suggest that it may not be possible to obtain expressions that are general enough with only a few terms, at least with the chosen candidate library.

STLSQ

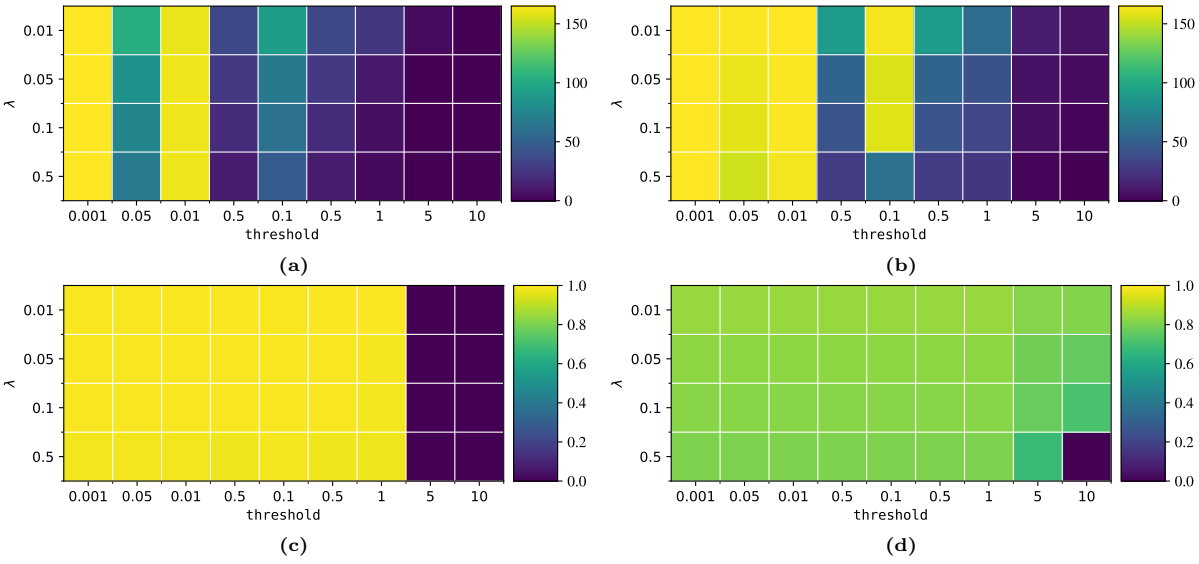


Figure A.2: Number of terms (Δ_k (a) and Δ_ω (b)) and R^2 validation score (Δ_k (c) and Δ_ω (d)) of the discovered models using the STLSQ regressor depending on the value of the hyperparameters λ and ρ.

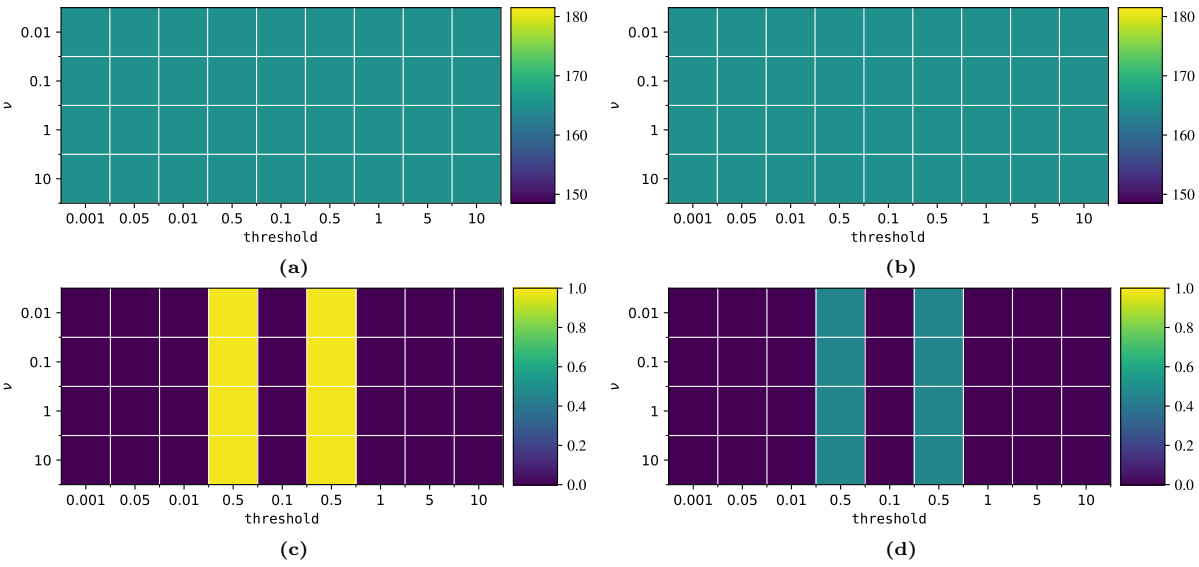


Figure A.3: Number of terms (Δ_k (a) and Δ_ω (b)) and R^2 validation score (Δ_k (c) and Δ_ω (d)) of the discovered models using the SR3 regressor depending on the value of the hyperparameters λ and ρ.

Figure A.2 shows the R^2 validation scores and the number of terms of the discovered models using the STLSQ regressor. The results are similar to the previous cases. In general, the larger the `threshold` value, the sparser the model is. Although this is not always the case, interestingly, the explanation of why a larger `threshold` leads to more non-zero terms is not clear. For this regressor, the value of the `threshold` has a larger impact on sparsity than the value of the regularization parameter λ. Again a similar effect can be seen for the validation scores.

SR3

Figure A.2 shows the R^2 validation scores and the number of terms of the discovered models using the SR3 regressor. The results are very similar to those of the STLSQ regressor, with one main difference: the effect of the regularization is not significant enough to obtain very sparse expressions. The models with the least terms still have more than 100 terms, which means no adequate models are found with this regularization.

SNN

Figure A.4 shows the R^2 validation scores of the trained separated NNs models for different types of activation functions and number of neurons per hidden layer. As happened with the sparse symbolic regressions the validation scores are higher for the Δ_k corrections than for the Δ_ω corrections. As for the effect of the hyperparameters, the results are analogous to those of the joint NNs, and the same conclusions can be drawn.

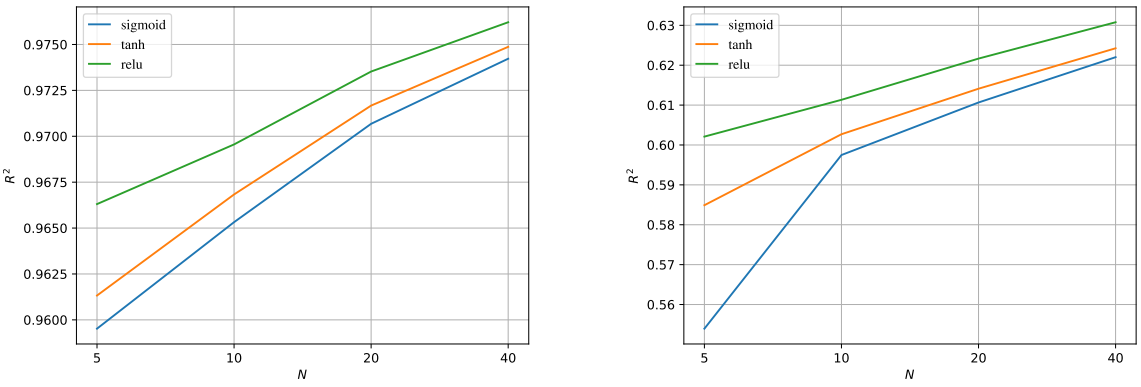


Figure A.4: Validation R^2 score for Δ_k (a) and Δ_ω (b) using SNNs.

A.2. Testing

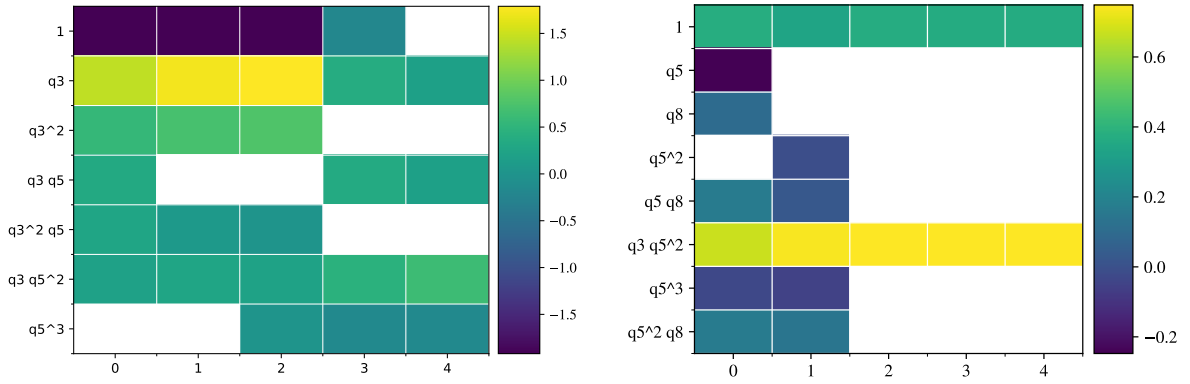
This section includes the results omitted from section 6.3.2.

Elastic net

Figure A.5 shows the Δ_k (left panel) and Δ_ω (right panel) models with the best validation scores discovered using the elastic net regression. Compared to the models discovered with LASSO, elastic net models have more terms and in the case of Δ_ω the terms depend on more features.

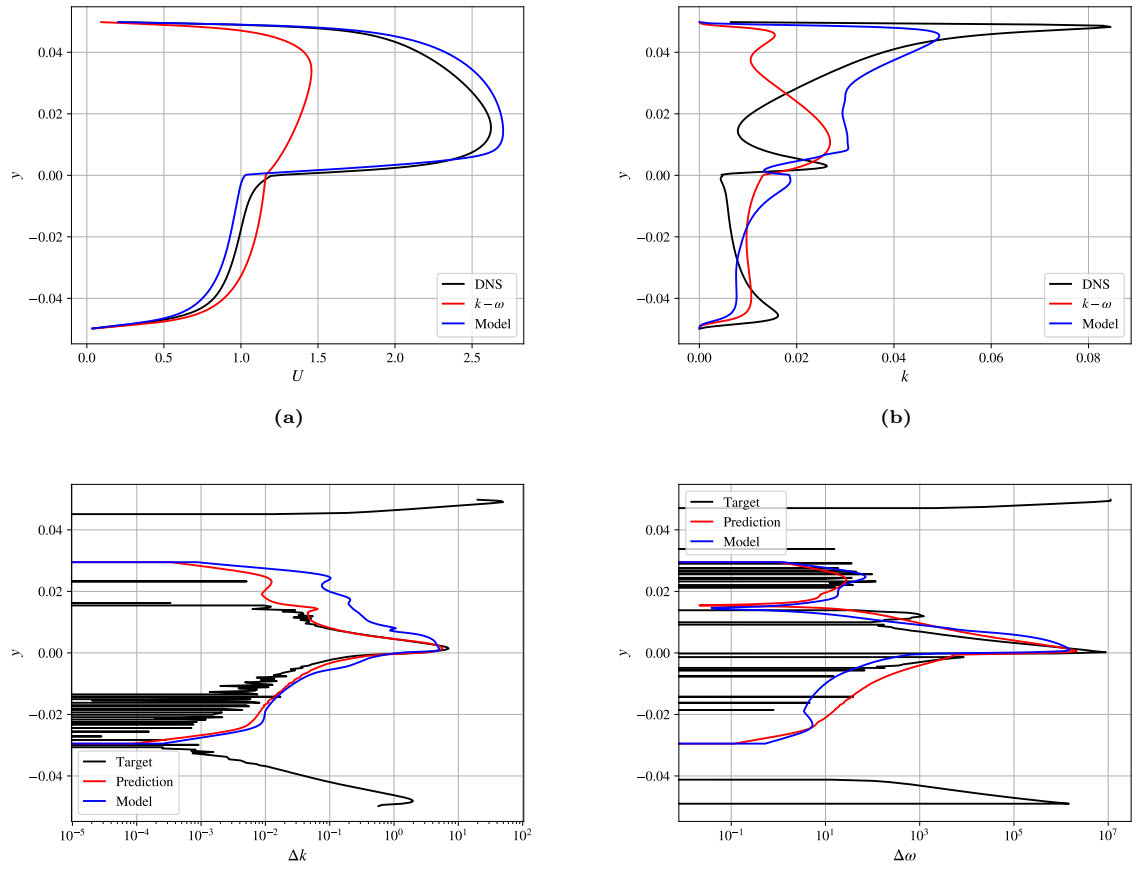
Figure A.7 shows the average MSE error of the velocity and turbulent kinetic energy for the propagated models in the training cases and in the testing cases. The results are very similar to those of the LASSO models. However, with the elastic net regressor we were able to find more models which improved the accuracy of the baseline model on average.

Lastly, figure A.6 shows the results of the propagation of the best model discovered using elastic net for case 8, which was not used for training. The results are analogous to those of the best LASSO model.



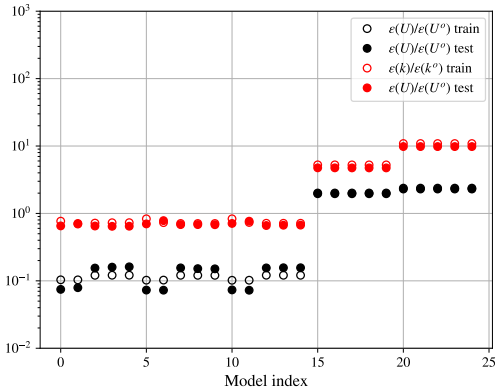
(a) **(b)**

Figure A.5: Non-zero coefficients of the propagated symbolic expressions for Δ_k (a) and Δ_ω (b) of the discovered models using the elastic net regressor. The x-axis is the model index (a lower index indicates a higher validation score), and the y-axis shows the terms associated with the non-zero coefficients.



(a) **(b)** **(c)** **(d)**

Figure A.6: Velocity (a), turbulent kinetic energy (b), Δ_k (c) and Δ_ω (d) for case 8. Legend panels (a) and (b): *DNS: DNS results.* $k-\omega$*: baseline model. Model: elastic net regressed model.* Legend panels (c) and (d): *Target: Regression targets. Prediction: correction prediction in the training phase. Model: prediction field in the converged state of the simplified model.* (Following figure A.5, model 2 for Δ_k and model 0 for Δ_ω)

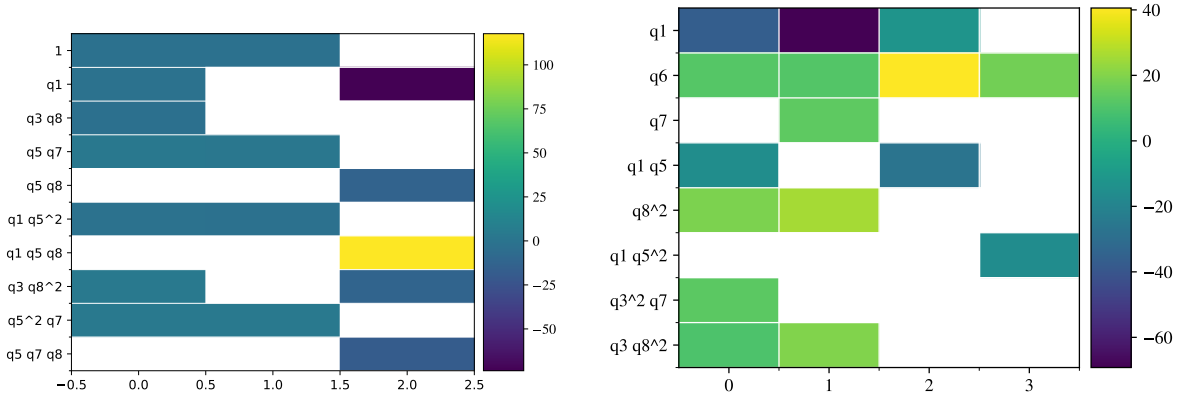


(a)

Figure A.7: Average MSE error of the velocity and the turbulent kinetic energy of the propagated elastic net models. The model index is $i = i_\omega \cdot n_\omega + i_k$, where the i_k and i_ω are the model indices for Δ_k and Δ_ω given in figure A.5 and n_ω is the number of propagated Δ_ω models.

STLSQ

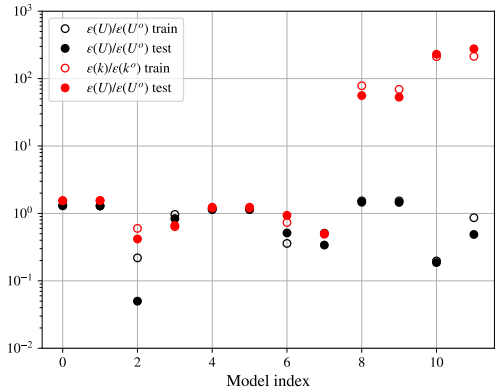
Figure A.8 shows the Δ_k (left panel) and Δ_ω (right panel) models with the best validation scores discovered using the STLSQ regression. Compared to the previous cases the expression obtained with this regressor depends on a larger variety of input features. They also have coefficients that are larger by one order of magnitude.



(a) **(b)**

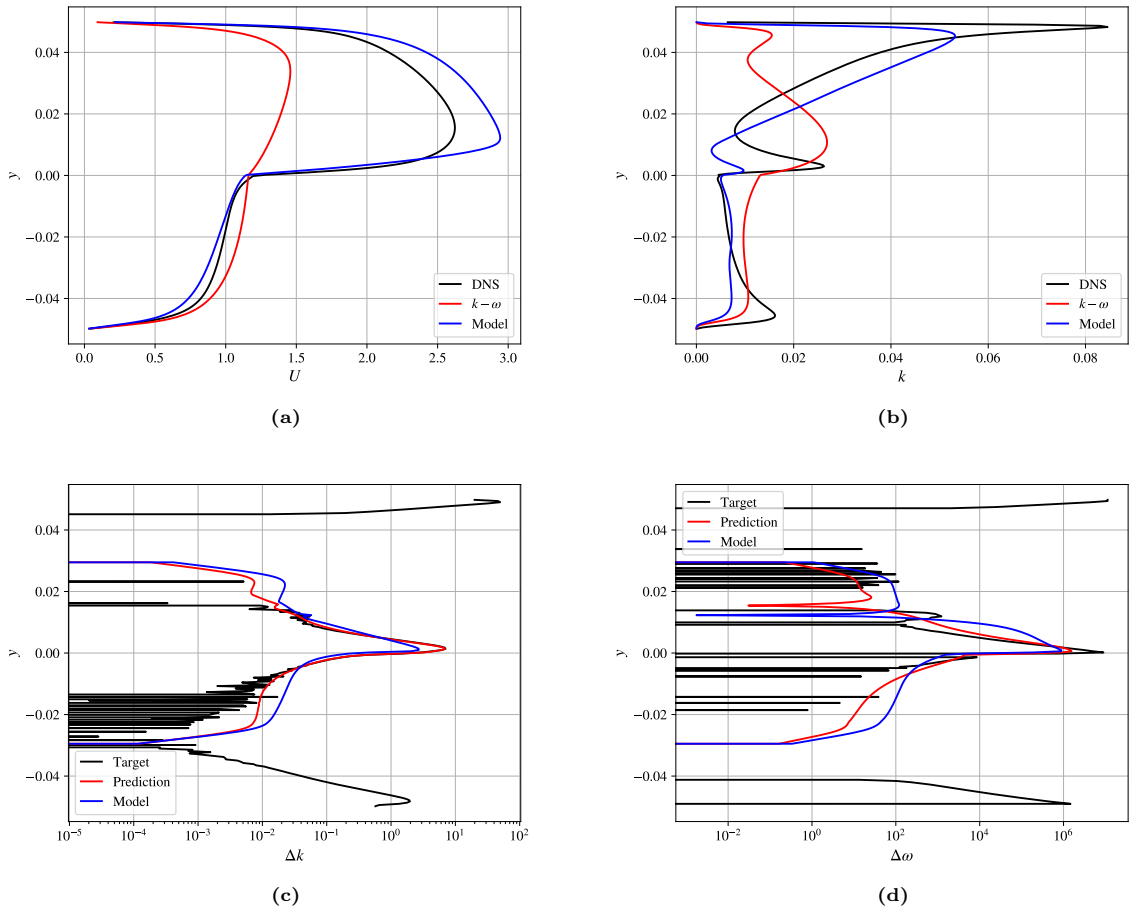
Figure A.8: Non-zero coefficients of the propagated symbolic expressions for Δ_k (a) and Δ_ω (b) of the discovered models using the STLSQ regressor. The x-axis is the model index (a lower index indicates a higher validation score), and the y-axis shows the terms associated with the non-zero coefficients.

Looking at the average MSE for the velocity and turbulent kinetic energy shown in figure A.10, we see that this models have a larger error in general than the models discovered with the previous methods. However, there is one model for which the average test MSE error of the velocity is the lowest of all models seen so far.



(a)

Figure A.9: Average MSE error of the velocity and the turbulent kinetic energy of the propagated STLSQ models. The model index is $i = i_\omega \cdot n_\omega + i_k$, where the i_k and i_ω are the model indices for Δ_k and Δ_ω given in figure A.8 and n_ω is the number of propagated Δ_ω models.

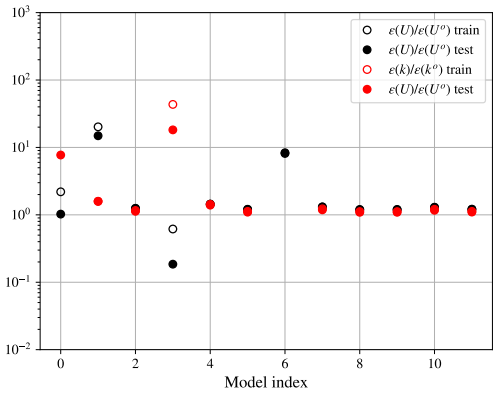


(a) **(b)** **(c)** **(d)**

Figure A.10: Velocity (a), turbulent kinetic energy (b), Δ_k (c) and Δ_ω (d) for case 8. Legend panels (a) and (b): *DNS: DNS results. $k-\omega$: baseline model. Model: STLSQ regressed model.* Legend panels (c) and (d): *Target: Regression targets. Prediction: correction prediction in the training phase. Model: prediction field in the converged state of the simplified model.* (Following figure A.8, model 0 for Δ_k and model 2 for Δ_ω)

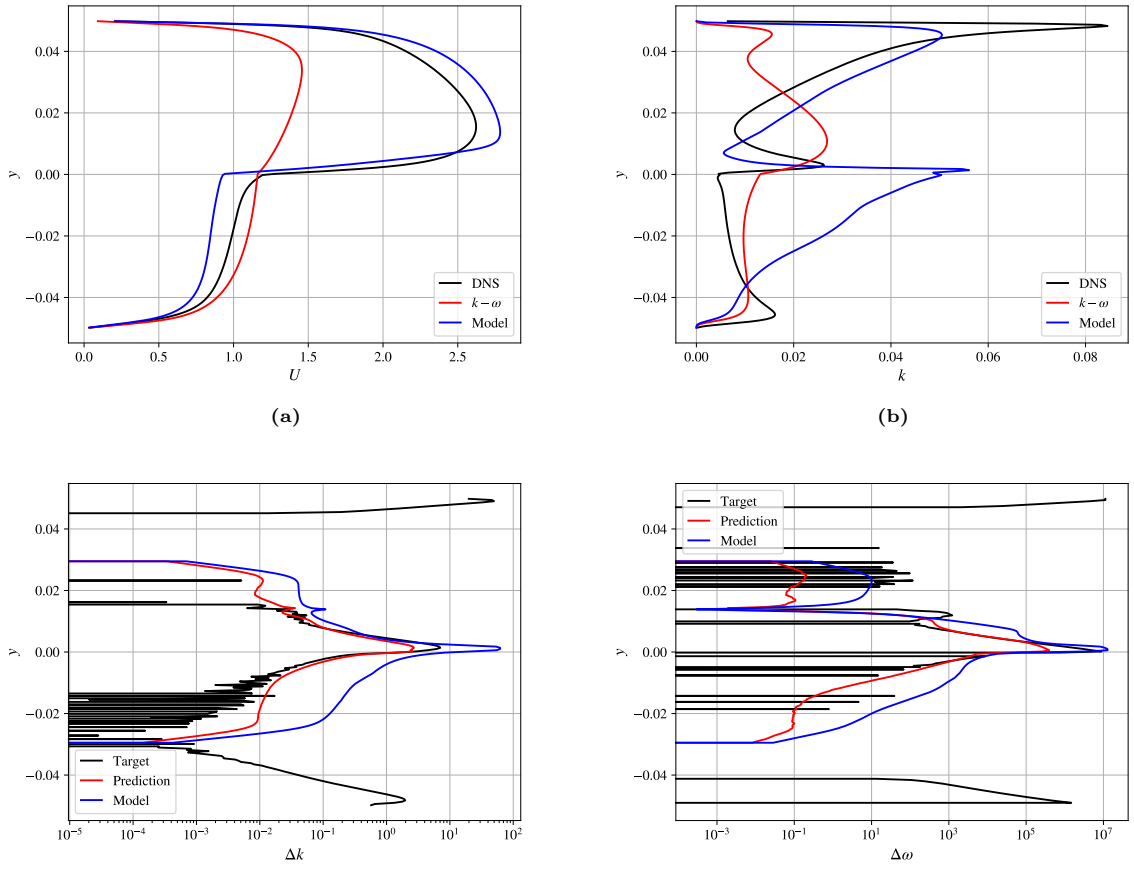
Figure A.10 shows the results of the propagation of the best model discovered using STLSQ for

case 8, which was not used for training. Compared to the results of the previous methods, the velocity profile is slightly overestimated, while the turbulent kinetic energy in the gas phase is better predicted.



(a)

Figure A.11: Average MSE error of the velocity and the turbulent kinetic energy of the propagated JNN models in the training and testing cases.



(a) **(b)**

(c) **(d)**

Figure A.12: Velocity (a), turbulent kinetic energy (b), Δ_k (c) and Δ_ω (d) for case 8. Legend panels (a) and (b): *DNS: DNS results. $k-\omega$: baseline model. Model: JNN regressed model.* Legend panels (c) and (d): *Target: Regression targets. Prediction: correction prediction in the training phase. Model: prediction field in the converged state of the simplified model.*

SR3

All models found with SR3 had more than 100 non-zero terms. Therefore, no adequate model was discovered with this regressor.

JNN

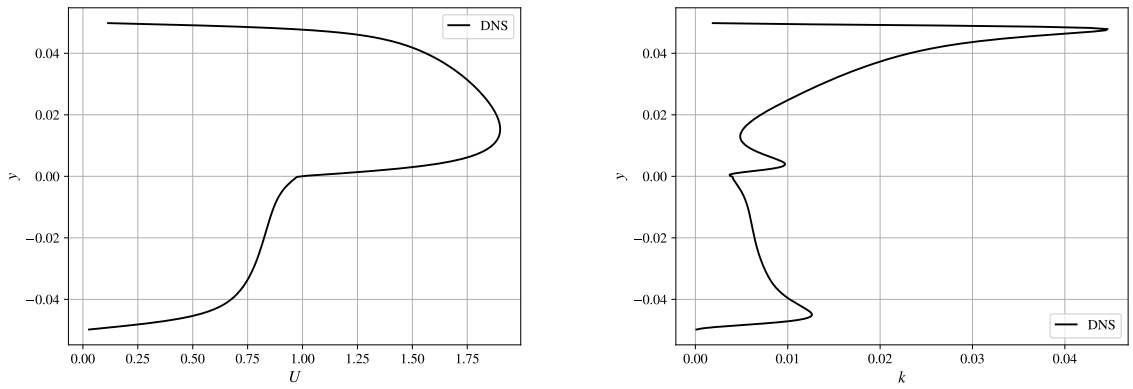
The MSE errors of the JNN models, shown in figure A.11, are very similar to the SNN ones. Most models do not do anything, and the models that are doing something are increasing the MSE of k compared to the baseline model. Only one trained model improves the baseline model errors in the velocity.

The results of propagating the best JNN trained model, shown in figure A.12, is again similar to that of the best SNN model, albeit the corrections are not overpredicted as much as in the SNN case, yielding better velocity and turbulent kinetic energy predictions.

B

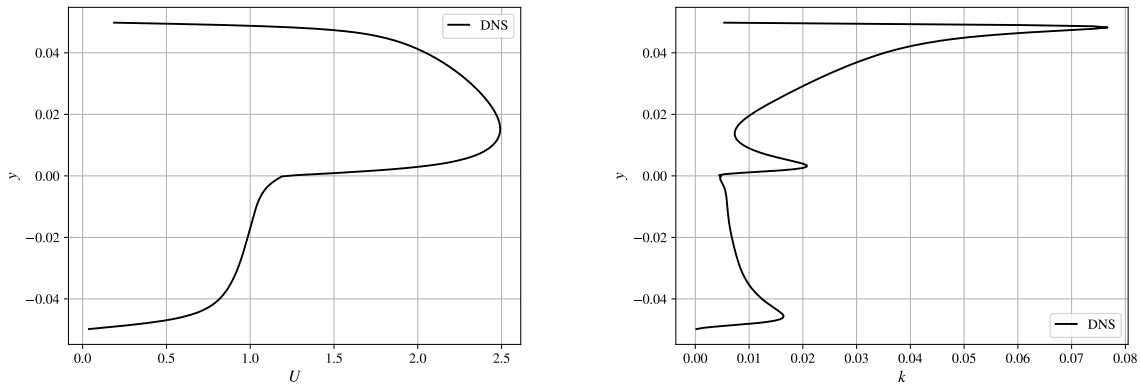
DNS results

The results of the DNS simulations of the different cases are shown in this chapter.



(a) **(b)**

Figure B.1: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 1.



(a) **(b)**

Figure B.2: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 2.

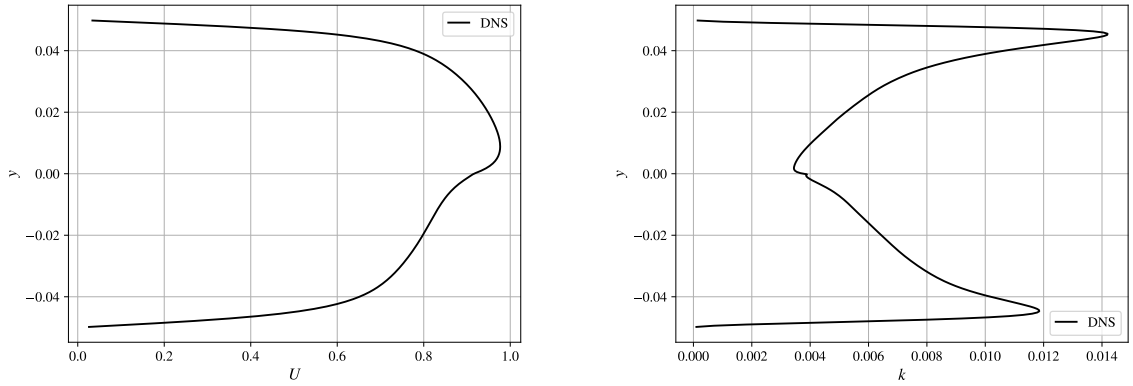


Figure B.3: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 3.

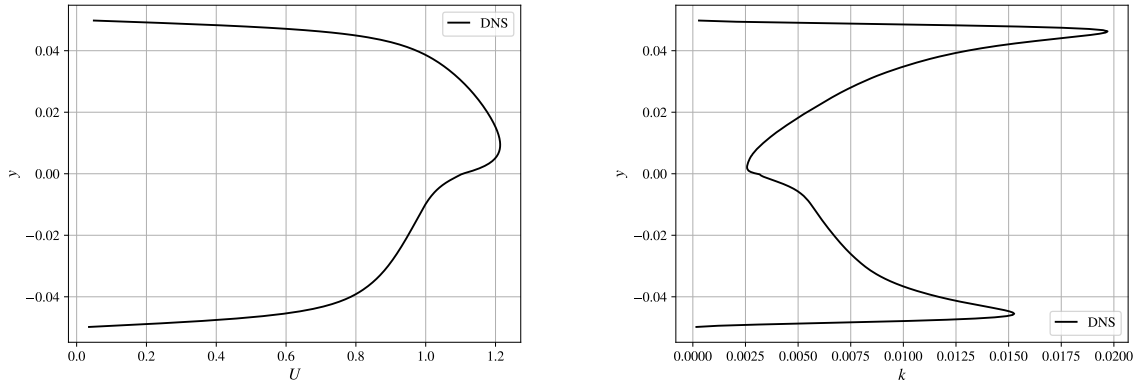


Figure B.4: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 4.

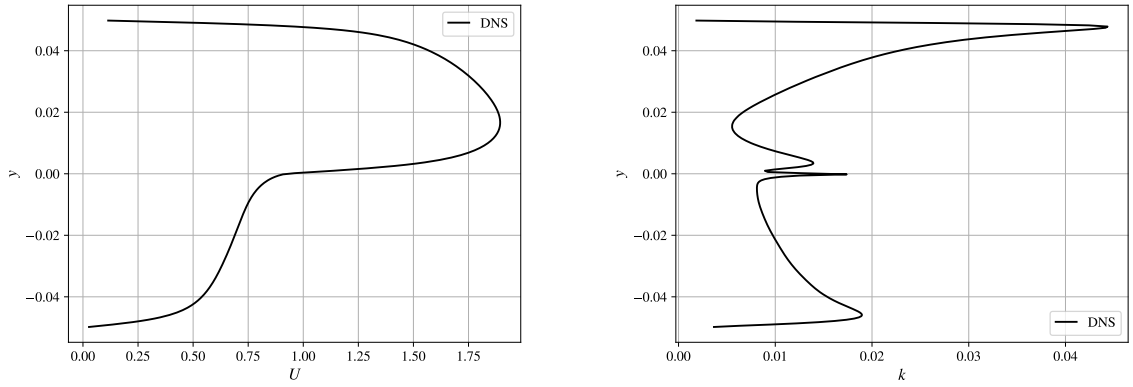
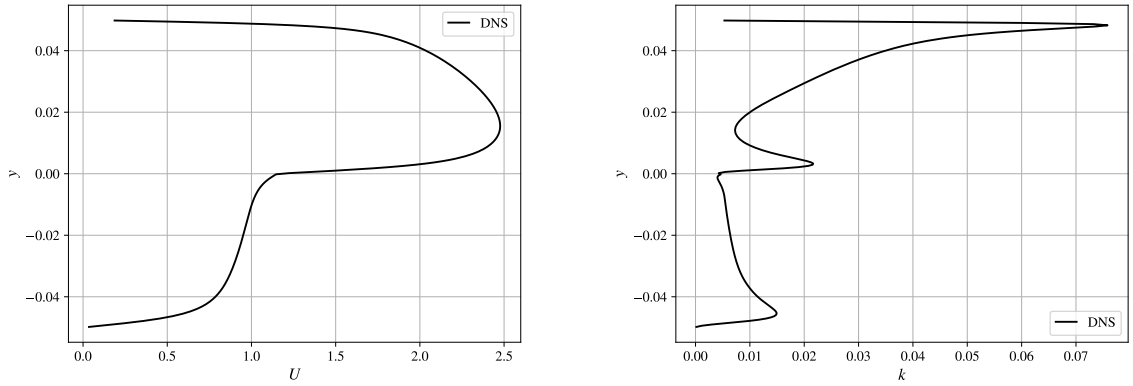
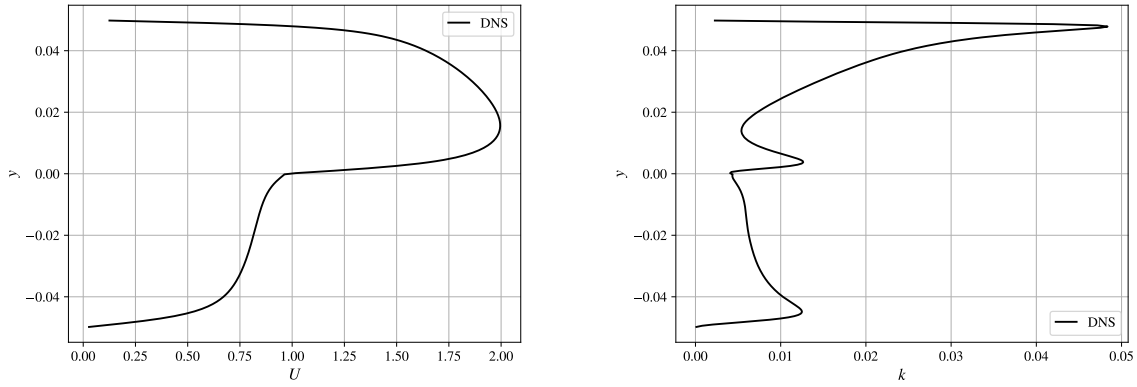


Figure B.5: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 5.



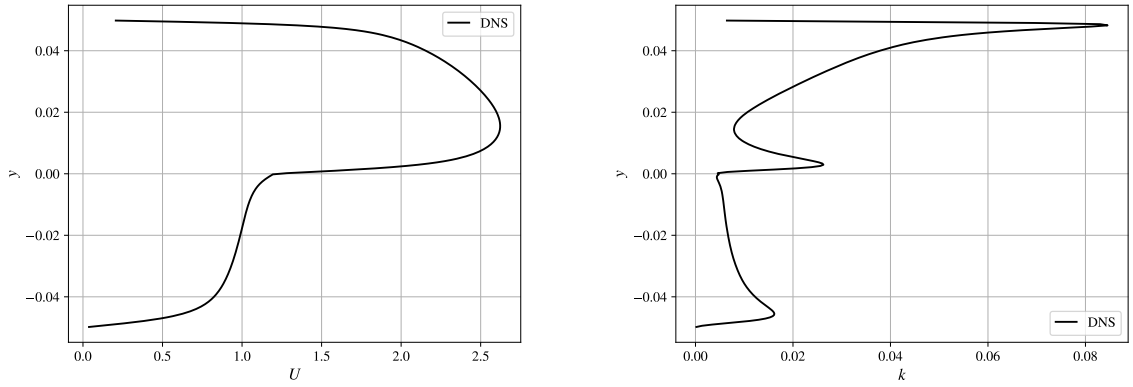
(a) **(b)**

Figure B.6: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 6.



(a) **(b)**

Figure B.7: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 7.



(a) **(b)**

Figure B.8: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 8.

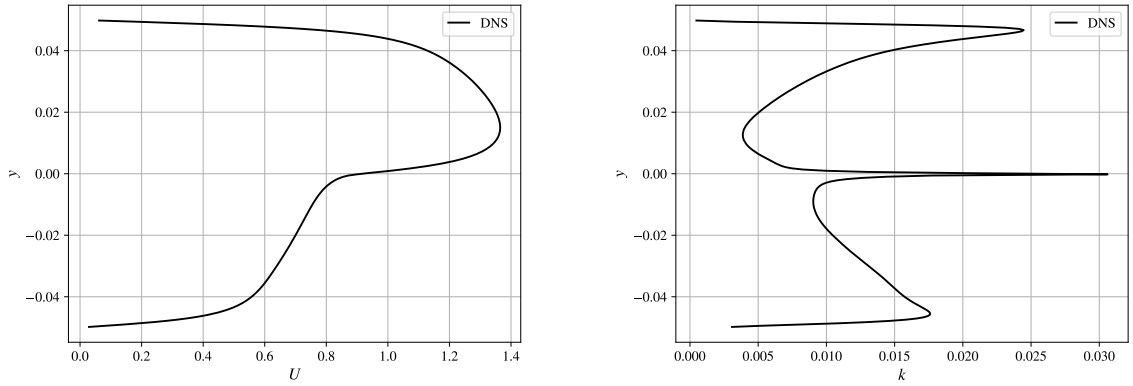


Figure B.9: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 9.

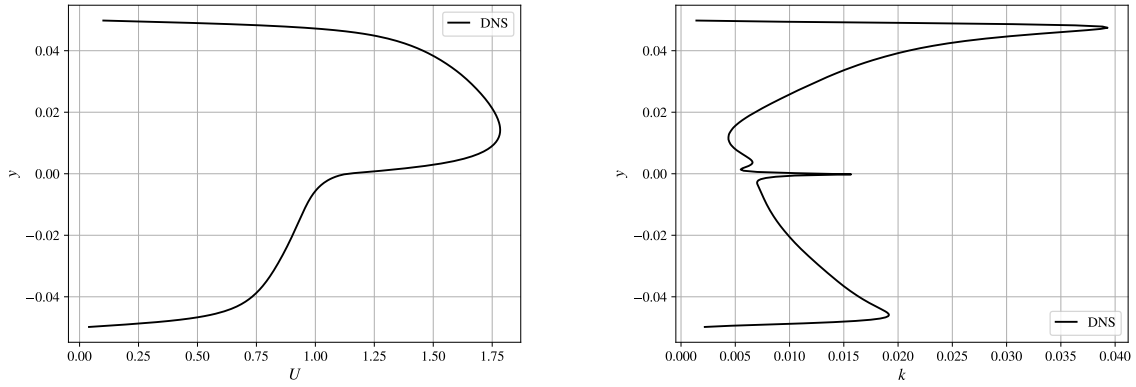


Figure B.10: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 10.

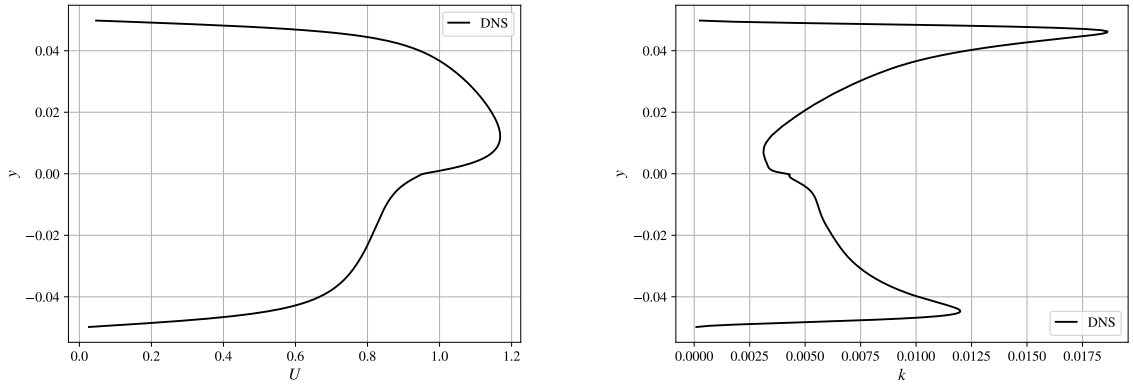


Figure B.11: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 11.

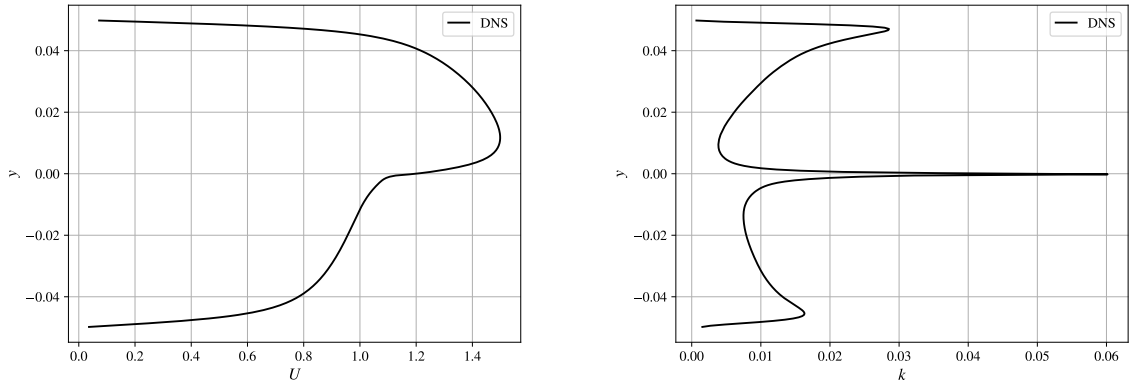


Figure B.12: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 12.

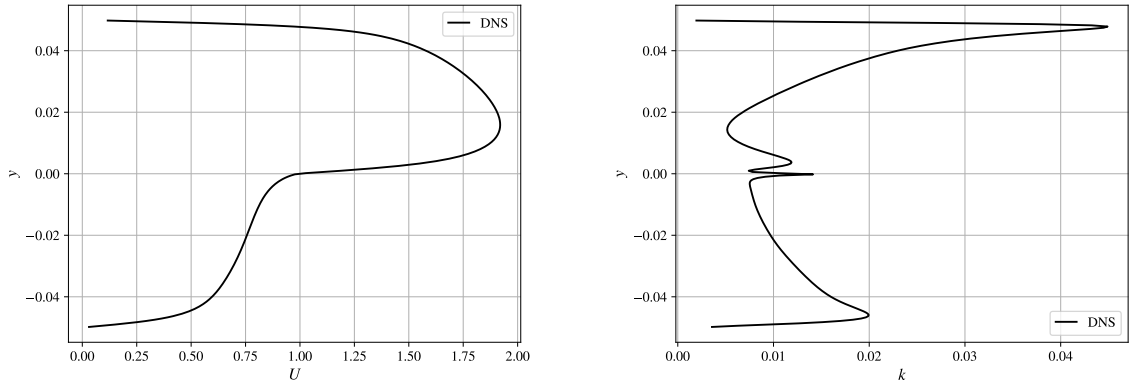


Figure B.13: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 13.

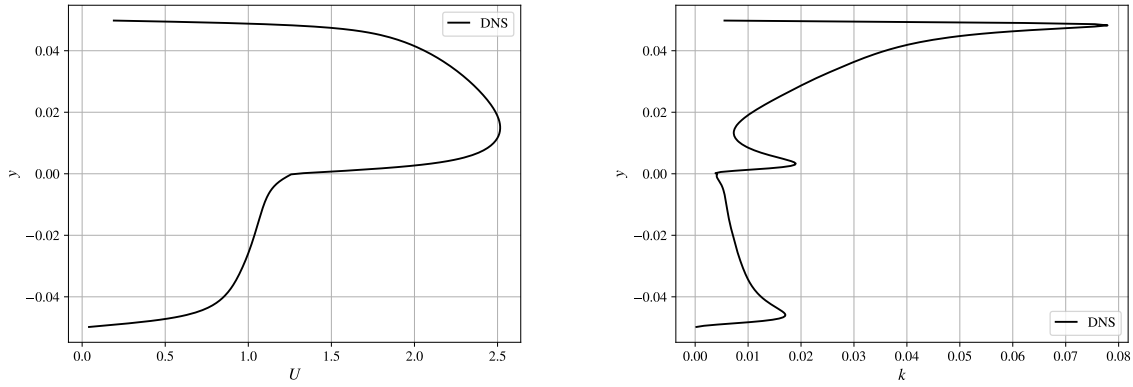
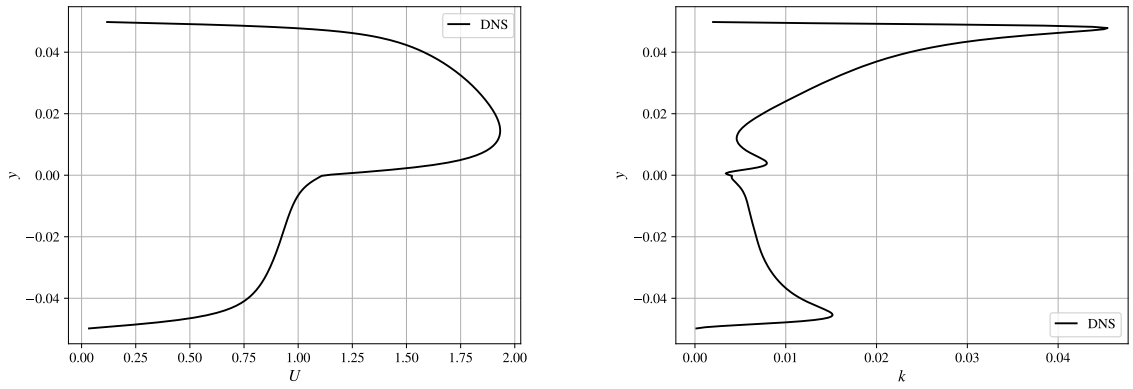
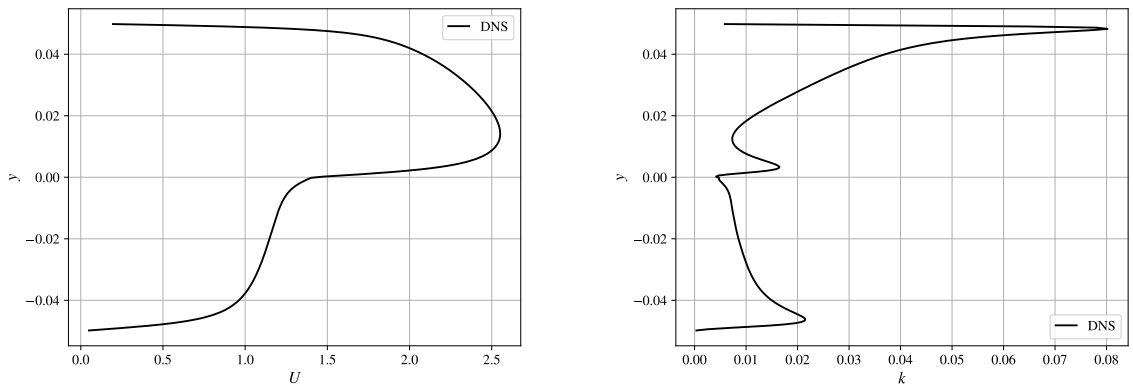


Figure B.14: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 14.



(a) **(b)**

Figure B.15: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 15.



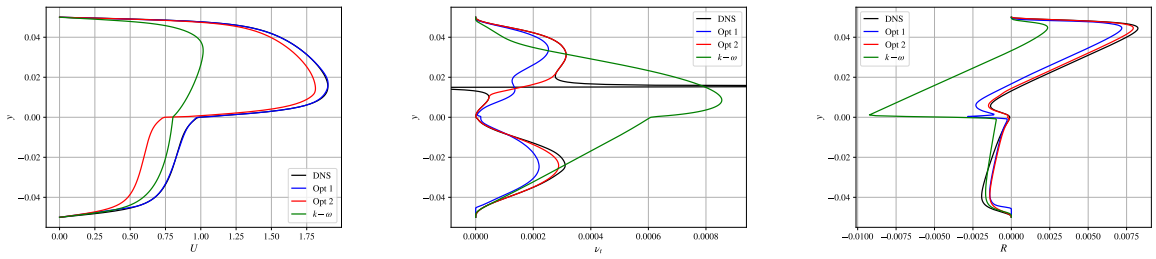
(a) **(b)**

Figure B.16: DNS average streamwise velocity (a) and turbulent kinetic energy (b) field of case 16.

C

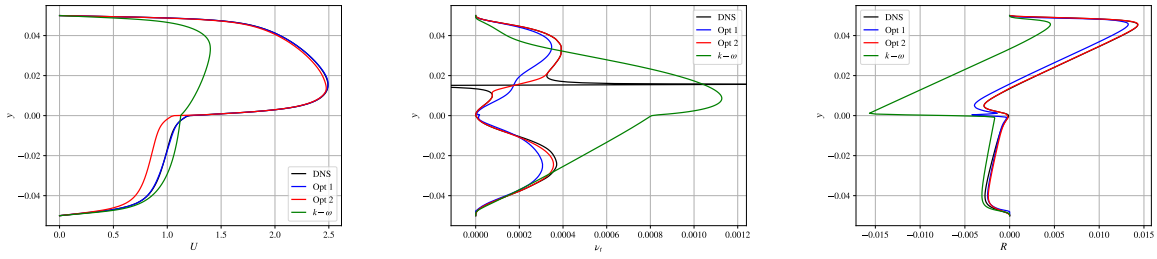
Turbulent viscosity field inversion results

This chapter includes the results from the turbulent viscosity field inversion using both methods proposed in the thesis.



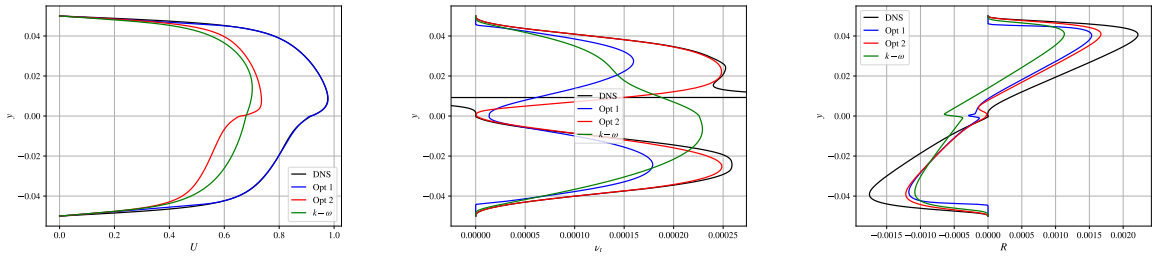
(a) **(b)** **(c)**

Figure C.1: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 1. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



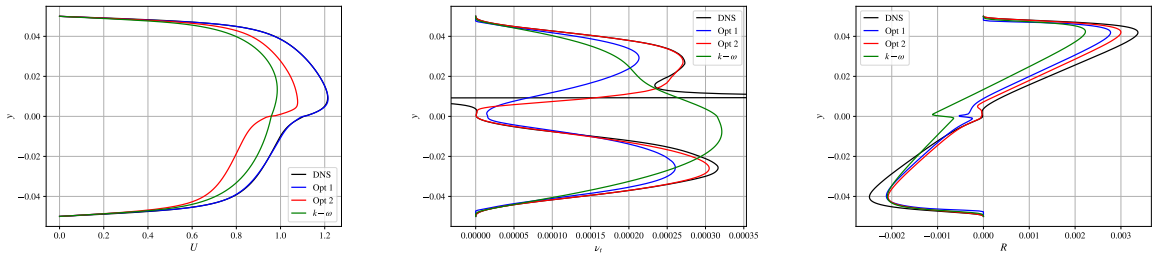
(a) **(b)** **(c)**

Figure C.2: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 2. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



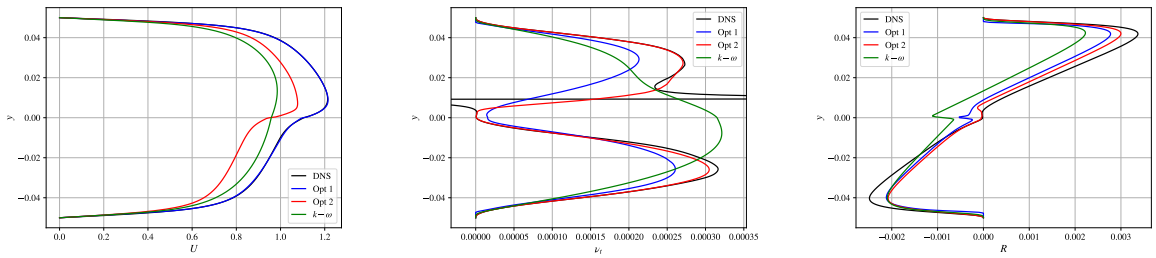
(a) **(b)** **(c)**

Figure C.3: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 3. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



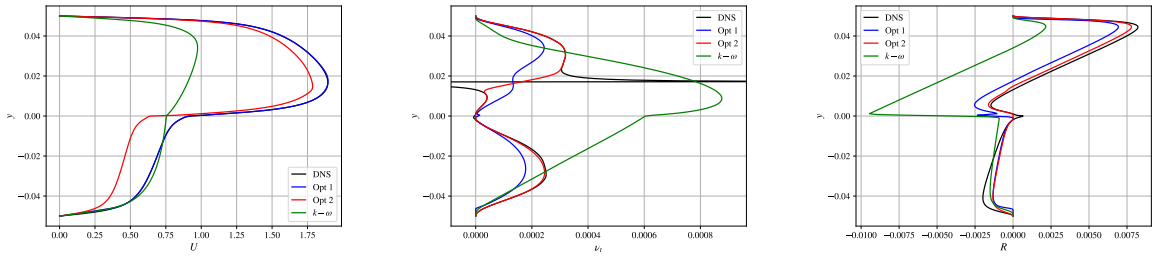
(a) **(b)** **(c)**

Figure C.4: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 4. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



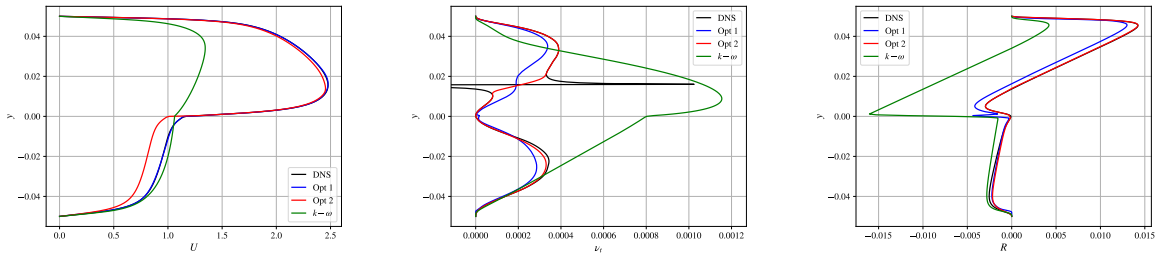
(a) **(b)** **(c)**

Figure C.5: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 4. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



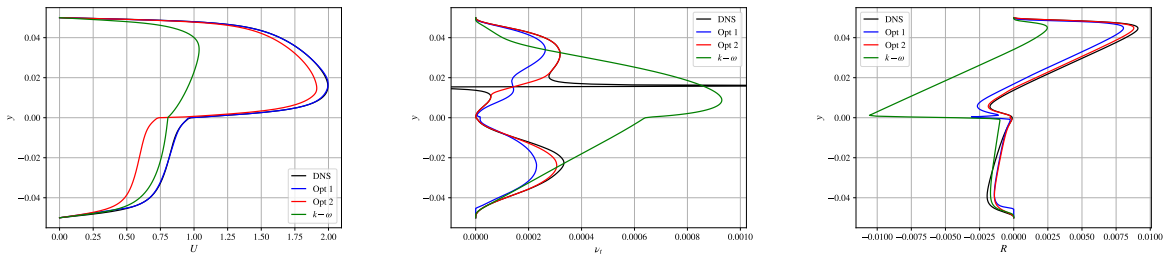
(a) **(b)** **(c)**

Figure C.6: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 5. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



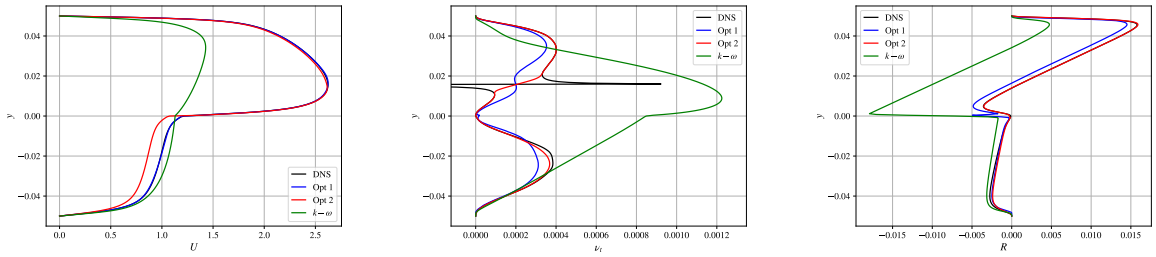
(a) **(b)** **(c)**

Figure C.7: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 6. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



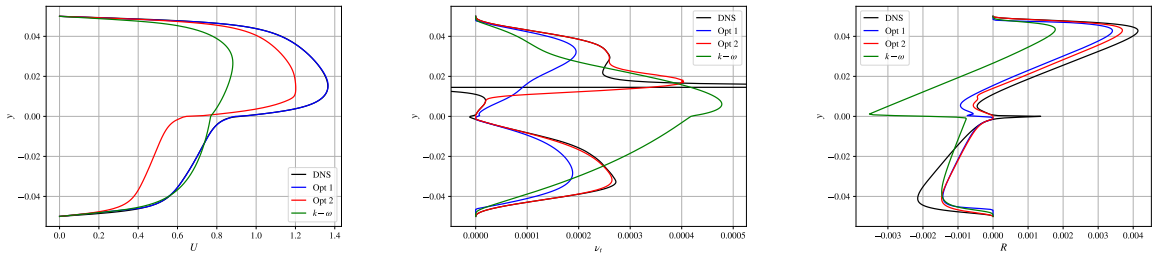
(a) **(b)** **(c)**

Figure C.8: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 7. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



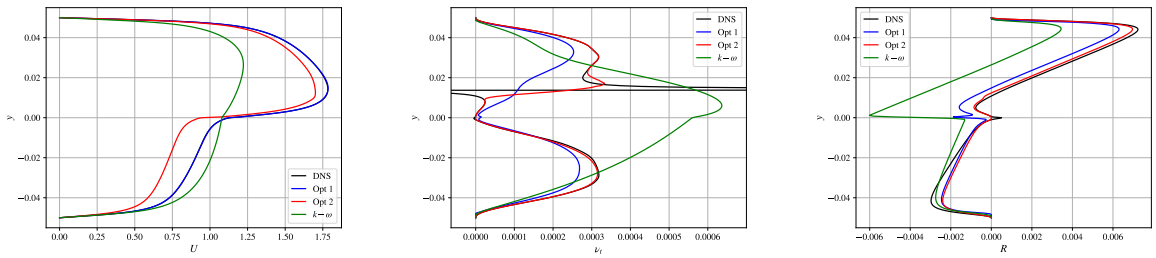
(a) **(b)** **(c)**

Figure C.9: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 8. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



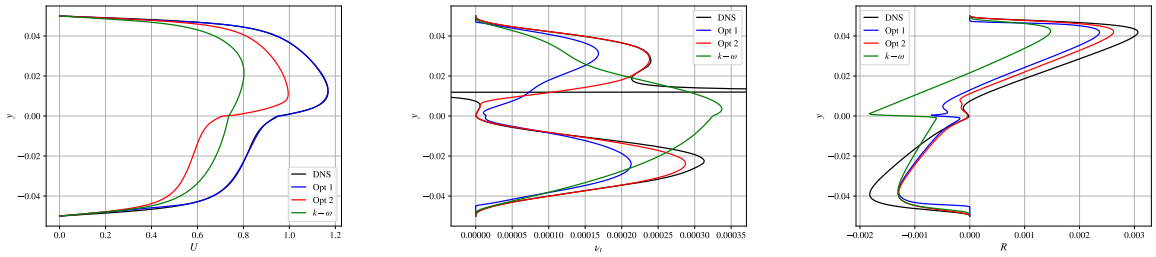
(a) **(b)** **(c)**

Figure C.10: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 9. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



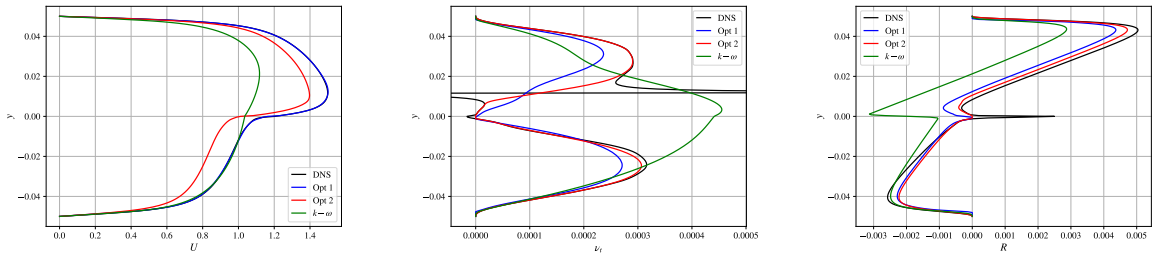
(a) **(b)** **(c)**

Figure C.11: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 10. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



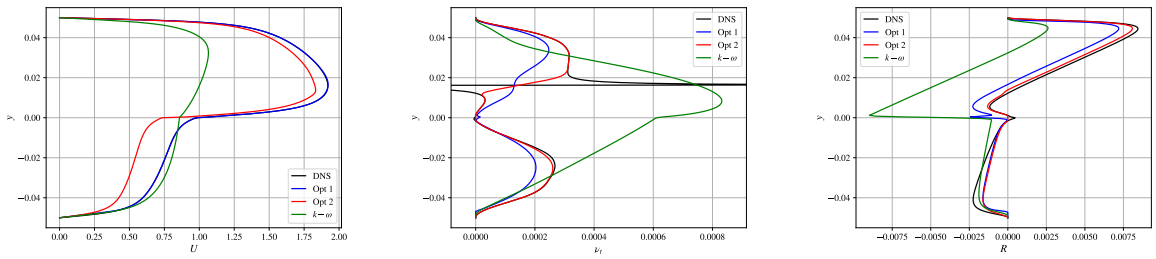
(a) **(b)** **(c)**

Figure C.12: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 11. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



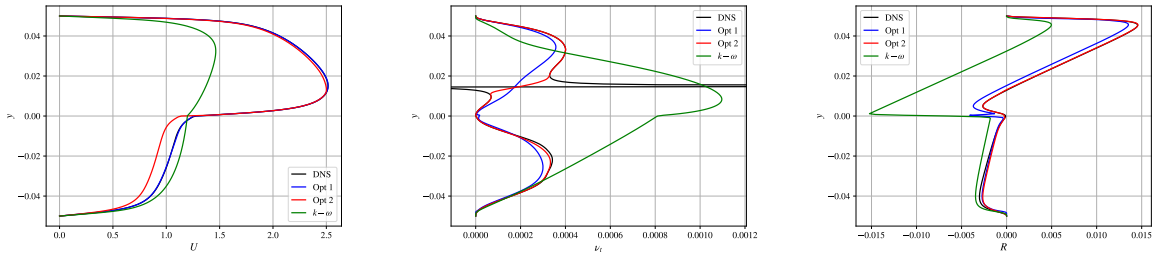
(a) **(b)** **(c)**

Figure C.13: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 12. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



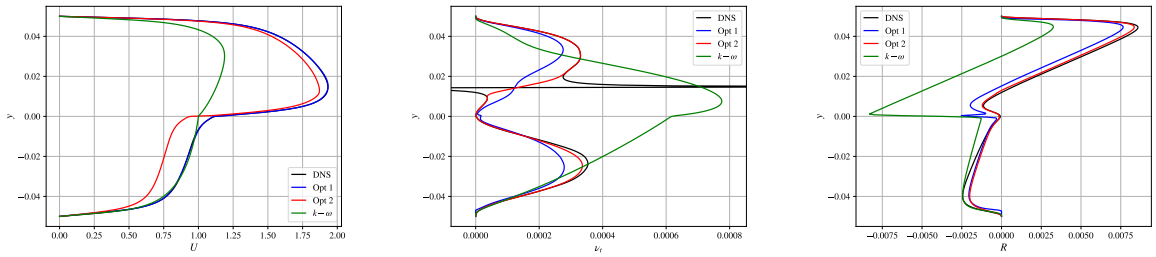
(a) **(b)** **(c)**

Figure C.14: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 13. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



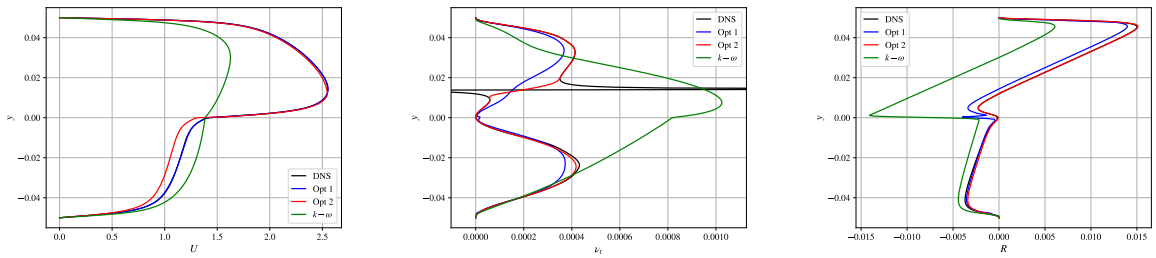
(a) **(b)** **(c)**

Figure C.15: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 14. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



(a) **(b)** **(c)**

Figure C.16: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 15. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*



(a) **(b)** **(c)**

Figure C.17: Turbulent viscosity obtained from the field inversion (b), resulting velocity field (a) and xy component of the Reynolds stress tensor (c) for case 16. Legend: *DNS: Velocity profile obtained from the DNS and turbulent viscosity obtained from formula* (4.4). *$k-\omega$: fields obtained performing a simulation of the baseline $k-\omega$ model. Opt 1: Fields obtained from the momentum equation field inversion. Opt 2: fields obtained from the Reynolds stress optimization process.*