POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Bayesian Optimization of RANS turbulence models using Ensemble Kalman Filter

Tesi di Laurea Magistrale in
Mathematical Engineering - Ingegneria Matematica

Author: **Matteo Eddy Cesaratto**

Student ID: 945343
Advisor: Prof. Lorenzo Valdettaro
Academic Year: 2021-22

Abstract

Navier-Stokes equations are widely exploited to describe fluid flow. In most of the real situations we face turbulent flows, in which particles move in a chaotic, difficult to forecast, way. Several mathematical tools have been proposed to model this motion: the most detailed Direct Numerical Simulations (DNS), which are highly computationally expensive, the Large Eddy Simulation (LES), in which only structures up to a given threshold are resolved, and the Reynolds Averaged equations (RANS), in which the mean flow is modelled. Several types of turbulence models have been proposed, to model the Reynolds stress tensor and guarantee the closure of the system: in this work we will focus on Eddy Viscosity models, based on Boussinesq assumption. Examples are the k-ε model, k-ω model and Spalart–Allmaras model.
These strategies are those researchers most rely on, but a new approach has been proposed to better fit the models results to real data: Field Inversion (FI) strategy allows to perform a data-driven correction of the models, comparing RANS results with high-fidelity data, coming from DNS or real measurements. In this thesis, FI approach is studied and exploited to improve model constants and get more precise results compared to those obtained with standard models. Specifically, Bayesian approach using Ensemble Kalman Filter (EnKF) is used, first to directly infer the turbulent viscosity ν_T without any further assumption on ν_T, and then to try to correct the constant values playing a role in the k-ω model.

RANS simulations have been performed in OpenFOAM using the simpleFoam solver. For EnKF field inversion DAFI library coupled with OpenFOAM has been taken as a base.

Keywords: Navier-Stokes equations, RANS, Eddy Viscosity turbulence models, Bayesian optimisation, Field Inversion, Ensemble Kalman Filter, OpenFOAM, DAFI

Abstract in lingua italiana

Le equazioni di Navier-Stokes sono largamente utilizzate per descrivere il comportamento di fluidi in movimento. Nella maggior parte dei casi reali, ci troviamo di fronte a moti turbolenti, dal comportamento caotico e difficile da prevedere. Diversi strumenti matematici sono stati proposti per provare a descrivere questi tipi di moto: le Direct Numerical Simulations (DNS), molto dettagliate e per questo molto costose dal punto di vista computazionale, le Large Eddy Simulations, che invece modellizzano le strutture sotto una certa scala, e le equazioni mediate alla Reynolds (RANS), in cui la turbolenza viene modellizzata attraverso un modello di chiusura del tensore di Reynolds. Svariate idee sono state presentate per modellizzarlo: in questo lavoro, in particolare, concentreremo la nostra attenzione sui modelli a viscosità turbolenta, basati sull'ipotesi di Boussinesq. Esempi sono i modelli k-ε, k-ω e Spalart–Allmaras.
Al momento, questo è l'approccio più diffuso: tuttavia, l'adattamento di tecniche di Inversione di Campo (FI) al caso della fluidodinamica permette di ottenere risultati che meglio approssimano i dati reali. In questa tesi, viene analizzato quest'approccio, in particolare servendosi dell'Ensemble Kalman Filter (EnKF) all'interno del contesto dell'ottimizzazione Bayesiana, sfruttando dati high-fidelity provenienti da simulazioni DNS. In un primo momento si è cercato di stimare direttamente i valori della viscosità turbolenta ν_T senza ulteriori ipotesi sulla sua espressione, poi si è cercato di correggere alcune delle costanti che sono coinvolte nel modello k-ω.

Le simulazioni RANS sono state implementate in OpenFOAM, facendo affidamento sul solutore simpleFoam. Per l'algoritmo di FI per mezzo di EnKF è stata sfruttata la libreria DAFI.

Parole chiave: equazioni di Navier-Stokes, RANS, modelli di turbolenza, Inversione di Campo, ottimizzazione Bayesiana, Ensemble Kalman Filter, OpenFOAM, DAFI

Contents

Introduction

General context

Fluid behaviour is a crucial aspect for industrial and engineering applications. The Navier-Stokes equations are a well known model to describe fluid flow, obtained by coupling mass conservation principle and momentum balance for a fluid. From their general formulation, the case of Newtonian, incompressible fluids can be retrieved, and through proper computation, a dimensionless version can be written, so as to highlight the parameter of interest Re (Reynolds number).
The value of Re characterizes the nature of the flow, whether laminar or turbulent: in the former case, for low Re, the fluid particles move on parallel, well ordered streamlines, while in the latter, for high Re (indicatively Re $>$ 2000), motion is chaotic and difficult to describe. In this case, RANS equations are obtained by decomposing each field in a mean and fluctuating component and then averaging through ensemble average. The resulting system needs to be closed: a plenty of models have been proposed. Many of them are Eddy viscosity models, based on the Boussinesq assumption ([8]) of similarity between the Reynolds stress tensor τ^R and the viscous stress tensor τ for Newtonian fluids. In this work, k-ε, k-ω, k-ω SST models are considered ([12],[22],[16]).
These models are, then, discretized and numerically solved: to this purpose, the open source software OpenFOAM is used, in particular the SIMPLE algorithm is exploited [4]. In this thesis, we will refer to the numerical solution of the Navier-Stokes equations coupled with a turbulence model as the 'forward solution', to distinguish it from the following Field Inversion step.

In order to better fit the numerical results with the experimental or DNS ones (high-fidelity data), a data-driven approach is considered [20]: the Field Inversion technique allows to correct the original turbulence model in order to get better results. To do that, several algorithms have been proposed: some of them are based on the gradient computation for the minimization of a cost function, other, doing without derivative computations, correct input fields taking advantage of Bayesian statistics techniques. In this work the Ensemble Kalman Filter has been analysed and exploited [9]. At first, ν_T field is directly inferred

and used in the customized OpenFOAM solver *nutFoam* to compute mean longitudinal velocity field, then we try to correct input coefficients of Eddy Viscosity turbulence models, both keeping them constants over the whole domain and then considering them as a non constant field. Numerical results have been obtained for the common cases of Turbulent Channel Flow and Periodic Hills Channel.

Thesis outline

In Chapter 1 the general setting of turbulence modeling is presented, from Navier-Stokes equations (Section 1.1) to the Eddy Viscosity models (sections 1.2, 1.3). Finally, the SIMPLE algorithm and its implementation are discussed (Section 1.4) and a base case is shown with its implementation in OpenFOAM (Section 1.5).
In Chapter 2 the Field Inversion framework is presented (Section 2.1) and some techniques are further analysed: gradient based methods (Section 2.2) and derivative free methods (Section 2.3), among which the Ensemble Kalman Filter (Section 2.3.1). Finally, an example of Field Inversion coupled with the Navier-Stokes equation is shown (Section 2.4).
In Chapter 3 some complete cases are shown and discussed and a new OpenFOAM model implementation is described to allow scalar field as input coefficients for k-ω model.

1 | The Navier-Stokes equations and Turbulence models

In this chapter the Navier-Stokes equations are briefly presented (Section 1.1), from their general formulation to a more applied version, for the specific case of Newtonian fluid in incompressible flows. Then, two different regimes are distinguished, laminar and turbulent, and discussed in relation to the dimensionless Reynolds number. We will focus, in particular, on turbulent regime (Section 1.2) and some of the most effective mathematical models are presented (Section 1.3): the eddy viscosity models (k-ε, k-ω). Finally, the SIMPLE algorithm and its OpenFOAM implementation are presented, to solve the system resulting from the discrete formulation of the problem (Section 1.4).

1.1. The Navier-Stokes equations

In the latest years of 19^{th} century, the well known Navier-Stokes equations were formulated describing the flow of a fluid. Joining the momentum balance equation for a fluid volume with the mass conservation law they retrieved the system in its more general form:

$$\begin{cases} \rho\dfrac{\partial}{\partial t}u_i + \rho u_j\dfrac{\partial}{\partial x_j}u_i - \dfrac{\partial}{\partial x_j}T_{ij} - \rho f_i = 0 \ , \ \text{on } \Omega & \text{(1.1a)} \\ \dfrac{\partial}{\partial t}\rho + \dfrac{\partial}{\partial x_j}(\rho u_j) = 0 \ , \ \text{on } \Omega & \text{(1.1b)} \end{cases}$$

where the componentwise notation and Einstein notation are exploited. $\boldsymbol{u} \in \mathbb{R}^n$ is the velocity field, ρ the fluid density, $\boldsymbol{f} \in \mathbb{R}^n$ an external forcing, $T \in \mathbb{R}^{n\times n}$ the stress tensor describing the properties of the fluid considered and $i, j = 1, ..., n$ the vector components. In case of compressible flows, an energy evolution equation and the state equation need to be provided.
Instead, in this work we will focus only on incompressible flows for Newtonian fluids. Once

written $T = -pI + \tau$, where p is the pressure field, for a Newtonian fluid the stresses have been modelled through a linear and isotropic relation with respect to the strain rate tensor $S := \frac{1}{2}(\nabla \boldsymbol{u} + \nabla^T \boldsymbol{u})$. Thus: $\tau = 2\mu S - \frac{2}{3}\mu(\nabla \cdot \boldsymbol{u})I$, where μ is the dynamic viscosity of the fluid. On the other hand, incompressibility means $\frac{\partial}{x_j} u_j = 0$ in the whole domain. Then, (1.1) for Newtonian fluids, in incompressible flows reads:

$$\begin{cases} \rho \dfrac{\partial}{\partial t} u_i + \rho u_j \dfrac{\partial}{\partial x_j} u_i + \dfrac{\partial}{\partial x_i} p - \mu \dfrac{\partial^2}{\partial x_j^2} u_j - \rho f_i = 0 \ , \ \text{on } \Omega & \text{(1.2a)} \\ \dfrac{\partial}{\partial x_j} u_j = 0 \ , \ \text{on } \Omega & \text{(1.2b)} \end{cases}$$

We will consider in the following the case where ρ in constant.
As last consideration about the formulation of the Navier-Stokes system, a more versatile version can be obtained by normalizing all the quantities by a reference value, depending on the case at hand. Let consider, for instance, a channel flow where δ is the channel half-height and U a reference velocity magnitude. Then, we can assume $\boldsymbol{U}$ and $L = 2\delta$ as reference quantities for that flow, and defining $\boldsymbol{u}^* := \frac{\boldsymbol{u}}{U}$, $\boldsymbol{x}^* := \frac{\boldsymbol{x}}{L}$, $t^* := \frac{t}{L/U}$, $p^* := \frac{p}{\rho U^2}$, $\boldsymbol{f}^* := \frac{\boldsymbol{f}}{U^2/L}$, $\nabla^* := L\nabla$ and $\Delta^* := L^2\Delta$ as dimensionless quantities and operators, (1.2) becomes:

$$\begin{cases} \dfrac{\partial}{\partial t^*} u_i^* + u_j^* \dfrac{\partial}{\partial x_j^*} u_i^* + \dfrac{1}{\rho} \dfrac{\partial}{\partial x_i^*} p^* - \dfrac{1}{\mathrm{Re}} \dfrac{\partial^2}{\partial x_j^{*2}} u_j^* - f_i^* = 0 \ , \ \text{on } \Omega & \text{(1.3a)} \\ \dfrac{\partial}{\partial x_j^*} u_j^* = 0 \ , \ \text{on } \Omega & \text{(1.3b)} \end{cases}$$

where $\mathrm{Re} := \frac{UL}{\nu}$, being $\nu := \frac{\mu}{\rho}$ the kinematic viscosity of the fluid.
Note that from now on (1.3) formulation is considered, but the * notation will be avoided for the sake of clarity: it allows to model different scales and fluids with the same numerical solution, provided that they have the same Re.
Considered a flow case, the value of Re characterizes whether it is in a laminar or turbulent regime. Up to $Re \approx 2000$ the flow is considered to be laminar: streamlines ordered and it is well simulated by the usual weak/discrete formulation workflow of the two equations in (1.3) only. For $\mathrm{Re} \gtrsim 2000$, instead, the flow begins to be chaotic, presenting more complex structures: this case needs to be properly modelled, as analysed in Section 1.2.

1.2. Turbulent flow models

A straightforward solution process leads to try solving eq.(1.2) in the same way as for laminar regime, with a more refined mesh to capture also the smallest turbulent structures: the idea behind DNS strategy moves in this direction. However, the smallest scales (called Kolmogorov scales, η) can be shown, by dimensional arguments, to be $\eta \sim \mathrm{Re}^{-\frac{3}{4}} L$, $u_\eta \sim \mathrm{Re}^{-\frac{1}{4}} U$ for velocities and $t_\eta \sim \mathrm{Re}^{-\frac{1}{2}} \frac{L}{U}$. Suppose to simulate even a simple turbulent flow, such as a car driving at $U = 15\mathrm{m/s}$, with reference dimension $L = 1\mathrm{m}$, in the air ($\nu = 1.5 \cdot 10^{-5}\mathrm{m}^2/\mathrm{s}$): thus $\mathrm{Re} = 10^6$ (cfr. [19]), and the the smallest scales are approximately $\eta \sim 3.2 \cdot 10^{-5}\mathrm{m}$ and $t_\eta \sim 6.67 \cdot 10^{-5}\mathrm{s}$ for time. A spatial mesh refinement of order 10^{-5} is required to capture Kolmogorov scale for length in each direction, thus entailing a tridimensional mesh of about $3.2 \cdot 10^13$ nodes, and a time step discretization of order 10^{-5}. $3.2 \cdot 10^{16}$ points in the time-space mesh are needed, making the computation too expensive to be performed even for common cases.

For this reason, other strategies, modelling the turbulence rather than solving it, have found large application.
Let us consider a scalar or vector quantity ϕ as the sum of a mean value $\bar{\phi}$ and a fluctuating component ϕ', such that $\bar{\phi'} = 0$ (zero-mean fluctuation).
Thus: $\phi(\boldsymbol{x}) = \bar{\phi}(\boldsymbol{x}) + \phi'(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \Omega$. The Reynolds Averaged Navier-Stokes equations (RANS) aim to model the mean fields for the flow ($\bar{\boldsymbol{u}}$ for velocity, $\bar{p}$ for pressure) and to model the fluctuating components $\boldsymbol{u}'$ and p'. In case of statistically stationary flow, the mean field for quantity ϕ is obtained by applying the mean operator $\bar{\phi}(\boldsymbol{x}) \coloneqq \frac{1}{T}\int_0^T \phi(\boldsymbol{x}, t)dt$ to (1.2),
After some computations, the RANS system reads:

$$\begin{cases} \dfrac{\partial}{\partial t}\bar{u_i} + \bar{u_j}\dfrac{\partial}{\partial x_j}\bar{u_i} + \dfrac{1}{p}\dfrac{\partial}{\partial x_i}\bar{p} - \nu\dfrac{\partial^2}{\partial x_j^2}\bar{u_j} - \dfrac{1}{\rho}\dfrac{\partial}{\partial x_j}\tau_{ij}^R - \bar{f_i} = 0 \ , \ \text{on } \Omega & \text{(1.4a)} \\ \dfrac{\partial}{\partial x_j}\bar{u_j} = 0 \ , \ \text{on } \Omega & \text{(1.4b)} \end{cases}$$

where the Reynolds stress tensor τ^R is introduced, collecting all the fluctuating terms appearing in (1.4). More precisely: $\tau_{ij}^R \coloneqq -\rho\overline{u_i' u_j'}$. It results in a 4 equations system with 10 unknowns ($\bar{\boldsymbol{u}}, \tau_{ij}^R, \bar{p}$): it needs to be properly closed in order to be solved.
Several ideas have been proposed to close (1.4) such as the Eddy viscosity models and the Reynolds stress models. In Section 1.3 some of the Eddy viscosity models are analysed. In this work we focus on statistically steady flow, the time dependence is excluded, con-

sidering $\frac{\partial}{\partial t}\bar{\boldsymbol{u}} = 0$.

As an intermediate method between RANS and DNS, Large Eddy Simulation (LES) can be considered: a spatial filtering operation is applied to the velocity field. The resolved scales of turbulence are simulated directly, whereas it is assumed that the scales below a given threshold size have a more universal nature, and then they are modelled through a closure model.

1.3. Eddy viscosity models

A possible way to close the RANS system in (1.4) is represented by the Eddy viscosity models, where the Reynolds stress tensor τ^R is modelled by exploiting the so called Boussinesq assumption (cfr. [8]).
Let us consider the term $\tau_{ij}^R := -\rho\overline{u_i' u_j'}$ that we want to model. We can, at first, decompose it in a diagonal, isotropic tensor and in an anisotropic tensor τ_{ij}^A: $\tau_{ij}^R = \frac{1}{3}\delta_{ij}\tau_{kk}^R + \tau_{ij}^A$. In (1.4) the $j-$th spatial derivative is considered, thus $\frac{\partial}{\partial x_j}\tau_{ij}^R = \frac{1}{3}\delta_{ij}\frac{\partial}{\partial x_j}\tau_{kk}^R + \frac{\partial}{\partial x_j}\tau_{ij}^A = \frac{1}{3}\frac{\partial}{\partial x_i}\tau_{kk}^R + \frac{\partial}{\partial x_j}\tau_{ij}^A$. By defining the turbulent kinetic energy per unit mass $k := \frac{1}{2}\overline{u_i'^2}$, we can write $\frac{\partial}{\partial x_j}\tau_{ij}^R = -\frac{2}{3}\rho k + \frac{\partial}{\partial x_j}\tau_{ij}^A$ and (1.4) becomes:

$$\begin{cases} \dfrac{\partial}{\partial t}\bar{u}_i + \bar{u}_j\dfrac{\partial}{\partial x_j}\bar{u}_i + \dfrac{1}{\rho}\dfrac{\partial}{\partial x_i}P - \nu\dfrac{\partial^2}{\partial x_j^2}\bar{u}_j - \dfrac{1}{\rho}\dfrac{\partial}{\partial x_j}\tau_{ij}^A - \bar{f}_i = 0 \ , \ \text{on } \Omega & (1.5a) \\ \dfrac{\partial}{\partial x_j}\bar{u}_j = 0 \ , \ \text{on } \Omega & (1.5b) \end{cases}$$

where $P := \bar{p} + \frac{2}{3}\rho k$.

Thus our focus can be only on modeling the anisotropic term τ_{ij}^A. Following Boussinesq idea, we can proceed in analogy with the kinetic theory of gases, assuming the hypothesis of similarity between the motion of turbulent structures in a flow and the molecular agitation: the resulting expression is $\tau_{ij}^A = 2\rho\nu_T\bar{S}_{ij}$, which shares such a similarity with the expression of the stress tensor T for a Newtonian fluid.
In this way, the focus is uniquely on ν_T, the so-called turbulent kinematic viscosity, since it has the same dimension as a kinematic viscosity.

We now compare two widely used families of methods to model this term: one equation models, whose most famous case is the Spalart-Allmaras model, and the two equations models, such as the k-ω and the k-ε models.

For the former, the turbulent kinematic viscosity ν_T is modelled by mean of a single PDE for the evolution of ν_T itself. The latter, instead, propose a dimension-coherent expression for ν_T, as a function of the turbulent kinetic energy k and of a term related to the dissipation rate and then solve a couple of PDE for these two terms: note that, being a kinematic viscosity, $\nu_T = [V][L]$, meaning that ν_T has the same dimensions as the product of a velocity and a length.

1.3.1. The Spalart-Allmaras model

Spalart-Allmaras model ([16]) was proposed in 1994 to describe the evolution of ν_T by mean of a single PDE. It consists of a transport model for ν_T itself, where a production and destruction term appear. The turbulent kinematic viscosity is computed as $\nu_T = \tilde{\nu} f_{v1}$, being $f_{v1} \coloneqq \frac{\chi^3}{\chi^3 + c_{v1}^3}$ and $\chi \coloneqq \frac{\tilde{\nu}}{\nu}$. $\tilde{\nu}$ obeys the transport equation:

$$\frac{\partial}{\partial t}\tilde{\nu} + u_j \frac{\partial}{\partial x_j}\tilde{\nu} = \frac{1}{\sigma}\left[\frac{\partial}{\partial x_j}(\nu + \tilde{\nu})\frac{\partial}{\partial x_j}\tilde{\nu} + c_{b2}\frac{\partial}{\partial x_i}\tilde{\nu}\frac{\partial}{\partial x_i}\tilde{\nu}\right] + \\ + c_{b1}\left(1 - f_{t2}\right)S\tilde{\nu} - \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right](\frac{\tilde{\nu}}{d})^2 \ , \ \text{on } \Omega \quad (1.6)$$

Moreover, $\tilde{S} \coloneqq S - \frac{\tilde{\nu}}{\kappa^2 d^2}$ and $f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}}$, with $S = \sqrt{\frac{\partial}{\partial x_j}u_i - \frac{\partial}{\partial x_i}u_j}$ the magnitude of the vorticity and d the distance from the closest wall. Finally, $f_w \coloneqq g\left[\frac{1+c_{w3}^6}{g^6+c_{w3}^6}^{\frac{1}{6}}\right]$, with $g \coloneqq r + c_{w2}(r^6 - r)$ and $r \coloneqq \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}$ (which can be truncated at 10, since f_w reaches a plateau value for high r). At the boundary, $\tilde{\nu} = 0$.

This model was designed for applications involving wall-bounded flows and has been shown to give good results for boundary layers subjected to adverse pressure gradients.

1.3.2. The k-ε model

Let us define the dissipation rate per unit mass of kinetic energy as $\varepsilon \coloneqq \nu\overline{\frac{\partial v_i'}{\partial x_j}\frac{\partial v_i'}{\partial x_j}}$.
In this model, a couple of partial differential equations describes the behaviour of k and ε, such to compute ν_T. Being the dimensions of $\varepsilon = [V]^3[L]^{-1}$, $k = [V]^2$ and $\nu_T = [V][L]$, the unique way to express this latter as function of k and ε is $\nu_T \propto \frac{k^2}{\varepsilon}$ by mean of a dimensional analysis. In particular, we will consider C_μ a non-dimensional constant such that $\nu_T = C_\mu \frac{k^2}{\varepsilon}$.

After some filtering and modeling, we get the following advection-reaction-diffusion equation for the unknown term k:

$$\frac{\partial}{\partial t}k + \bar{u}_j\frac{\partial}{\partial x_j}k = \frac{\partial}{\partial x_j}\left[(\nu + \frac{\nu_T}{\sigma_k})\frac{\partial}{\partial x_j}k\right] + \frac{1}{\rho}\tau_{ij}^R\frac{\partial}{\partial x_j}\bar{u}_i - \varepsilon \ , \ \text{on } \Omega \tag{1.7}$$

$\bar{u}$ is the advective field transporting k, $\frac{\partial}{\partial x_j}\left[(\nu + \frac{\nu_T}{\sigma_k})\frac{\partial}{\partial x_j}k\right]$ represents its diffusion and $\frac{1}{\rho}\tau_{ij}^R\frac{\partial}{\partial x_j}\bar{u}_i$ and $-\varepsilon$ play the role of production and external dissipation term, respectively. For the sake of simplicity, the equation for ε is built by analogy with (1.7), thus reading:

$$\frac{\partial}{\partial t}\varepsilon + \bar{u}_j\frac{\partial}{\partial x_j}\varepsilon - \frac{\partial}{\partial x_j}\left[(\nu + \frac{\nu_T}{\sigma_\varepsilon})\frac{\partial}{\partial x_j}\varepsilon\right] - C_{\varepsilon,1}\frac{\varepsilon}{k}\tau_{ij}^R\frac{\partial}{\partial x_j}\bar{u}_i + C_{\varepsilon,2}\frac{\varepsilon^2}{k} = 0 \ , \ \text{on } \Omega \tag{1.8}$$

The involved coefficients are calibrated to best suit some benchmark cases, and common choices are $C_\mu = 0.09, C_{\varepsilon,1} = 1.44, C_{\varepsilon,2} = 1.92, \sigma_k = 1$ and $\sigma_\varepsilon = 1.3$ ([22]).

However, a very refined mesh is required to properly capture what happens close to the wall in the viscous subregion. For high Re this region could be as thin to make the computation too heavy: a universal modeling needs to be set up to capture the behaviour of the flow when sufficiently close to the wall. Moreover, boundary conditions for the inlet need to be discussed.
To solve the first issue, the so called 'Law of the Wall' has been proposed [21]. Focusing on the region near the wall ($\frac{y}{\delta} \ll 1$, with δ a characteristic length for the flow, for example the half-height of a channel), we define the references quantities:

$$u_\tau \coloneqq \sqrt{\frac{\tau_w}{\rho}} \qquad \text{friction velocity}$$
$$y_+ \coloneqq \frac{y}{\delta_\nu} \qquad \text{wall coordinate}$$

τ_w represents shear stress at the wall and, starting from definition of τ for Newtonian fluids given in Section 1.1, it can be computed as:

$$\tau_w = \tau_{xy}|_{y\approx 0} = 2\mu S_{xy} - \frac{2}{3}\mu\left(\nabla\cdot\boldsymbol{u}\right)\delta_{xy} = \mu\left[\frac{\partial}{\partial y}u_x + \frac{\partial}{\partial x}u_y\right] \approx \mu\left[\frac{\partial}{\partial y}u_x\right] \tag{1.10}$$

while $\delta_\nu := \frac{\nu}{u_\tau}$ is the viscous lengthscale.

In the region closest to the wall (viscous sublayer, approximately $y_+ < 5$), it can be consider valid $\bar{u}_x = u_\tau y_+$, while in the so called 'Logarithmic region' (approximately $y_+ > 30$ but still $\frac{y}{\delta} \ll 1$) the logarithmic law of the wall holds: $\bar{u}_x = u_\tau \left[\frac{1}{\kappa} \log y_+ + A\right]$, being $A = 5.5$ and $\kappa \in [0.38, 0.43]$, the von Karman constant, both determined experimentally. In Figure 1.1 a profile for $\bar{u}_x$ is sketched in a log-linear plot.

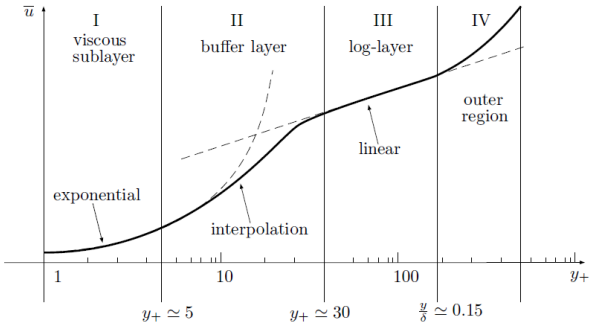

Figure 1.1: Mean streamwise velocity profile $\bar{u}_x$ near the wall (log-linear plot).
We can distinguish the linear behaviour in the viscous sublayer and the logarithmic profile in the log-layer.
Picture from [19].

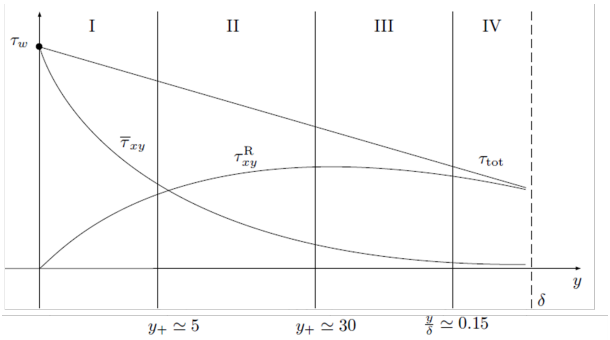

Figure 1.2: $\tau_{TOT} = \bar{\tau}_{xy} + \tau^R_{xy}$: in the viscous sublayer the whole τ_{TOT} can be explained by means of $\bar{\tau}_{xy}$, while in the log-layer they are of the same order of magnitude.
Picture from [19].

Moreover, as can be seen in Figure 1.2, in the logarithmic region the total shear stress τ_{TOT} is mainly due to the Reynolds shear stress τ^R_{xy}:

$$\tau_{TOT} = \bar{\tau}_{xy} + \tau^R_{xy} \approx \tau^R_{xy} = \tau^A_{xy} = 2\rho\nu_T \bar{S}_{xy} = 2\rho\nu_T \frac{1}{2}\left(\frac{\partial}{\partial y}\bar{u}_x + \frac{\partial}{\partial x}\bar{u}_y\right) = \\ = 2\rho\nu_T \frac{1}{2}\left(\frac{\partial}{\partial y}\bar{u}_x\right) = \rho\nu_T \frac{u_\tau}{\kappa y} \quad (1.11)$$

but also

$$\tau_{TOT} = \bar{\tau}_{xy} + \tau^R_{xy} = \rho u_\tau^2 - Ky \tag{1.12}$$

with the terms $\bar{\tau}_{xy}$ and Ky negligible near the wall. Thus, it results in $\rho\frac{\nu_T u_\tau}{\kappa y} = \rho u_\tau^2$, and by expressing $\nu_T = C_\mu \frac{k^2}{\varepsilon}$ as said before:

$$\varepsilon = \frac{C_\mu k^2}{u_\tau \kappa y} \tag{1.13}$$

Finally, consider $\varepsilon_{FLU} \coloneqq \varepsilon - \frac{\nu}{\rho}\frac{\partial^2}{\partial x_i \partial x_j}\tau^R_{ij} \approx \varepsilon - \frac{\nu}{\rho}\frac{\partial^2}{\partial x \partial x}\tau^R_{xy} \approx \varepsilon$.
Then, $\rho\varepsilon_{FLU} = \tau^R_{ij}\hat{S}_{ij} \approx \tau^R_{xy}\hat{S}_{xy} + \tau^R_{yx}\hat{S}_{yx} = 2\tau^R_{xy}\hat{S}_{xy} = 2\tau^R_{xy}\frac{1}{2}\frac{\partial}{\partial y}\hat{u}_x = \rho\frac{u_\tau^3}{\kappa y}$, therefore we can write:

$$\varepsilon = \frac{u_\tau^3}{\kappa y} \tag{1.14}$$

and combining 1.13 and 1.14, we get the expression for k, which results to be constant inside the logarithmic layer:

$$k = \frac{u_\tau^2}{\sqrt{C_\mu}} \tag{1.15}$$

To sum up the model near the wall ($30 \leq y_+ \leq 100$, to be verified a posteriori):

$$\begin{cases} \bar{u}_x = u_\tau \left[\dfrac{1}{\kappa}\log y_+ + A\right] & (1.16) \\ \varepsilon = \dfrac{C_\mu k^2}{u_\tau \kappa y} & (1.17) \\ k = \dfrac{u_\tau^2}{\sqrt{C_\mu}} & (1.18) \end{cases}$$

and in the viscous sublayer, the closest to the wall:

$$\bar{u}_x = u_\tau y_+$$

As for the inlet boundary conditions, provided the average velocity field U at the inlet, and given a turbulence intensity level defined as $I \coloneqq \sqrt{\frac{\overline{v_x'^2}}{U^2}}$, if the fluctuations $\overline{v_i'^2}$ are considered isotropic ($\overline{v_i'^2} = \overline{v_j'^2}, \forall i,j = x,y,z$), we can write $k = \frac{1}{2}(\overline{v_x'^2} + \overline{v_y'^2} + \overline{v_z'^2}) =$

$\frac{2}{3}(IU)^2$. Common values for I are included in the range 5%-20%. Finally, for ε, define $Re_T \coloneqq \frac{lk^{1/2}}{\nu_T}$ the Reynolds number based on the turbulent fluctuations and the turbulent length scale l and assume it to have order of magnitude 1: $\frac{lk^{1/2}}{\nu_T} = \frac{l\varepsilon k^{1/2}}{C_\mu k^2} = 1$, then $\varepsilon = \frac{C_\mu k^{3/2}}{l}$ at the inlet.

1.3.3. The k-ω model

This model has been proposed in 1942 by Kolmogorov and further developed and corrected by other (Landau and Spalding, Wilcox, Speziale). Here the couple of partial differential equations describe k, the turbulent kinetic energy (again $k = [V]^2$), and ω, which can be seen as a characteristic frequency or the dissipation rate ε normalized by the turbulent kinetic energy k (i.e.: $\frac{\varepsilon}{k}$). In any case, $\omega = [T]^{-1} = [V][L]^{-1}$. Therefore, by matching the dimensions, we can express $\nu_T = \frac{k}{\omega}$. By noting that in 1.3.2 $\nu_T = C_\mu \frac{k^2}{\varepsilon}$, we can retrieve the coupled equations for k-ω by substituting $\varepsilon = C_\mu k\omega$. Then (1.7) becomes:

$$\frac{\partial}{\partial t}k + \bar{u}_j \frac{\partial}{\partial x_j}k - \frac{\partial}{\partial x_j}\left[(\nu + \sigma^*\nu_T)\frac{\partial}{\partial x_j}k\right] = \frac{1}{\rho}\tau_{ij}^R \frac{\partial}{\partial x_j}\bar{u}_i - \beta^* k\omega \ , \ \text{on } \Omega \tag{1.19}$$

and (1.8) becomes:

$$\frac{\partial}{\partial t}\omega + \bar{u}_j \frac{\partial}{\partial x_j}\omega - \frac{\partial}{\partial x_j}\left[(\nu + \sigma\nu_T)\frac{\partial}{\partial x_j}\omega\right] - \alpha\frac{1}{\rho}\frac{\omega}{k}\tau_{ij}^R \frac{\partial}{\partial x_j}\bar{u}_i + \beta\omega^2 = 0 \ , \ \text{on } \Omega \tag{1.20}$$

The involved coefficients are calibrated to best suit some benchmark cases. OpenFOAM sets as default values: $\sigma^* = 0.5$, $\beta^* = 0.09$, $\sigma = 0.5$, $\alpha = 0.52$ and $\beta = 0.072$.

For our purposes, the k-ω model has been preferred to the k-ε model, exploiting its superior performance for wall-bounded boundary layers and its suitability for adverse pressure gradient and separation.
In general, the two equations approach is more precise but entails a greater computational burden than Spalart-Allmaras model. In [17], whose result is reported in Section 2.4, Spalart-Allmaras is used as low fidelity but not expensive model, and k-ω model as high fidelity but more expensive model, while in our cases of Chapter 3, k-ω model is used both for low and high fidelity simulations, with different grid refinement.

1.4. The SIMPLE algorithm

The Semi-Implicit Method for Pressure Linked Equations (SIMPLE), presented in [14], is an iterative method proposed by Spalding to solve Navier-Stokes system.

Consider again Navier-Stokes equations. Let us focus on the steady, vector formulation, and take $\boldsymbol{f} = \mathbf{0}$:

$$\begin{cases} (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} - \nu \Delta \boldsymbol{u} = -\dfrac{1}{\rho} \nabla p \ , \ \text{on } \Omega & \text{(1.21a)} \\ \nabla \cdot \boldsymbol{u} = 0 \ , \ \text{on } \Omega & \text{(1.21b)} \end{cases}$$

Note that from now on, for the sake of simplicity and to fit the usual notation, we will denote by p what would properly be $\frac{1}{\rho}p$, the so called kinematic pressure.
Following the usual workflow, the weak formulation of (1.21) reads:

"Find $(\boldsymbol{u}, p) \in (V \times Q)$ such that:

$$\begin{cases} a(\boldsymbol{u}, \boldsymbol{v}) + c(\boldsymbol{u}, \boldsymbol{u}, \boldsymbol{v}) = -b(\boldsymbol{v}, p) & \text{(1.22a)} \\ b(\boldsymbol{u}, q) = 0 & \text{(1.22b)} \\ \boldsymbol{u} = \boldsymbol{g} \quad , \quad \text{on } \Gamma_D & \text{(1.22c)} \end{cases}$$

$\forall \boldsymbol{v} \in V_0, \forall q \in Q$"

being $V \coloneqq [H^1(\Omega)]^n$ and $Q \coloneqq L^2(\Omega)$,with n the domain dimension, proper spaces for $\boldsymbol{u}$ and p, $\boldsymbol{v}$ and q suitable test functions, $\boldsymbol{g}$ a function representing Dirichlet boundary conditions on $\Gamma_D \subset \partial\Omega$ and $V_0 \subset V$ of functions vanishing on Γ_D. Moreover, $a(\cdot,\cdot)$, $b(\cdot,\cdot)$, $c(\cdot,\cdot,\cdot)$ are bilinear and trilinear forms, defined as follows:

$$a(\boldsymbol{u}, \boldsymbol{v}) \coloneqq \int_\Omega \nu \nabla \boldsymbol{u} : \nabla \boldsymbol{v} d\Omega \tag{1.23a}$$

$$b(\boldsymbol{u}, q) \coloneqq -\int_\Omega q \nabla \cdot \boldsymbol{v} d\Omega \tag{1.23b}$$

$$c(\boldsymbol{w}, \boldsymbol{u}, \boldsymbol{v}) \coloneqq \int_\Omega (\boldsymbol{w} \cdot \nabla)\boldsymbol{u} \cdot \boldsymbol{v} d\Omega \tag{1.23c}$$

By expressing each function with a discrete approximation in a desired space, the discrete

formulation reads similarly:

"Find $(\boldsymbol{u}_h, p_h) \in (V_h \times Q_h)$ such that:

$$\begin{cases} a(\boldsymbol{u}_h, \boldsymbol{v}_h) + c(\boldsymbol{u}_h, \boldsymbol{u}_h, \boldsymbol{v}_h) = -b(\boldsymbol{v}_h, p_h) & \text{(1.24a)} \\ b(\boldsymbol{u}_h, q_h) = 0 & \text{(1.24b)} \\ \boldsymbol{u}_h = \boldsymbol{g}_h \quad , \quad \text{on } \Gamma_D & \text{(1.24c)} \end{cases}$$

$\forall \boldsymbol{v}_h \in V_{h,0}, \forall q_h \in Q_h$"

where all the functions are considered discretized in the proper spaces V_h, Q_h and $V_{h,0}$, of dimension N_v, N_q and N_v respectively.

Since the system (1.24) needs to be fulfilled for each test functions in $V_{h,0} \times Q_h$, it is enough to be satisfied for all the (finite) basis functions of those spaces. Denoted as $\{\boldsymbol{\phi}_j\}_{j=1}^{N_v}$ the vector basis functions for $V_{h,0}$ and $\{\psi_j\}_{j=1}^{N_q}$ the scalar basis functions for Q_h, we can express $\boldsymbol{u}_h$ and p_h as a linear combination of the basis functions: thus $\boldsymbol{u}_h = \sum_j^{N_v} U_j \phi_j$ and $p_h = \sum_j^{N_q} P_j \psi_j$.
Substituting this formulation in 1.24 can be written in its matrix formulation as:

$$\begin{cases} A\boldsymbol{U} + C(\boldsymbol{U})\boldsymbol{U} + B^T \boldsymbol{P} = 0 & \text{(1.25a)} \\ B\boldsymbol{U} = 0 & \text{(1.25b)} \end{cases}$$

with the matrices resulting from the computation of the bilinear and trilinear forms. Namely:

$$A_{ij} = a(\phi_j, \phi_i) \ , \ A \in \mathbb{R}^{N_v} \times \mathbb{R}^{N_v} \quad \text{(1.26a)}$$

$$B_{ij} = b(\phi_j, \psi_i) \ , \ B \in \mathbb{R}^{N_q} \times \mathbb{R}^{N_v} \quad \text{(1.26b)}$$

$$C(\boldsymbol{U})_{ij} = \sum_{k=1}^{N_v} U_k c(\phi_k, \phi_j, \phi_i) \ , \ C(\boldsymbol{U}) \in \mathbb{R}^{N_v} \times \mathbb{R}^{N_v} \quad \text{(1.26c)}$$

which can be reduced to:

$$M(\boldsymbol{U}) \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{P} \end{bmatrix} = \boldsymbol{0} \quad \text{(1.27)}$$

where $M = \begin{bmatrix} \boldsymbol{A} + \boldsymbol{C}(\boldsymbol{U}) & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{0} \end{bmatrix}$, and $\boldsymbol{u}_h$ appearing in this matrix makes the problem non linear, thus needing a specific treatment to linearize this term.

Consider now the formulation in 1.25a, but without including the term ∇p_h in the vectorial form: in particular it can be written as $A\boldsymbol{U} + C(\boldsymbol{U})\boldsymbol{U} = -\nabla p_h$, being ∇p_h a discrete evaluation of ∇p. Being a non linear equation in $\boldsymbol{U}$, an iterative method is needed.
Once set an initial guess both for $\boldsymbol{U}$ ($\boldsymbol{U}_\Gamma 30256$) and $\boldsymbol{P}$ ($\boldsymbol{P}_\Gamma 30256$), define $\tilde{M}^k \coloneqq \tilde{M}(\boldsymbol{U}^k) = A + C(\boldsymbol{U}^k)$ the matrix resulting at each iteration. This matrix can be decomposed in a diagonal and an off-diagonal matrix: $\tilde{M}^k = D^k - H^k$, with D^k diagonal, H^k off-diagonal. Therefore, the algebraic formulation of the momentum equation can be written as $\tilde{M}^k\boldsymbol{U} = (D^k - H^k)\boldsymbol{U} = -\nabla \boldsymbol{P}^k$, which can be solved for $\boldsymbol{U}$ by mean of the inverse of the diagonal matrix D^k:

$$\boldsymbol{U} = {D^k}^{-1}(H^k\boldsymbol{U}^k) - {D^k}^{-1}\nabla \boldsymbol{P}^k \tag{1.28}$$

where $H^k\boldsymbol{U}^{\boldsymbol{k}}$ is considered in place of $H^k\boldsymbol{U}$ to easily solve the equation. Denote the resulting solution $\boldsymbol{U}$ as $\boldsymbol{U}^{k+1/2}$: it is a partial solution, fulfilling the momentum equation (1.25)a, but anything as been said about the continuity equation (1.25b).
Merging $\boldsymbol{U}^{k+1/2}$ just found in 1.25b, we end up with a Poisson problem to be solved for $\boldsymbol{P}$:

$$\nabla \cdot ({D^k}^{-1}\nabla \boldsymbol{P}) = \nabla \cdot ({D^k}^{-1}(H^k\boldsymbol{U}^{\boldsymbol{k}})) \tag{1.29}$$

Denoted by $\boldsymbol{P}^{k+1}$ the solution of (1.29), we correct $\boldsymbol{U}^{k+1/2}$ by substituting $\boldsymbol{P}^{k+1}$ in (1.28), thus obtaining $\boldsymbol{U}^{k+1}$.

Algorithm 1.1 SIMPLE algorithm (coupled with k-ω turbulent model)

1: Initial guess for velocity and pressure fields: $\boldsymbol{U}_0$, $\boldsymbol{P}_0$
2: **while** convergence criterion not fulfilled **do**
3: Build matrix $\tilde{M}^k = A + C(\boldsymbol{U}^k)$
4: Find matrices D^k and H^k s.t. $\tilde{M}^k = D^k - H^k$
5:
6: Solve momentum equation 1.28: $\boldsymbol{U}^{k+1/2}$
7: Solve continuity equation 1.29: $\boldsymbol{P}^{k+1}$
8: Correct the velocity vector by mean of $\boldsymbol{P}^{k+1}$: $\boldsymbol{U}^{k+1/2} \to \boldsymbol{U}^{k+1}$
9:
10: $\boldsymbol{U}^k = \boldsymbol{U}^{k+1}$, $\boldsymbol{P}^k = \boldsymbol{P}^{k+1}$
11: **end while**

A new algorithm iteration can now start, shifting $\boldsymbol{U}^{k+1}$ to $\boldsymbol{U}^k$ and $\boldsymbol{P}^{k+1}$ to $\boldsymbol{P}^k$, up to a

desired convergence criterion.
In case of turbulent flows, once computed $\boldsymbol{U}^{k+1}$, an algebraic system can be solved for the turbulent kinetic energy (k) and for ε or ω, depending on the chosen model.
In Algorithm 1.1, the SIMPLE algorithm coupled with k-ω model is summarized.

1.5. OpenFOAM implementation and test case

OpenFOAM is an open source software developed in C++ to perform high-performance CFD simulations [4]. It provides several solvers and all the most wide spread turbulent models, included those described in Section 1.3. Moreover, it gives the user the possibility to customize his own solvers or models.
As for mesh construction we relied on the application *blockMesh*, included in OpenFOAM itself: all the faces and edges can be specified, together with the type of each mesh face (inlet, outlet, wall...) and the mesh refinement, which can be either homogeneous or locally thickened. Finally, initial and boundary conditions can be easily specified, together with flow properties.
At first, we exploited OpenFOAM to analyse the two most used turbulence models (k-ε and k-ω, see Section 1.3) on a common benchmark case, the bidimensional fully developed turbulent channel flow (TCF): a simulation for each model has been performed and the results have been compared with high-fidelity results, obtained by mean of DNS simulations by [3].

The channel is a 2D domain, with half height $h = 1$m, and without an actual z extension, where just a single computational cell is present (Table 1.1). The domain has been simulated to be longitudinally infinite, thus cyclic boundary condition have been imposed at the inlet and outlet. The lower boundary condition is a no slip condition, due to the presence of the wall, while the upper boundary is meant as a symmetry line, so that only half of the whole channel is simulated, reducing computational costs (Table 1.1). The chosen data makes the case fully turbulent, and since a fully developed flow is considered, a stationary simulation has been preferred.

As a convergence criterion for the iterative solution of the system, U_x, k, ε and ω are requested to be less then or equal to 10^{-3}: this conditions make k-ω to converge in 599 iterations and k-ε to converge in 1329 iterations of the SIMPLE algorithm for the mesh described in Table 1.1. In Figure 1.4 longitudinal velocity profiles obtained with k-ω and k-ε models are compared against DNS observations taken from [3]. On background the complete velocity field is reported for the whole channel height, as obtained by Open-

domain dimensions [m]		mesh refinement		domain BC	
L	8π	$\mathbf{N_x}$	100	**inlet**	cyclic
h	1.0	$\mathbf{N_y}$	100	**outlet**	cyclic
d	0.1	$\mathbf{N_z}$	1	**base wall**	no slip
				central line	symmetry line

Table 1.1: Domain properties and boundary conditions (cfr: [3]).
$N_z = 1$ makes the case bidimensional.

flow properties	
$\boldsymbol{\nu}$	10^{-4} m^2/s
$\mathbf{U_x}$	1.0 m/s
$\mathbf{Re_h}$	10^4
I	5%

Table 1.2: Flow properties.
Re is computed with respect to the half-height of the channel (cfr. [3]). I represents the turbulent intensity at the inlet.

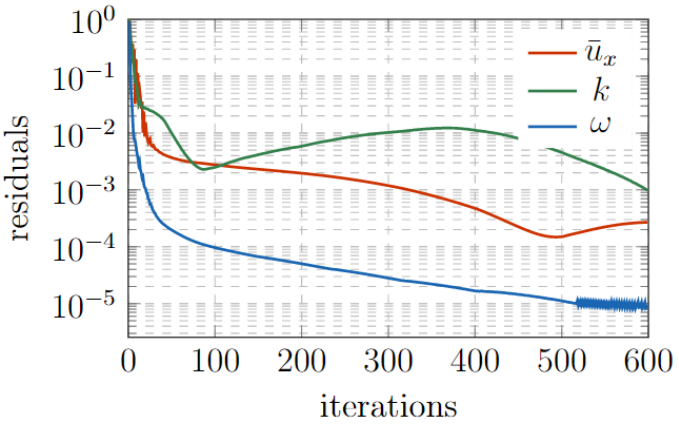

Figure 1.3: Residuals for $\bar{u}_x$, k and ω in *simpleFoam* solver coupled with k-ω model in the turbulent channel flow. Related domain and flow properties are in Tables 1.1 and 1.2

FOAM with the k-ω model. As commonly noted, k-ω model better fits the behaviour of the velocity field close to the wall, while k-ε is slightly preferable in the middle of the channel, where k-ω underestimates the longitudinal velocity field.

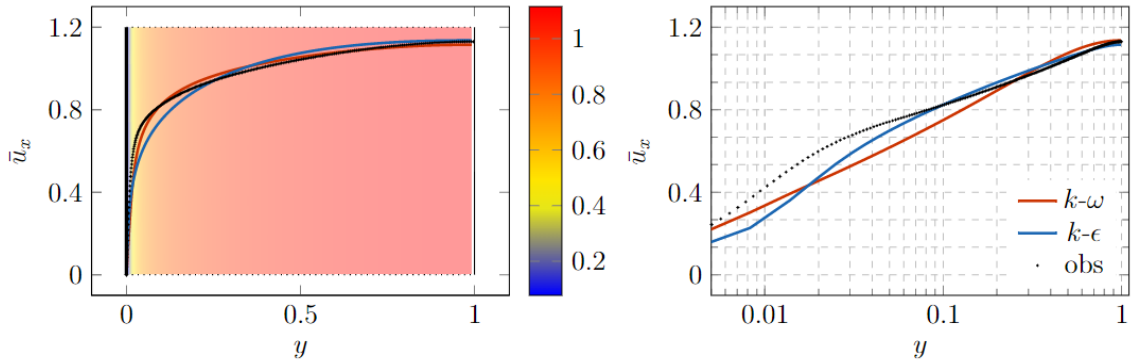

Figure 1.4: k-ε model and k-ω model profiles for longitudinal velocity $\bar{u}_x$ compared to the DNS observations from [3].
On the left side profiles are plotted, on the right side the same profiles are plotted in logarithmic scale.
k-ω better fits the observations close to the wall than k-ε, which is more precise in the central region.
Focus is only on the velocity field because it will be the core of the analysis in Chapter 2 and 3
Mesh is refined closed to the wall through *simpleGrading* OpenFOAM functionality.

2 | Field Inversion

In this chapter Field Inversion (FI) workflow for state estimation is presented. After a general description of the problem and its formulation in the Bayesian framework (Section 2.1) two cases are distinguished: gradient based methods (Section 2.2) and the so called 'Derivative Free methods' (Section 2.3), including the Kalman Filter methods (KF), and in particular the Ensemble Kalman Filter (EnKF, subSection 2.3.1). Finally, DAFI library is presented and its EnKF implementation is exploited, focusing on its coupling with OpenFOAM (Section 2.4).

2.1. The Field Inversion workflow

In many areas of science and engineering, it is a common task to infer physical input fields from observations of an output quantity: in field inversion workflow two sets of fields are related to each other through a forward model and observations of the output fields are used to correct the input fields. The forward model generally consists of a system of PDEs and the resulting numerical model, usually affected by uncertainties, is tried to be improved by incorporating observations. This task can be performed by using data assimilation techniques iteratively: the iterations are required to account for the non-linearity of the forward model since data assimilation filtering techniques assume a linear mapping between the state and observations ([18]).

More formally, denote the forward model by $R(\boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is an input field required by the model, and by $\mathbf{d}$ the measured observations: the goal of the field inversion is to reconstruct $\boldsymbol{\beta}_{OPT}$ from the data, such that $R(\boldsymbol{\beta}_{OPT})$ best approximates $\mathbf{d}$.

One approach is to assume that the data are affected by independent, normally distributed errors and to find the Maximum Likelihood estimator (MLE) to find the corrective field which maximizes the probability of the data given the corrective field: thus, the objective is to estimate $\boldsymbol{\beta}_{MLE} \coloneqq \arg\min_{\boldsymbol{\beta}} \|R(\boldsymbol{\beta}) - \boldsymbol{d}\|^2$.
If the forward model is non-linear, there might be several local optima, or in some cases $\boldsymbol{\beta}_{OPT}$ might not exist, for example because of too heavy noise in the data, or even there

might exist multiple corrective fields leading to the output corresponding to the data. In this case, the solution to the inverse problem is not unique. Finally, the problem might be ill-conditioned, and it is possible that a small change in the data causes a large change in the resulting model.

Another strategy is to consider a Bayesian setting for the problem and find a probability distribution over corrective fields rather than a single corrective field, as in the case of the maximum likelihood approach; from Bayes' rule:

$$p(\boldsymbol{\beta}|\boldsymbol{d}) = \frac{p(\boldsymbol{d}|\boldsymbol{\beta})p(\boldsymbol{\beta})}{p(\boldsymbol{d})} \tag{2.1}$$

is the posterior distribution over the corrective fields given the data, where $p(\boldsymbol{\beta})$ is the prior distribution for $\boldsymbol{\beta}$, $p(\boldsymbol{d}|\boldsymbol{\beta})$ quantifies the likelihood of observing $\boldsymbol{d}$ given parameter realizations $\boldsymbol{\beta}$ and $p(\boldsymbol{d})$ is a normalizing term ensuring the total posterior probability to be 1. Assuming that the observed data $\boldsymbol{d}$ consists of the sum of the forward model output $R(\boldsymbol{\beta})$ and a Gaussian noise $\boldsymbol{\xi} \sim \mathcal{N}(0, C_m)$, with C_m the covariance matrix for $\boldsymbol{\xi}$, then $\boldsymbol{d}|\boldsymbol{\beta} \sim \mathcal{N}(R(\boldsymbol{\beta}), C_m)$. Also, that $\boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{\beta}_{PRIOR}, C_\beta)$, where $\boldsymbol{\beta}_{PRIOR}$ is the prior guess for $\boldsymbol{\beta}$ mean and C_β represents the prior covariance matrix.
Therefore, (2.1) becomes

$$p(\boldsymbol{\beta}|\boldsymbol{d}) \propto e^{-J} \tag{2.2}$$

where

$$\begin{aligned} J &\coloneqq \frac{1}{2}\left\|\boldsymbol{d} - R(\boldsymbol{\beta})\right\|_{C_m} + \frac{1}{2}\left\|\boldsymbol{\beta} - \boldsymbol{\beta}_{PRIOR}\right\|_{C_\beta} \\ &= \frac{1}{2}\left[\boldsymbol{d} - R(\boldsymbol{\beta})\right]^T C_m^{-1}\left[\boldsymbol{d} - R(\boldsymbol{\beta})\right] + \frac{1}{2}\left[\boldsymbol{\beta} - \boldsymbol{\beta}_{PRIOR}\right]^T C_\beta^{-1}\left[\boldsymbol{\beta} - \boldsymbol{\beta}_{PRIOR}\right] \end{aligned} \tag{2.3}$$

By minimizing J, the maximum a posteriori estimate (MAP) is found.
Covariance matrices can be chosen in several ways, depending on the influence assumed among the variables ([20]).

In section 2.2 and 2.3 two different optimization procedures are proposed to minimize the function J in (2.3).

In this work, the forward model $R(\cdot)$ corresponds to the RANS model and the measured data corresponds to high-fidelity data: these data can be taken from experimental mea-

surements, from DNS simulations, or from RANS simulations performed on highly refined meshes.

Refer, in particular, to the RANS model: the forward model captures the correct physics but there are uncertainties in some input fields, such as the Reynolds stress tensor τ^R, which requires a modelisation representing the largest source of uncertainty in RANS simulations. The input quantity $\boldsymbol{\beta}$ to be inferred can be directly τ^R in (1.4), as in [23], or the ν_T term in Eddy Viscosity models (section 1.3), as in [18], or the constant values appearing in the turbulence models equations for k, ε or ω: this tackles the spread of uncertainty in the model and makes the output fields, for instance the velocity field, better fit the experimental data.

2.2. Gradient based methods

Several methods have been proposed to maximize J in (2.3).
In gradient-based optimization methods, an approximation of the gradient of J is used to determine the direction to step into [13]. For the computation of the gradient, it is necessary that the objective function is sufficiently smooth. Furthermore, gradient-based methods only guarantee convergence when there exists a single global minimum or if the initial guess is sufficiently close to the global minimum, given an appropriate step size ([20]).

The most immediate gradient-based optimization algorithm is Gradient Descent Algorithm (or Gradient Ascent Algorithm if we want to maximize $-J$): it is a first-order iterative optimization algorithm to find a local minimum (respectively, maximum) of a differentiable function. The idea is to iteratively use the gradient as the direction in which to change the parameters at step n with a given step size α^n. The i-th element of the corrective terms then updated according to

$$\beta_i^{n+1} = \beta_i^n - \alpha^n \frac{\partial}{\partial \beta_i} J^n \tag{2.4}$$

being, indeed, $\frac{\partial}{\partial \beta_i} J^n$ the gradient of function J, to be approximated somehow. Conjugate Gradient Method ([15]) and its generalization of Fletcher–Reeves nonlinear conjugate gradient method ([10]) improve the optimization by requiring subsequent search directions to be conjugate.

Newton's method, instead, exploits second derivative for each iteration update:

$$\beta_i^{n+1} = \beta_i^n + \alpha^n \left(\frac{\partial^2}{\partial \beta_i \beta_j} J^n \right)^{-1} \frac{\partial}{\partial \beta_i} J^n \tag{2.5}$$

with the Hessian $\frac{\partial^2}{\partial \beta_i \beta_j} J^n$ to be properly estimated ([20]). This method converges quadratically, but, due to the calculation of the Hessian it is significantly more expensive for high-dimensional problems. Furthermore, Newton's method can suffer far away from the optimum and when the Hessian is close to be singular. Quasi-Newton methods trade off the cost of computing the Hessian and the increased convergence of quadratic methods by approximating the Hessian.

In any case, these methods require a numerical approximation of the gradient and the Hessian: this can be achieved by mean of a sensitivity analysis, which has the drawback of requiring a great number of simulations, or by mean of the analysis of the adjoint problem, where the gradients can be obtained at a cost which is practically independent of the number of optimization parameters, as done in [20].

2.3. Derivative Free methods

On the other hand, Derivative-Free Methods perform field inversion avoiding to appeal to the derivatives of function J. A derivative free approach is advantageous, since obtaining the gradient of a cost function with respect to input fields in complex science and engineering models could be a non-trivial task. Genetic algorithms, for instance, use an analogy of the theory of evolution to search for an optimal solution ([7]): the optimization problem is parameterized into a set of genes, and relies on biologically inspired operators such as mutation and crossover. The evolution usually starts from a base population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are selected from the current population, and each individual's genome is recombined to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a desired fitness level has been reached for the population. These algorithms generally work for non-smooth objective functions with multiple local optima, but suffer from relatively slow convergence. Generally, the high-dimensional nature of some optimization problems requires a large population size, and therefore a large number of cost function evaluations.

Another approach to address estimation problems is Kalman Filter (KF), introduced in the 60s' as recursive filters to improve the prediction of the model state by taking into account measured data ([11]): KF is meant for situations where the model is linear and the model and observation errors are assumed to be zero mean Gaussians with given covariance matrices. In case of large dimension problem, Ensemble Kalman Filter (EnKF) is considered, as a Monte-Carlo approximation of KF ([9]).

2.3.1. The Ensemble Kalman Filters

In this work, Derivative-Free approach has been chosen, and in particular EnKF is used to reconstruct the desired parameters.

Consider again (2.3). To introduce KF, start considering a linear forward model, so that the model output can be expressed in a matrix formulation as $\boldsymbol{z} = H\boldsymbol{\beta}$. Moreover, let keep valid assumptions made in Section 2.1: $\boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{\beta}_{PRIOR}, C_\beta)$ and $\boldsymbol{\xi} \sim \mathcal{N}(0, C_m)$. Then to compute

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left[\frac{1}{2} \left\| \boldsymbol{d} - H\beta \right\|_{C_m} + \frac{1}{2} \left\| \boldsymbol{\beta} - \boldsymbol{\beta}_{PRIOR} \right\|_{C_\beta} \right] \tag{2.6}$$

gradient vanishing can be imposed:

$$\nabla \left(J(\boldsymbol{\beta}) \right) = {C_\beta}^{-1} (\boldsymbol{\beta} - \boldsymbol{\beta}_{PRIOR}) - H^T {C_m}^{-1} \left(\boldsymbol{d} - H\boldsymbol{\beta} \right) = 0 \tag{2.7}$$

Then:

$$\begin{aligned} {C_\beta}^{-1} (\boldsymbol{\beta} - \boldsymbol{\beta}_{PRIOR}) &= H^T {C_m}^{-1} \left(\boldsymbol{d} - H\boldsymbol{\beta} \right) \\ \left({C_\beta}^{-1} + H^T {C_m}^{-1} H \right) \boldsymbol{\beta} &= H^T {C_m}^{-1} \boldsymbol{d} + {C_\beta}^{-1} \boldsymbol{\beta}_{PRIOR} \end{aligned} \tag{2.8}$$

which results in:

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= \overbrace{\left[{C_\beta}^{-1} + H^T {C_m}^{-1} H \right]^{-1}}^{P} \left(H^T {C_m}^{-1} \boldsymbol{d} + {C_\beta}^{-1} \boldsymbol{\beta}_{PRIOR} \right) \\ &= P \left[H^T {C_m}^{-1} \boldsymbol{d} + {C_\beta}^{-1} \boldsymbol{\beta}_{PRIOR} + H^T {C_m}^{-1} H \boldsymbol{\beta}_{PRIOR} - H^T {C_m}^{-1} H \boldsymbol{\beta}_{PRIOR} \right] \\ &= P \left[P^{-1} \boldsymbol{\beta}_{PRIOR} + H^T {C_m}^{-1} \left(\boldsymbol{d} - H\boldsymbol{\beta}_{PRIOR} \right) \right] \\ &= \boldsymbol{\beta}_{PRIOR} + P H^T {C_m}^{-1} \left[\boldsymbol{d} - H\boldsymbol{\beta}_{PRIOR} \right] \\ &= \boldsymbol{\beta}_{PRIOR} + K \left[\boldsymbol{d} - H\boldsymbol{\beta}_{PRIOR} \right] \end{aligned} \tag{2.9}$$

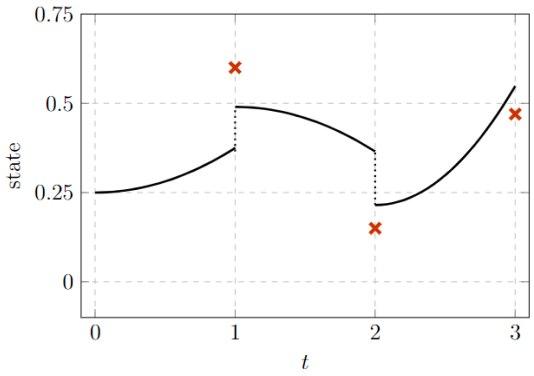

Figure 2.1: Qualitative representation of EnKF algorithm: at each time step a new observation is measured and state variable probability distribution is updated. Then, an ensemble of N evaluations of the model are performed. At each state update step, covariance matrix C'_β is estimated from the sample according to (2.13)

once defined

$$K \coloneqq PH^T {C_m}^{-1} = \left[C_\beta^{-1} + H^T {C_m}^{-1} H\right]^{-1} H^T {C_m}^{-1} \tag{2.10}$$

the Kalman Filter Gain matrix.

Therefore, the posterior consditional distribution of $\boldsymbol{\beta}|\boldsymbol{d}$ can be written as:

$$\boldsymbol{\beta}|\boldsymbol{d} \sim \mathcal{N}(\hat{\boldsymbol{\beta}}, \hat{C}_\beta) \tag{2.11}$$

where

$$\hat{C}_\beta = P = \left[C_\beta^{-1} + H^T {C_m}^{-1} H\right]^{-1} \tag{2.12}$$

The obtained distribution is the posterior distribution, a correction of the prior belief about $\boldsymbol{\beta}$ distribution by mean of data observations. Then, the model is evolved in time exploiting this new belief on input field, and corrected again with a new observation measured at the following time step. In Figure 2.1, a sketch of KF algorithm is shown: for ease of picture, the particular case in which state space and observation space (both scalar spaces) coincide is considered.

The EnKF is a Monte Carlo approximation of KF, which avoids evolving the covariance matrix C_β of $\boldsymbol{\beta}$: the covariances of the current estimate involved in the Kalman filter is approximated by computing the sample covariance of a set (ensemble) of cases (particles) sampled from the distribution. The covariance matrix of the state vector $\boldsymbol{\beta}$ need not be evolved, thus eliminating the costs associated with storing, multiplying and inverting the matrices appearing in (2.12) and (2.10).

It is estimated as

$$C'_\beta = \frac{1}{N-1}\sum_{j=1}^{N}(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}})(\boldsymbol{\beta}-\hat{\boldsymbol{\beta}})^T \tag{2.13}$$

being $\hat{\boldsymbol{\beta}} = \frac{1}{N}\sum_{j=0}^{N}\boldsymbol{\beta}_j$ the sample mean of columns of B.

This evaluation and update procedure can be repeated iteratively: originally, KF were introduced to calibrate model parameters through time (cfr. Figure 2.1), correcting them by assimilating new data at any desired time interval. Thus iterations consist in advancing the model in time. Instead, if a steady case is considered, time advancing dynamics is substituted by an artificial dynamics ([18]): at each iteration, belief on $\boldsymbol{\beta}$ is updated by mean of observations and the model is run again considering state variable sampled from the new updated distribution. This procedure is replicated up to a desired convergence criterion or a maximum number of iterations. In Algorithm 2.1, EnKF algorithm is summarized in its iterative version.

Algorithm 2.1 EnKF algorithm

1: Sample N ensembles $\boldsymbol{\beta}_j \sim p(\boldsymbol{\beta})$: prior ensemble
2: Build matrix $B = [\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_N] \in \mathbb{R}^{n\times N}$
3: Build matrix $D = [\boldsymbol{d}_1, ..., \boldsymbol{d}_N] \in \mathbb{R}^{n\times N}$ of observations $(\boldsymbol{d}_j = \boldsymbol{d} + \boldsymbol{\xi}_j, \boldsymbol{\xi}_j \sim \mathcal{N}(0, C_m))$
4: **while** convergence criterion not fulfilled **do**
5: Compute matrix C'_β, the sample covariance of columns of B
6: Compute matrix $K' = C'_\beta H^T (HC'_\beta H^T + R)^{-1}$
7: Evaluate the model for each column of B : $Z = HB$
8: Update the ensemble matrix B : $\hat{B} = B + K'(D - Z)$
9: **end while**

In Figure 2.2, Figure 2.1 is revised to sketch EnKF for a time dependent model: after a state update due to data measurements, an ensemble of $N_{samples}$ evaluations of the model are performed to estimate the sample covariance matrix C'_β according to (2.13).

Note that, excepted for the prior ensemble, columns of matrix B are not actually independent, since the updated distribution is somehow influenced by each column (i.e.: each particle of the ensemble): however, the method is thought as if they were independent. EnKF is preferable with respect to classical KF for models resulting in an high dimensions numerical solutions.

We are interested to couple EnKF with steady Raynolds Averaged Navier-Stokes equa-

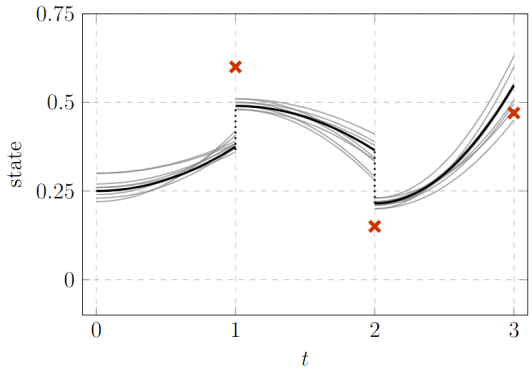

Figure 2.2: Qualitative representation of EnKF algorithm: at each time step a new observation is measured and state variable probability distribution is updated. Then, an ensemble of N evaluations of the model are performed. At each state update step, covariance matrix C'_β and mean value $\hat{\boldsymbol{\mu}}$ are estimated from the sample according to (2.13)

tions, which are fully non linear and with large dimensions numerical solutions: evaluation of the model $z = H\boldsymbol{\beta}$ considered above is replaced by $z = R(\boldsymbol{\beta})$, where $R(\cdot)$ represents our model and iterations are considered as fictitious dynamics for the model, and EnKF algorithm is exploited to perform Field Inversion.

2.4. DAFI implementation

DAFI is an open source library for ensemble-based Data Assimilation and Field Inversion ([1],[2],[17]) written in Python: its structure allows for interchanging different solution methods and different physics problems. The code provides native tools for integrating the library with OpenFOAM and operates in a non intrusive way: no gradient computations are required and no modifications to the physics solver are needed.

DAFI is particularly useful to inverse problems involving continuous fields over a desired domain while does not provide native functionalities to infer constant states: case studied in [17] is here reported as base example to better present and understand DAFI capabilities.

Let us consider (1.5) in a steady case and develop τ_{ij}^A according to Boussinesq assumption presented in section 1.3:

$$\begin{cases} \bar{u_j}\dfrac{\partial}{\partial x_j}\bar{u_i} + \dfrac{\partial}{\partial x_i}p_\rho - \dfrac{\partial}{\partial x_j}\left((\nu+\nu_T)\dfrac{\partial}{\partial x_j}\bar{u_i}\right) = 0 \ , \ \text{on } \Omega & \text{(2.14a)} \\ \dfrac{\partial}{\partial x_j}\bar{u_j} = 0 \ , \ \text{on } \Omega & \text{(2.14b)} \end{cases}$$

where $\bar{\boldsymbol{f}} = 0$ and $p_\rho = \frac{1}{\rho}P$. Following the workflow presented in Section 1.3 the term ν_T

can be modelled according to some further assumptions, exploiting the available closure models (k-ε, k-ω, Spalart-Allmaras, etc.). In this setting, instead, we want to infer ν_T field by mean of Field Inversion, thus doing without a turbulence model. On this purpose, a new customized OpenFOAM solver has been coded by [17], called *nutFoam* and based on the alreay existing *simpleFoam* solver implementing the SIMPLE algorithm (Section 1.4): in particular, a new field `nut` is declared inside `createFields.H` (Listing 2.1) and directly inserted `UEqn.H` where momentum equation is actually solved (Listing 2.3).

```
volScalarField nut (
    IOobject (
        "nut",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

Listing 2.1: createFields.H

```
nut = Foam::max(nut, nutMin);
tmp<fvVectorMatrix> tUEqn (
    fvm::div(phi, U)
  - fvm::laplacian(nut,U)
  - fvm::laplacian(nu,U)
  - fvc::div(nut*dev(T(fvc::grad(U)))) //nut field is directly
                                       considered in the momentum equation
  - fvc::div(nu*dev(T(fvc::grad(U))))
  ==
    fvModels.source(U)
);
```

Listing 2.2: UEqn.H for *nutFoam* solver

```
tmp<fvVectorMatrix> tUEqn (
        fvm::div(phi, U)
      + MRF.DDt(U)
      + turbulence->divDevSigma(U) //turbulence is modelled
     ==
      fvModels.source(U)
);
```

Listing 2.3: UEqn.H for *simpleFoam* solver

For the case at hand, a Periodic Hills Channel (PHC) has been considered. In Figure 2.4 the domain grid can be seen and in Table 2.1 and 2.2 properties of mesh and fluid for this flow case are summarized. Note that both for longitudinal and transversal direction a uniform ratio of expansion for the cells has been chosen to refine the mesh near the boundaries, specified in OpenFOAM by the keyword `simpleGrading` in `system/blockMeshDict` file.

domain dimensions [m]		**mesh refinement**		**domain BC**	
L	9	$\mathbf{N_x}$	100	**inlet**	cyclic
h	2.036	$\mathbf{N_y}$	30	**outlet**	cyclic
d	0.1	$\mathbf{N_z}$	1	**bottom wall**	no slip
$\mathbf{h_H}$	1.0			**top wall**	no slip

Table 2.1: Domain properties (cfr: [17]). $N_z = 1$ makes the case bidimensional. h is meant as the inlet and outlet faces height while h_H and is the height of the hills

flow properties	
$\boldsymbol{\nu}$	$0.00001785714285714285\overline{7}\ m^2/s$
$\mathbf{U}_b$	$[1.0, 0, 0]\ m/s$
$\mathbf{Re_H}$	5600
I	5%

Table 2.2: Flow properties: Re is computed with respect to h_H (cfr. [17]). I represents the turbulence intensity at the inlet.

EnKF analysis is implemented by DAFI in `nutfoam.py` while user input can be inserted in `dafi.in`. In particular, this latter file allows the user to specify:

- `inverse_method`: in our case EnKF;
- `nsamples`: number of samples composing the ensemble;
- `max_iteratios`: number of EnKF iterations to be performed;
- `klmodes_file, nklmodes`: discussed below;
- `nut_baseline_foamfile`: prior mean for ν_T distribution;
- `obs_file`: where read the observations to calibrate the state variables.

To establish a prior mean for ν_T field, a low cost simulation is performed at the beginning, exploiting Spalart-Allmaras model (Section 1.3.1): obtained field is taken as prior mean for EnKF algorithm. Moreover, to generate the prior samples, assumed to be Gaussians, a Gaussian Field around the estimated mean is required. Karhunen-Loeve (KL) decomposition is used to sample an approximation of it: more precisely, the Gaussian random field $G(\mathbf{x}, \omega)$ can be represented as the infinite series

$$G(\mathbf{x}, \omega) = G_0(\mathbf{x}) + \sum_{i=1}^{\infty} \sqrt{\lambda_i} b_i Y_i(\omega), \tag{2.15}$$

where $G_0(\boldsymbol{x})$ is the mean field, λ_i and b_i are the corresponding eigenvalues and eigenvectors of the covariance matrix C_{ν_T} and $Y_i(\omega)$, $i = 1, 2, \ldots$ is a sequence of i.i.d random variables with standard normal distribution. For a suitable N, specified by the user as `nklmodes` in `dafi.in`, the truncated KL series

$$G(\mathbf{x}, \omega) = G_0(\mathbf{x}) + \sum_{i=1}^{N} \sqrt{\lambda_i} b_i Y_i(\omega), \tag{2.16}$$

represents a reliable approximation of the original random field. For this task, eigenvalues and eigenvectors of the covariance matrix C_{ν_T} are computed in `klmodes.py`, where also the kernel can be specified. In this specific case, accounting for the periodic nature of the flow, DAFI relies on a mixed periodic squared exponential kernel, thus the covariance matrix entries can be written as:

$$C_{\nu_T}^{ij} = \sigma^2 exp\left\{ -\left(2\frac{sin^2(|x_i - x_j|\pi/p)}{l_x^2} + \frac{|y_i - y_j|^2}{2l_y^2} + \frac{|z_i - z_j|^2}{2l_z^2} \right) \right\} \tag{2.17}$$

where σ is a user defined coefficient for the variance of the Gaussian Field, l_x, l_y, l_z represent the correlation lengths in each direction, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are all the mesh cells pairs and p the periodicity of the domain. All the computed modes are saved in `klmodes_file` specified in `dafi.in`. [17] shows that the first 200 modes capture the largest part of the variance. Using this reduced set of modes to represent the state reduces the dimensionality of the state space from N_{xy}, corresponding to the number of cells, to 200, corresponding to the number of KL modes. The state would now consist of the 200 coefficients Y_i to be estimated.

As for the observations of mean velocity field $\bar{\boldsymbol{u}}$, [17] proposes to consider values obtained by a high refined RANS simulation coupled with k-ω model: in that work only one observation is considered to calibrate ν_T field, while in Chapter 3 cases with several

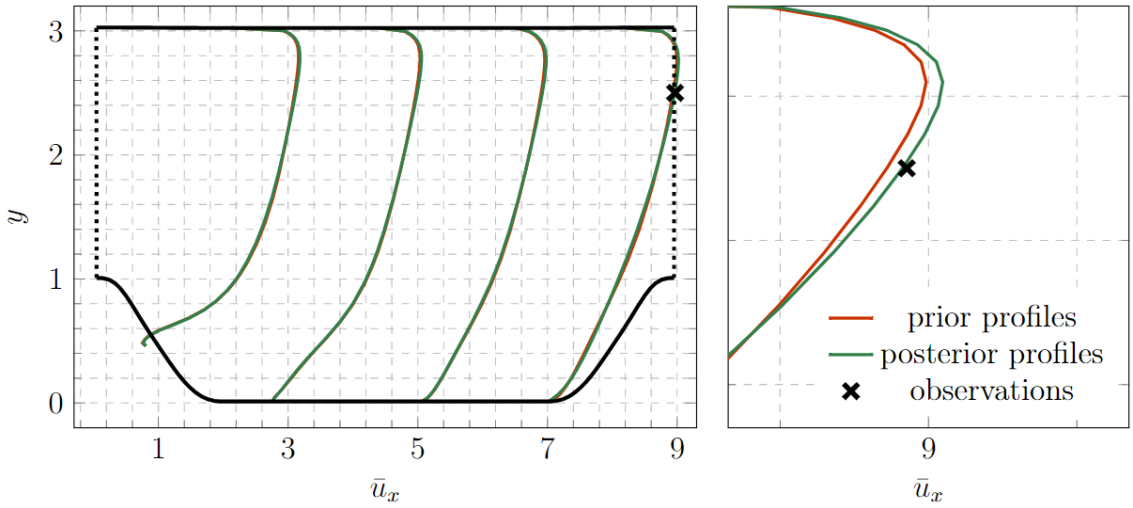

Figure 2.3: Profiles of longitudinal mean velocity field $\bar{u}_x$ for the PHC.
On the left side, profiles at different x are reported. On the right side, a focus around the only observation: after EnKF algorithm the profile approaches the measured value.

sparse observations have been faced.

The core of EnKF algorithm is implemented in `nutfoam.py`: it provides the connection between EnKF and OpenFOAM. At first, the flow case is initialized and a prior ensemble is sampled. Thanks to parallelized code, `nsamples` OpenFOAM simulations are run simultaneously exploiting *nutFoam* solver (Section 2.4), Kalman gain matrix K' and sample covariance matrix C'_{nu_T} are computed and distribution for ν_T is updated. DAFI itself provides methods to properly read and write files in OpenFOAM style, which have been sligthly modified to best fit our needs for this work.

In Table 2.3 some other parameters of this case are reported, then results are shown and briefly discussed.

EnKF prameters	
Nsamples	10
max iterations	10
nklmodes	200
obs position	[7.0, 2.5]

Table 2.3: EnKF parameters (cfr. [17]).

As can be seen in Figure 2.3, the inferred ν_T field leads to a better estimate of the measured data; it is obtained as the sample mean of all the ensemble at the last EnKF iteration. [17] also points out that the entire velocity field can be improved with a single observation. In Figure 2.4, the whole inferred ν_T field is plotted.

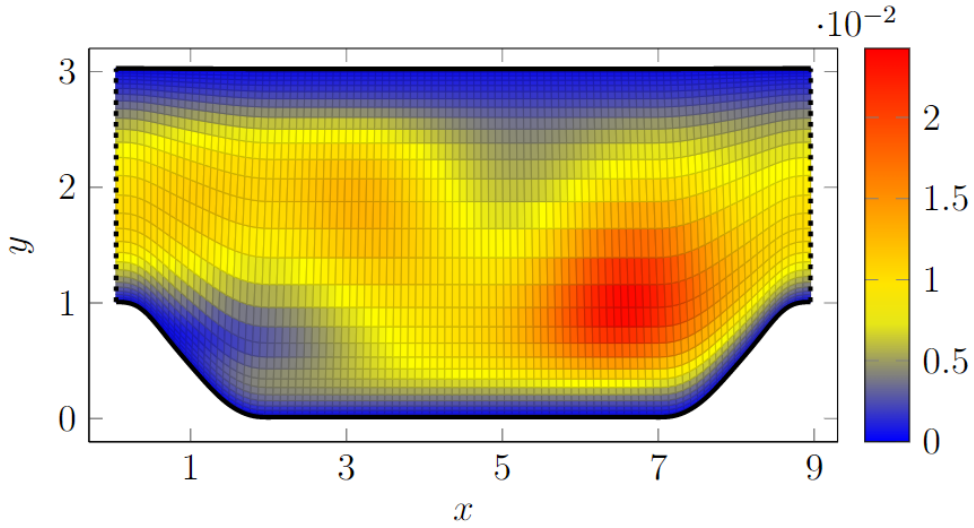

Figure 2.4: ν_T field resulting from the posterior sample mean of the inferred ensemble ν_T fields.

3 | Cases of study and results

In this chapter some cases of study and results are reported. In Section 3.1 the same paradigm presented in Section 2.4 is used to infer ν_T field in the case of a Turbulent Channel Flow (TCF), considering as high fidelity results DNS simulations from [3]. Then we try to extend the workflow considering k-ω model (Section 1.3) as turbulent model and trying to infer one of the coefficients included in it: in particular, β^* has been noted to be significant for such a study. In Section 3.2 EnKF algorithm has been slightly modified to give constant results over the domain instead of fields, while in Section 3.3 a new OpenFOAM turbulent model is proposed to allow the forward simulation to consider β^* as a field over the domain. In these last two cases, both Periodic Hills Channel Flow (PHC) and Turbulent Channel Flow (TCF) are studied.

We will refer to profiles obtained with prior distribution of the states as the 'baseline' and to profiles obtained with the inferred states as the 'corrected' profiles.

3.1. Inference on ν_T field for TCF

In Section 2.4 the case of field inversion for a PHC has been discussed, starting from [17]. In particular, ν_T field over the domain is inferred by mean of EnKF, without a turbulent model by mean of *nutSolver*. As first case of study, the same is done in this section for the TCF case presented in 1.5. In Table 3.1 and 3.2 the case settings about computational mesh and EnKF algorithm are listed, while flow properties are the same considered in Table 1.2.

For this specific case, the longitudinal correlation length l_x in the mixed periodic squared exponential kernel has been set to 5.0, much higher than the example presented in Section 2.4: this is aimed to reduce longitudinal variability of eigenfunctions when constructing the KL decomposition for the prior Gaussian Field for ν_T (cfr. Section 2.4), as expected for the geometry at hand. In figure 3.1, eigenvalues λ of the prior covariance matrix normalized by λ_{MAX} are reported: as can be noted, for KL truncation we can consider only the first 30 modes, which capture almost all the variance. In Figure 3.2 the first, fifth and

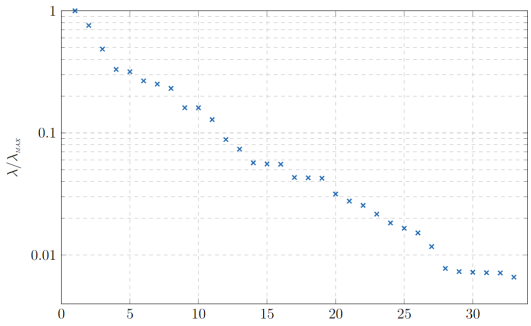

Figure 3.1: C_β normalized eigenvalues decay for TCF. Almost all the variance is captured by the first 30 modes.

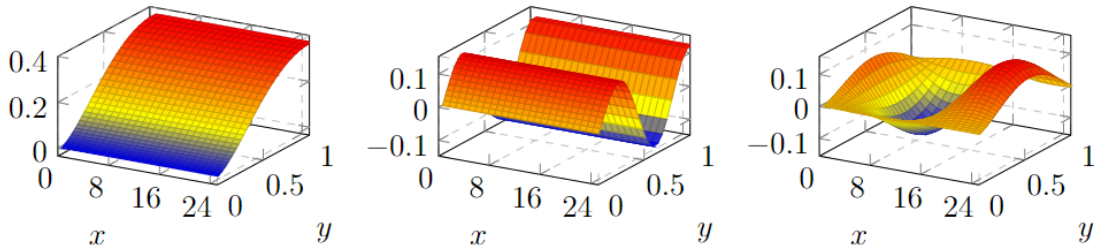

Figure 3.2: From left to right: first, fifth and ninth eigenfunctions (modes) for the considered covariance kernel. $\sigma^2 = 1.0$, $l_y = 0.25$ as suggested in [17], $l_x = 5.0$ to limit longitudinal fluctuations.

ninth considered modes are reported as an example.
As for observations, we rely on DNS results from [3], while k-ω model with usual constant values is used to get the prior mean field of ν_T. Finally, Figure 3.3 motivates the choice to perform 4000 *nutFoam* iterations for each sample of the ensemble for any EnKF update.

Our prior mean for ν_T has been obtained through a k-ω simulation, and in Figure 3.4 the resulting $\bar{u}_x$ profile is reported in red. This profile approximates quite well the DNS observations (black crosses) but still suffers from inaccuracy. The green profile is obtained performing a simulation which relies, for ν_T, on the sample mean of all posterior ν_T fields inferred by EnKF from the observations. As can be seen, this profile better fits not only

domain dimensions [m]		**mesh refinement**		**domain BC**	
L	8π	$\mathbf{N_x}$	60	**inlet**	cyclic
h	1.0	$\mathbf{N_y}$	30	**outlet**	cyclic
d	0.1	$\mathbf{N_z}$	1	**base wall**	no slip
				central line	symmetry line

Table 3.1: Domain properties and boundary conditions (cfr: [3]). $N_z = 1$ makes the case bidimensional.

EnKF parameters	
$N_{samples}$	15
EnKF iter	10
N_{modes}	30
obs positions	[12.5, 0.156] [12.5, 0.482] [12.5, 0.928]

Table 3.2: EnKF parameters.

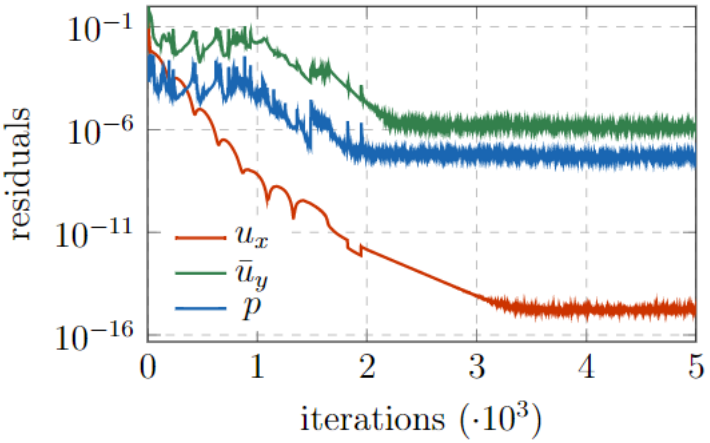

Figure 3.3: Residuals for $\bar{u}_x$, $\bar{u}_y$ and p in *nutFoam* solver for TCF.

the observations used to correct the model (Table 3.2 and Figure 3.4 left) but approaches all the DNS results: just few observations correct the profile even in other locations. To quantify the improvement, the L^2 distance between observation and profiles has been computed: it results that the profile obtained with the inferred ν_T field reduces error from 0.122 to 0.078, thus almost 40% less than the error of the baseline profile obtained with standard constant for k-ω model.

The resulting inferred ν_T field is plotted in Figure 3.5 and is coherent with ν_T field obtained with standard k-ω. However, comparing what we get with ν_T field of a highly refined k-ω simulation, we cannot appreciate any improvement than the baseline: this drawback is due to high non linearity of the problem and is pointed out also in [17] and [24], where it is shown that corrections are relevant only for observed field. This could be alleviated by including observations also from other quantities, as done in [24].

Finally, we can conclude the discussion by noting that the periodic, horizontal geometry of the domain let us expect results to vary mostly on the transversal direction than the longitudinal one. This, actually, happens for $\bar{u}_x$ and ν_T field, for example (cfr. Figure 1.4): thus we tried to approximate the inferred ν_T field by averaging it over x. In Figure

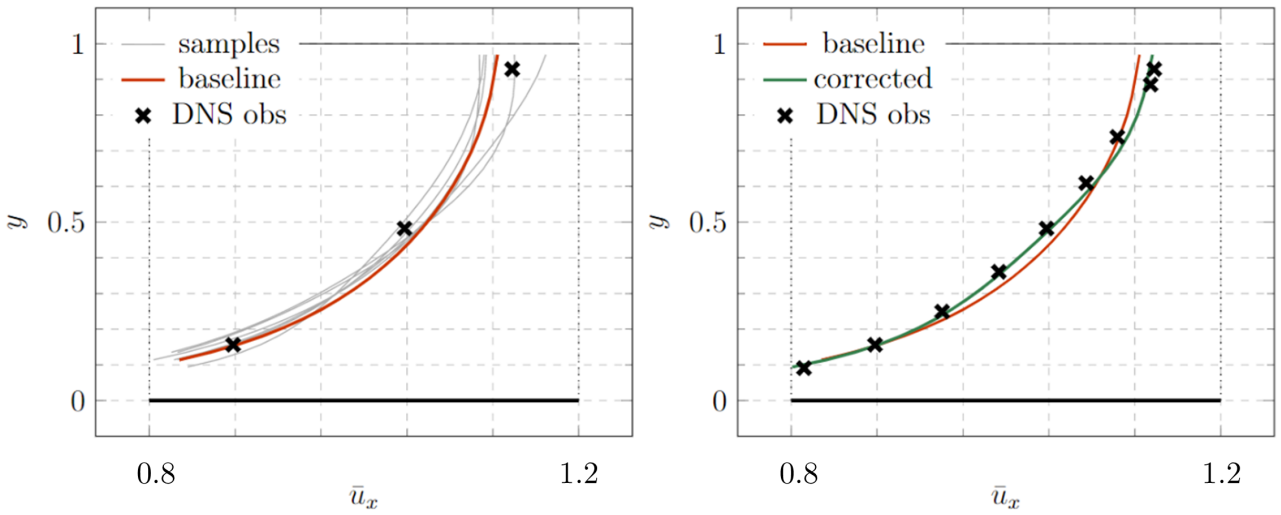

Figure 3.4: On the left: $\bar{u}_x$ baseline profile for $\bar{u}_x$ obtained with k-ω model. The underlying ν_T field represents the prior mean for samples of EnKF ensemble. Here, only 6 ensemble profiles are plotted, for clarity, taken at the first iteration. Marked observations are those effectively used to correct the state.
On the right: corrected profile obtained with inferred ν_T is compared against all the DNS observations and baseline profile. All DNS observations are reported, not only those used for state EnKF correction.

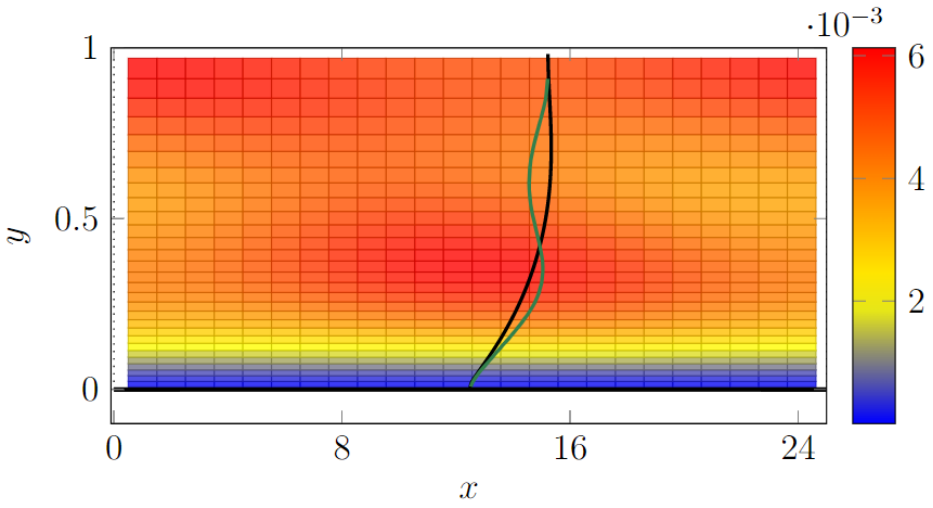

Figure 3.5: ν_T field resulting from the posterior sample mean of the inferred ensemble ν_T fields for TCF. Moreover, high refined ν_T profile (black) and mean along x axis of inferred ν_T (green, ν_T^{AVG}) are compared, both scaled to highlight the differences.

3.5, $\nu_T^{AVG}(y) := \frac{1}{N_x}\sum_{i=1}^{N_x}\nu_T(x_i, y)$ has been reported, in green. A simulation performed with ν_T^{AVG} instead of the properly inferred ν_T shows negligible worsening, as the L^2-norm error results to be 0.0867, 1.1 times the error measured in the proper case.

3.2. Inference on k-ω model constants

As presented in Introduction, the goal of this work is to apply the presented workflow to infer coefficients involved in Eddy Viscosity turbulence models (Section 1.3). These values

are usually calibrated on simple benchmark cases. In the following, the EnKF algorithm is slightly modified in order to infer a constant value for the coefficients, instead of a field. Indeed, natural setting of Kalman Filter as implemented in DAFI allows to consider only fields over the whole domain to be inferred and not constant values. On the opposite, standard formulation of Eddy Viscosity models takes as input constant coefficients, and not fields. For this reason, as explained in Algorithm 3.1, at each iteration, for each sample of the ensemble, the mean of the correction step output is given as input for the next prediction step. In this way, the implemented EnKF algorithm is not modified in its correction step, but OpenFOAM receives as input a constant scalar as coeffcient.

Algorithm 3.1 EnKF algorithm, modified to return a constant corrected state.

1: Sample N ensembles $\boldsymbol{\beta}_j \sim p(\boldsymbol{\beta})$: prior ensemble
2: Build matrix $B = [\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_N] \in \mathbb{R}^{n \times N}$
3: Build matrix $D = [\boldsymbol{d}_1, ..., \boldsymbol{d}_N] \in \mathbb{R}^{n \times N}$ of observations $(\boldsymbol{d}_j = \boldsymbol{d} + \boldsymbol{\xi}_j, \boldsymbol{\xi}_j \sim \mathcal{N}(0, C_m))$
4: **while** convergence criterion not fulfilled **do**
5: Compute matrix C'_β, the sample covariance of columns of B
6: Compute matrix $K' = C'_\beta H^T (H C'_\beta H^T + R)^{-1}$
7: For each column B_j compute the sample mean $\hat{\mu}_j^B$
8: Evaluate the model for each column of B : $Z = H\hat{\mu}_j^B$
9: Update the ensemble matrix B : $\hat{B} = B + K'(D - Z)$
10: **end while**

Adding this constraint, we can expect we are decreasing the efficiency of the original algorithm: indeed, we are reducing an information described by a field over the whole domain just to a single constant value. This will result in an higher number on EnKF iterations needed to get the final inferred result: a possible measure of the degree of convergence reached by the algorithm could be the sample variance of the ensemble.

This approach has been tested on several constant coefficients involved in k-ω model: some considerations are listed below and results can be compared in Figure 3.6. In particular, an increase of β^* results in a reduction of mean longitudinal velocity in the region closest to wall and in an increased velocity in the free stream: thus, in the case of TCF, the maximum value for $\bar{u}_x$ reaches higher values than the case of low β^* while the profile is under estimated near the wall (Figure 3.6, left). For high values of β^*, k equation would reduce to

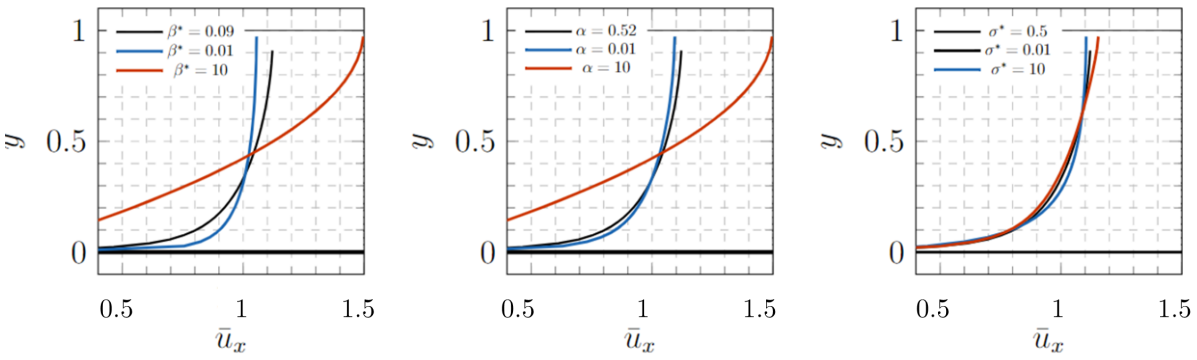

Figure 3.6: Comparisons of different values of β^*,α and σ^* for k-ω model in TCF.

$$\beta^* k\omega \approx 0 \ , \ \text{on } \Omega \tag{3.1}$$

and numerically has been seen that it implies ν_T to vanish. Actually, profile obtained for high β^* approaches the laminar case, with its typical parabolic shape and with the maximum longitudinal velocity at the center line $u_{x,0} = \frac{3}{2}U_b$.

A similar behaviour, but less significant, has been registered also for α (Figure 3.6, centre). Instead, an independence region has been found with respect to σ^*: even choosing significantly different values, the resulting differences are negligible (Figure 3.6, right). For usual values of σ^*, the resulting k slightly oscillates around its initial value: we observed numerically that this behaviour is slightly enhanced by low value of σ^*, while for high values k tends to flatten out to a constant value.

The main weakness of this method is that we are trying to find a constant value all over the domain, thus EnKF state correction should conciliate regions presenting different $\bar{u}_x$ behaviour, in particular for PHC case, where we can distinguish not only close/far from the wall, but also before/after a hill. This aspect does not allow to consider many points for the calibration, and also the considered points should belong to similar region in terms of mean longitudinal velocity field. When attempting to join together a great number of observations from different region, the computation of Kalman Gain matrix K' in Algorithm 2.1 suffers by ill-conditioning and the algorithm stops because of the singularity of matrix $HC'_\beta H^T + R$.

In the next sections, we focus our analysis on parameter β^* in the cases of TCF and PHC.

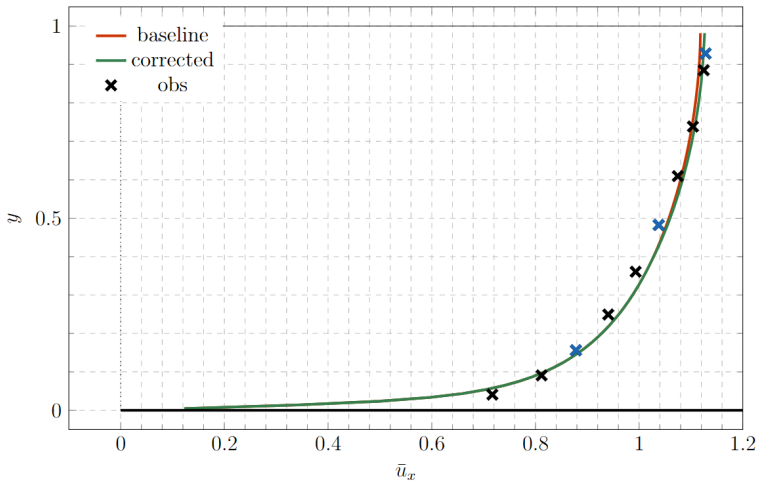

Figure 3.7: Profiles comparisons for $\bar{u}_x$. As expected, inferring a constant over the whole domain leads to minimal improvement of the corrected profile. EnKF algorithm inferred $\beta^* = 0.0927$.
In blue are the observations actually used for EnKF, in black other DNS data from [17].

3.2.1. β^* constant for TCF

At first, the idea is applied to TCF case, with the same characteristics listed in Table 3.1. Again, $N_{samples} = 15$ and $N_{modes} = 30$, while the number of EnKF iterations has been increased to 25 as said in Section 3.2. Table 3.3 summurizes this informations, together with observations location, which is not modified. Also parameters for KL decomposition are the same discussed in Section 3.1.

Since TCF is one of the benchmark cases on which constants are calibrated, we expect to find a confirmation of the value of β^* rather than a significant correction. Actually, sample mean of the inferred β^* values results to be 0.0927, with a negligible sample covariance around this value of the order of 10^{-6}. Corrections on the resulting profile are not evident, and the corrected posterior profile does not approach observations so much: computed L^2 error with respect to DNS observations is reduced from 0.122 to 0.116, only of 5%. As noticed in Section 3.1, other profiles do not exhibit significant improvements, as shown in Figure 3.8. However, the slight correction of mean longitudinal velocity profile

EnKF parameters	
$\boldsymbol{N_{samples}}$	15
EnKF iter	25
$\boldsymbol{N_{modes}}$	30
obs positions	[12.5, 0.156] [12.5, 0.482] [12.5, 0.928]

Table 3.3: EnKF parameters fot TCF.

leads to a correction of the measured shear stress at the wall $\tau_w := \tau_{xy}\,|_{y=0}$: precisely, OpenFOAM *wallShearStress* function computes $\frac{1}{\rho}\tau_{TOT}$ ([6]) which can be approximated to $\frac{1}{\rho}\bar{\tau}_{xy}$ near the wall (cfr. Figure 1.2) thus resulting in the squared friction velocity u_τ^2 (cfr. Section 1.3.2), and it goes from 0.0306165 m^2/s^2 of the baseline to 0.002896 m^2/s^2 of the corrected profile, against a measure of 0.00276551 m^2/s^2 of the highly refined simulation. Its behaviour inside the whole half channel is reported in Figure 3.8

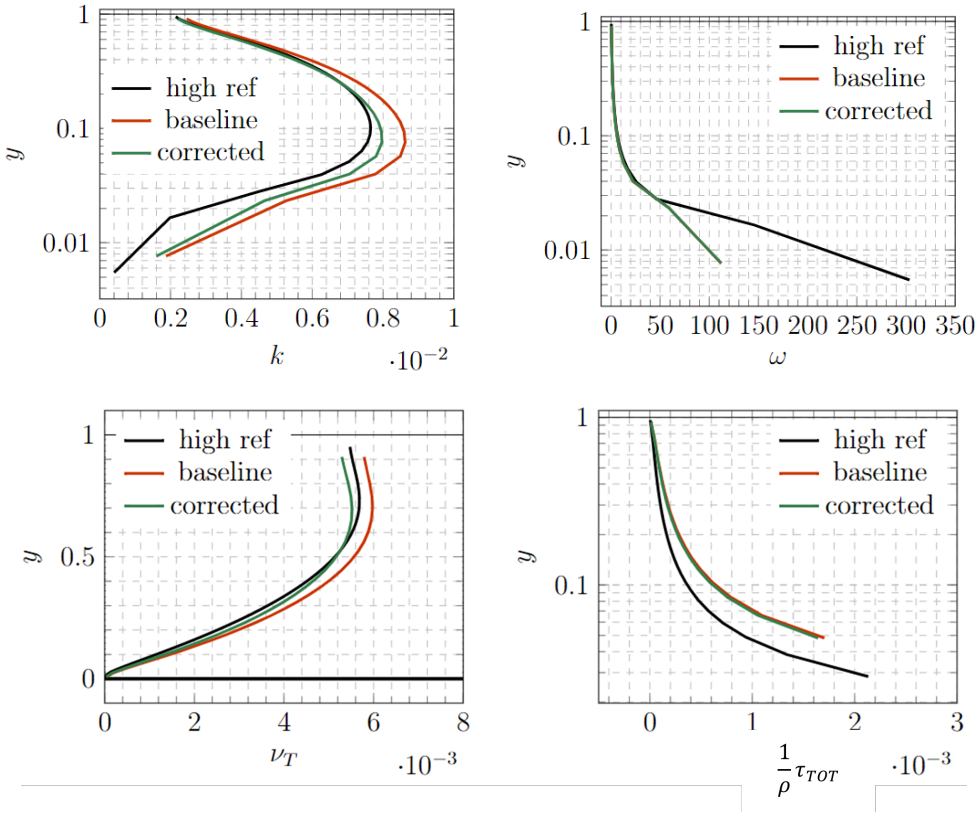

Figure 3.8: Profiles comparisons for baseline, corrected and high refined k, ω, ν_T, τ_w. Since [3] provides only $\bar{u}_x$ data, k,ω,ν_T and τ_w are extrapolated from an highly refined simulation ($N_y = 100$). k, ω and $\frac{1}{\rho}\tau_{TOT}$ profiles are in logarithmic scale to better highlight differences.

3.2.2. β^* constant for PHC

Let us consider now the PHC case, with the same geometry and flow properties of Tables 2.1 and 2.2. The only exception is related to N_y, which has been reduced because of computational reasons and also to have a reduced accuracy in the baseline prior profile. Applying EnKF analysis in this framework will return a correction constant that provides more precise results still relying on a coarse mesh: we can interpret the inferred result as a correction to account for in order to tackle inaccuracies introduced by the coarsening of the mesh. Observations are obtained from an highly refined k-ω simulation.

baseline mesh		highly ref mesh	
$\mathbf{N_x}$	100	$\mathbf{N_x}$	100
$\mathbf{N_y}$	15	$\mathbf{N_y}$	70
$\mathbf{N_z}$	1	$\mathbf{N_z}$	1

Table 3.4: Mesh refinement for PHC.

As hinted above, we expect the algorithm to suffer different behaviour of the longitudinal mean velocity profile over the domain. Actually, a first attempt with sparse observations failed: a too high number of observations, in the order of 10, leads to singularity of matrix $HC'_\beta H^T + R$ which can not be inverted in the computation of the Kalman Gain K', while a reduced number of observations has a negligible correction of β^* as a consequence. For this reason, we distinguished three different zones inside the domain where perform the analysis separately: in particular, we run EnKF with observations only in the central line of the domain, another with observations only in the region just before the hill and the last with observations gathered immediately after the hill. Table 3.5 summarizes observations location. Then the expected result is an inferred constant fitting the specific region on which it has been calibrated.

observations		β^*
post hill	[1,0.6] [2,0.1] [2,0.6] [3,0.3]	0.168321
central line	at $x = 5$, 8 obs uniformly distributed	0.138526
pre hill	[6,0.3] [7,0.1] [7,0.6] [8,0.6]	0.098404

Table 3.5: Observations for each region of PHC.

EnKF parameters	
$\boldsymbol{N}_{samples}$	10
EnKF iter	25
$\boldsymbol{N}_{modes}$	110

Table 3.6: EnKF parameters for PHC.

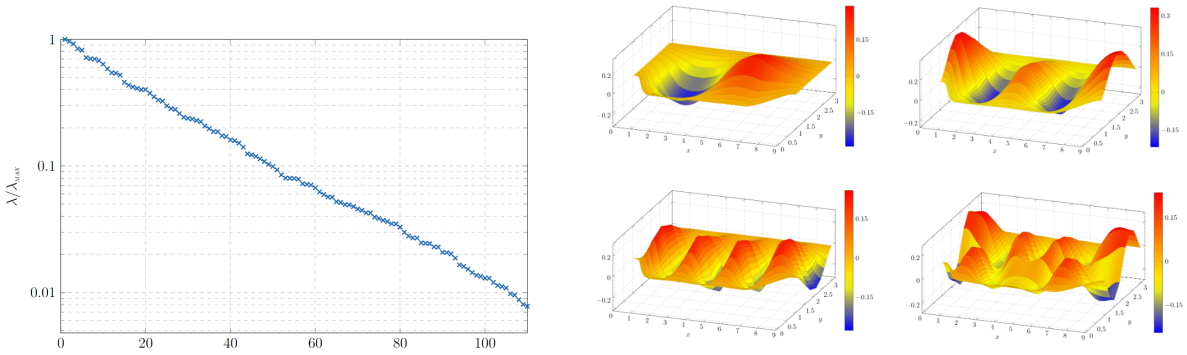

Figure 3.9: C_β normalized eigenvalues decay. Almost all the variance is captured by the first 110 modes.

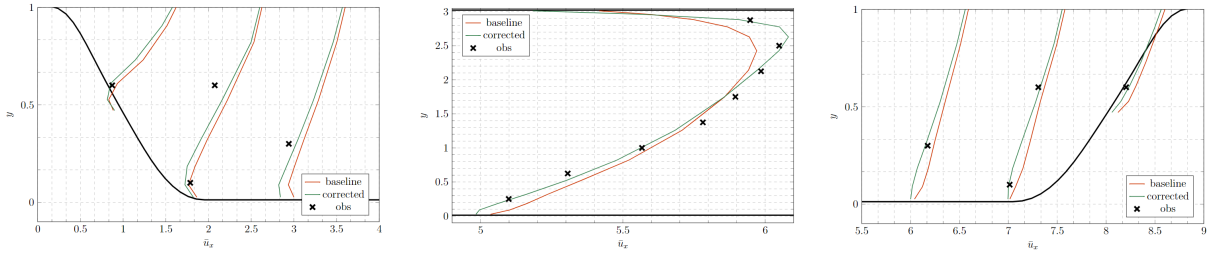

Figure 3.10: Profiles based on the respective inferred β^* against the baseline, in different region of the domain: after a hill (left), central region (centre), before a hill (right).

As for EnKF parameters (Table 3.6), for computational reasons the ensemble is composed of 10 sample, 25 iterations are run and first 110 modes are included, as motivated in Figure 3.9.

EnKF algortihm infers $\beta^* = 0.138526$ for the central line and $\beta^* = 0.168321$ after an hill: following what is discussed in Figure 3.6, we could expect such a result, since the coarser baseline we took as starting point underestimates longitudinal velocity profile far from the wall and overestimates it close to the wall. This misfit decreases in the third considered region, before the hill, and the inferred value is $\beta^* = 0.098404$. These values are obtained as the mean of the values returned by each sample of the ensemble, which show a variance of the order of 10^{-3}. In Figure 3.10 a comparison among profiles is shown: we estimate a reduction of about 15% of L^2 error between each corrected mean longitudinal velocity field and the highly refined case.

Finally, we run a simulation on the same coarse mesh of baseline but assigning respective inferred values of β^* at each zone: this domain subdivision has been performed qualitatively (Figure 3.12, above left). Above right, resulting values of mean longitudinal velocity is plotted, showing the recirculation region just past the hill.

In this case, L^2 error decreases of about 20% over the whole domain. In any case, as said in Section 3.1, we do not observe any significant correction for other fields, such as k, ω and ν_T. On the contrary, being τ_w directly dependent on $\bar{u}_x$ ($\tau_w \propto \frac{\partial}{\partial x_\perp}\bar{u}_\parallel$, with $x_\perp$ normal direction and $\bar{u}_\parallel$ parallel velocity component to the wall), we can expect a consequent improvement. The improvement can actually be observed, in particular at the upper wall (Figure 3.11), probably as a consequence of the higher correction of $\bar{u}_x$.

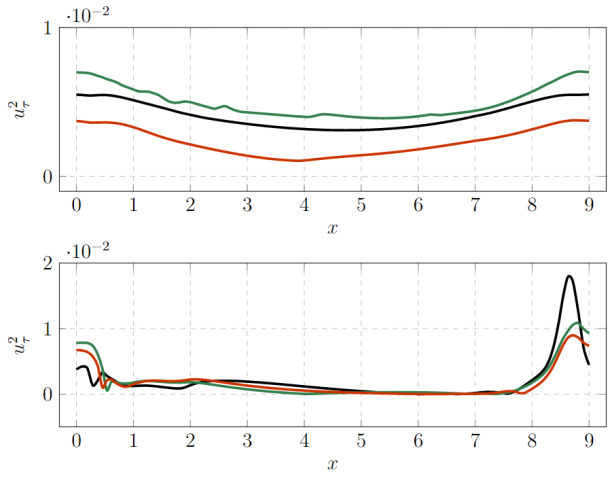

Figure 3.11: u_τ^2 profiles compared at upper (above) and lower (below) wall.

We conclude with a comparison among prior (Figure 3.12 below left), corrected (Figure 3.12 above right) and high refinement (Figure 3.12 below right) mean longitudinal velocity field. As can be seen, recirculation region is reduced in the prior field and not completely corrected by EnKF: this could be, probably, better investigated by taking more observations in that region. Instead, we have a correction of the magnitude in the upper part of the domain, coherently with Figure 3.10.

3.3. Extension to k-ω model coefficients varying over the domain

In previous examples, we preserved the idea of providing constant input coefficients to our turbulence model, according to the canonical formulation. This forced to sacrifice the EnKF inferred field at each iteration, reducing it to a constant value, specifically the sample mean of the field. Even if we get some improvements, this workflow suffers from slow convergence and poor final corrections.
Things got better when we increased the number of degrees of freedom for our coefficients:

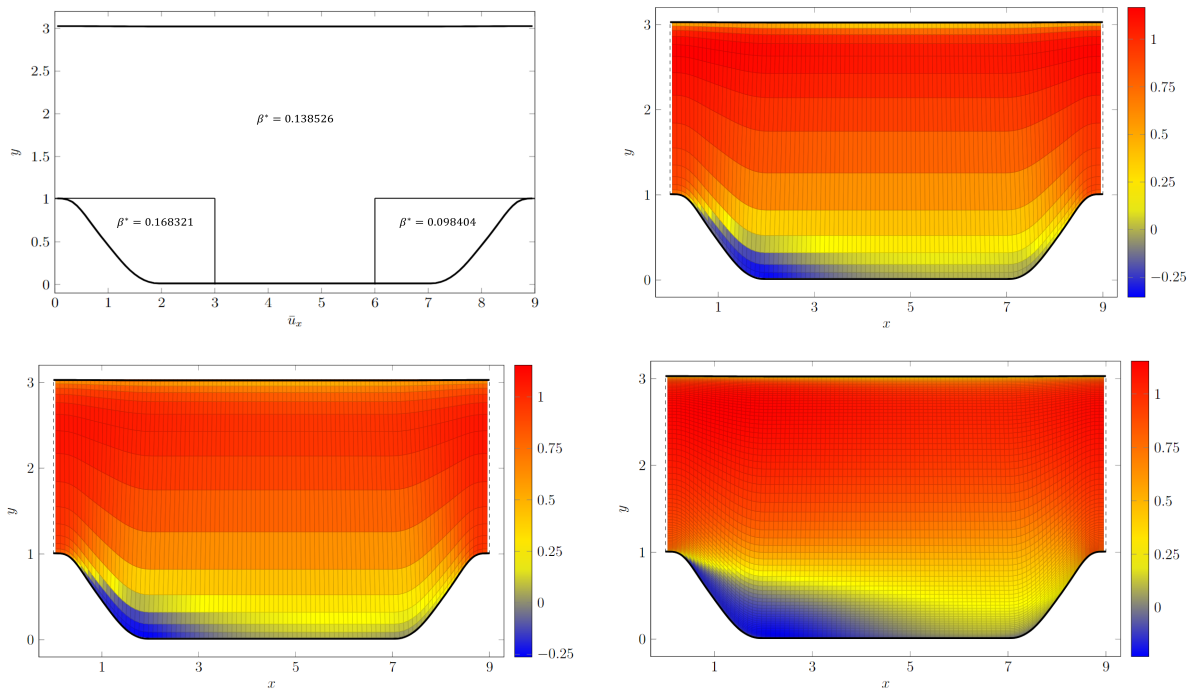

Figure 3.12: Above: subdivision of the domain according to zones described (left) and $\bar{u}_x$ field resulting when associated inferred β^* values have been used (right).
Below: prior $\bar{u}_x$ field (left) and high refinement $\bar{u}_x$ field (right).

more accurate results are found when allowing coefficients to be piecewise constant, as proposed in Section 3.2.2. Now we extend our focus to the case in which coefficients can be considered as a field, varying all over the domain. The only constraint, due to code structure, is value at the wall, which must be specified as boundary condition for the field: since k-ω performs quite well here, we decided to keep standard values at the wall as proposed by [22] and discussed in Section 1.3.3.

In its original formulation, OpenFOAM provides classical version of k-ω model, where all coefficients are meant as constant. To adapt it to our purposes, a new model has been implemented allowing to work with fields: we will refer to it as k-ωField model and it relies directly on k-ω model original implementation. Its implementation is listed in Appendix A.

This approach, again, has been applied both to TCF and PHC, and results are discussed below.

3.3.1. β^* field for TCF

For β^* optimization on TCF we used same EnKF data as Table 3.2, domain properties as Table 3.1 and flow properties as Table 1.2. The inferred $\beta^*_{AVG}(y) \coloneqq \frac{1}{N_x}\sum_{i=1}^{N_x}\beta^*(x_i, y)$ field is plotted in Figure 3.13, again as the longitudinal mean as described in Section 3.1. We can notice that inferred β^* values oscillate around default value 0.09. In the closest region to the wall, up to $y \sim 0.15$, baseline underestimates the real profile, thus according to what we pointed out in Figure 3.6 β^* should be decreased to better fit observations. The opposite behaviour is expected up to the a half of the domain, where the overestimate is reduced by increasing β^*. Then, for $0.45 \lesssim y \lesssim 0.75$ baseline still overestimate real observations, but Figure 3.6 suggests a reduction of the coefficient. Finally, in the central zone a $\beta^* > 0.09$ best fits observations. Therefore, Figure 3.13 is quite coherent with our expectations. Moreover, sample average of this inferred field is 0.095, not far from values suggested in Section 3.2.1 for the same case.

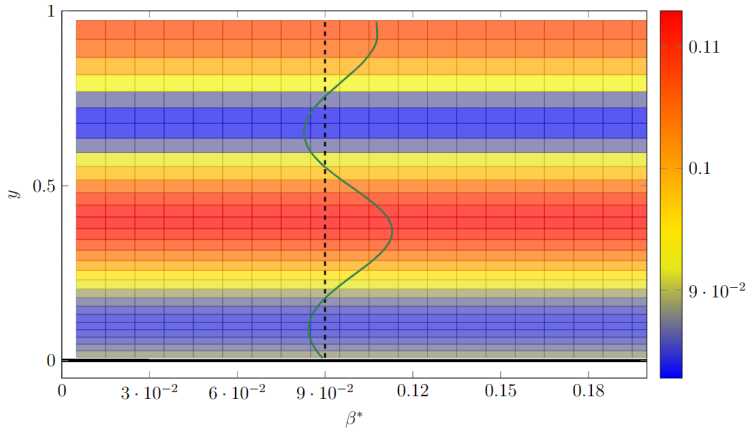

Figure 3.13: β^* field for TCF optimised as a field over the domain. Inferred profile is compared against default constant value 0.09.

Comparison of $\bar{u}_x$ obtained profiles is plotted in Figure 3.14, where an improvement with respect to all DNS observations can be appreciated, not only with respect to those actually used for EnKF correction. Moreover, with respect to Figure 3.7, we can observe a better fit also in the central region of the half channel.

This correction reduces L^2 error with observations of 25% with respect to the prior baseline, from 0.122 to 0.091. Instead, no remarkable improvement can be observed for other quantities k, ω, ν_T and τ_w. For this last quantity, absence of significant improvement could be motivated by observing position of observations, which do not approach enough the wall.

β^* field resulting in Figure 3.13 suggested us a further step, following the same idea

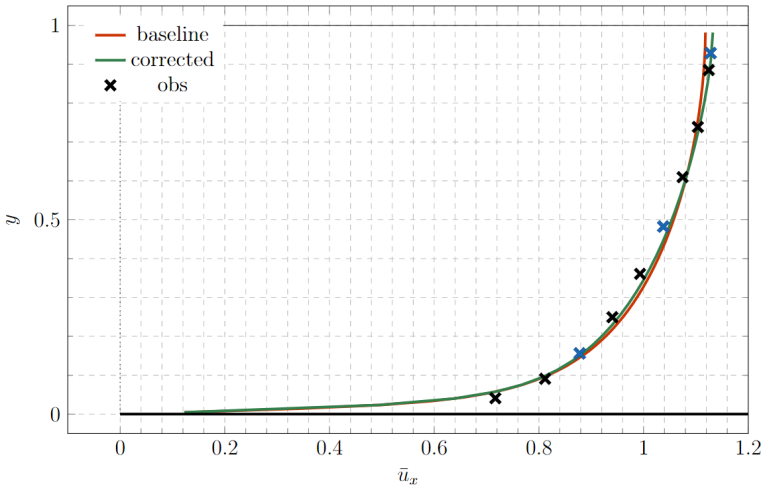

Figure 3.14: $\bar{u}_x$ profile obtained with optimised β^* as a field over the domain. In blue, measurements actually used as observations in EnKF.

domain division	β^*
$0 \lesssim y \lesssim 0.15$	0.084281346
$0.15 \lesssim y \lesssim 0.45$	0.097918427
$0.45 \lesssim y \lesssim 0.75$	0.084358754
$0.75 \lesssim y \lesssim 1.0$	0.099132271

Table 3.7: TCF domain division and zonal constant coefficients β^*

proposed for PHC in Section 3.2.1: we divided the domain in four longitudinal stripes and for each zone respective β^* average was proposed as zonal constant. Qualitatively, regions have been delimited by values discussed above and summarized in Table 3.7 and Figure 3.15 left, together with coefficient value for each.

Figure 3.15 right shows a slight worsening of corrected profile with respect to that obtained in Figure 3.14: this could be due to a loss of information on β^* when flattening its value at its mean, in particular in the central stripes, where inferred β^* field suggests the largest departure from base value 0.09. However, also in this case L^2 error is reduced, even if only of 11%, to 0.109, with respect to baseline, as a consequence of loss of correction in the central region of the half channel. As for TCF in Section 3.2.1, no interesting improvements are registered for k, ω, ν_T and τ_w.

3.3.2. β^* field for PHC

We conclude our cases applying optimization of β^* as a field in the case of PHC.
Higher flexibility of this method allows us to exploit a greater number of observations

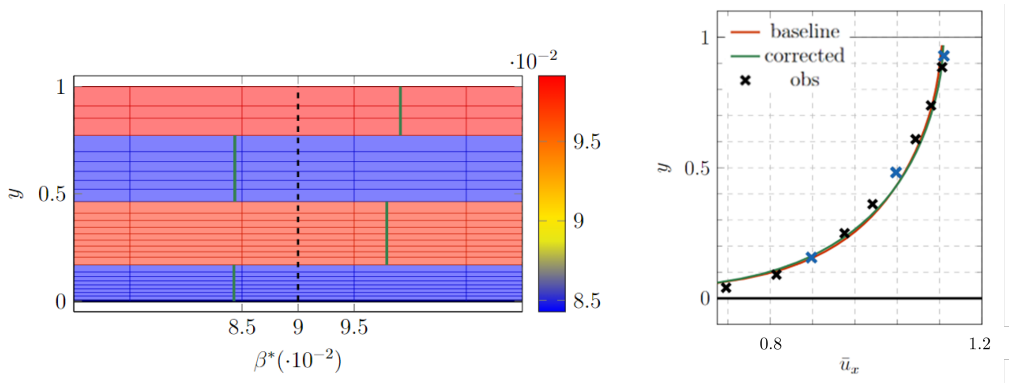

Figure 3.15: On the left: β^*_{AVG} field and β^*_{AVG} for each deifferent region. On the right: $\bar{u}_x$ profiles resulting from β^*_{AVG}. In blue, measurements actually used as observations in EnKF, in black other DNS observations.

and to distribute them all over the domain: in this case, 25 observations are randomly choosen, with the only constraint of integer abscissa. Sparsity of observations accounts for heterogeneity of different regions at once, differently from case in Section 3.2.2, where different regions were analyzed in different runs. This requires an increased number of EnKF iterations, specifically 20 are performed. In Table 3.8 all EnKF data are summarized. As for mesh refinement, because of instabilities occurred for too coarse mesh, setting presented in Table 2.1 has been choosen for baseline and EnKF runs, while mesh in Table 3.4 has been used as higly refined mesh.
Starting from a more accurate baseline than Section 3.2.2, we expect a reduced rate of correction.

EnKF parameters	
$\boldsymbol{N}_{samples}$	15
EnKF iter	20
$\boldsymbol{N}_{modes}$	200

Table 3.8: EnKF parameters for PHC.

Below, results are plotted: Figure 3.17 left shows that β^* has strong variability all over the domain. However, the average value of the inferred field is greater than 0.09 as suggested also by results of β^* as a constant in Section 3.2.2, and this trend is preserved also in each domain subregion: the most remarkable departure from standard β^* is registered by the average value of β^* field in the central line and in the region just after the hill, while a slightly lower increase is registered in the average of β^* field just before the hill. Differently from case in Section 3.2.2, higher value is inferred, on average, at central line than after the hill.
In Figure 3.17 right the resulting mean longitudinal velocity field is plotted. L^2 error is

actually reduced, but with a small rate, 9% as motivated above.
Finally, posterior fields for k, ω, and ν_T are compared with high refinement results in Figure 3.18: again, general behaviour of these fields is preserved but not improved and magnitudes are comparable, except for ω which is underestimated at the boundary.

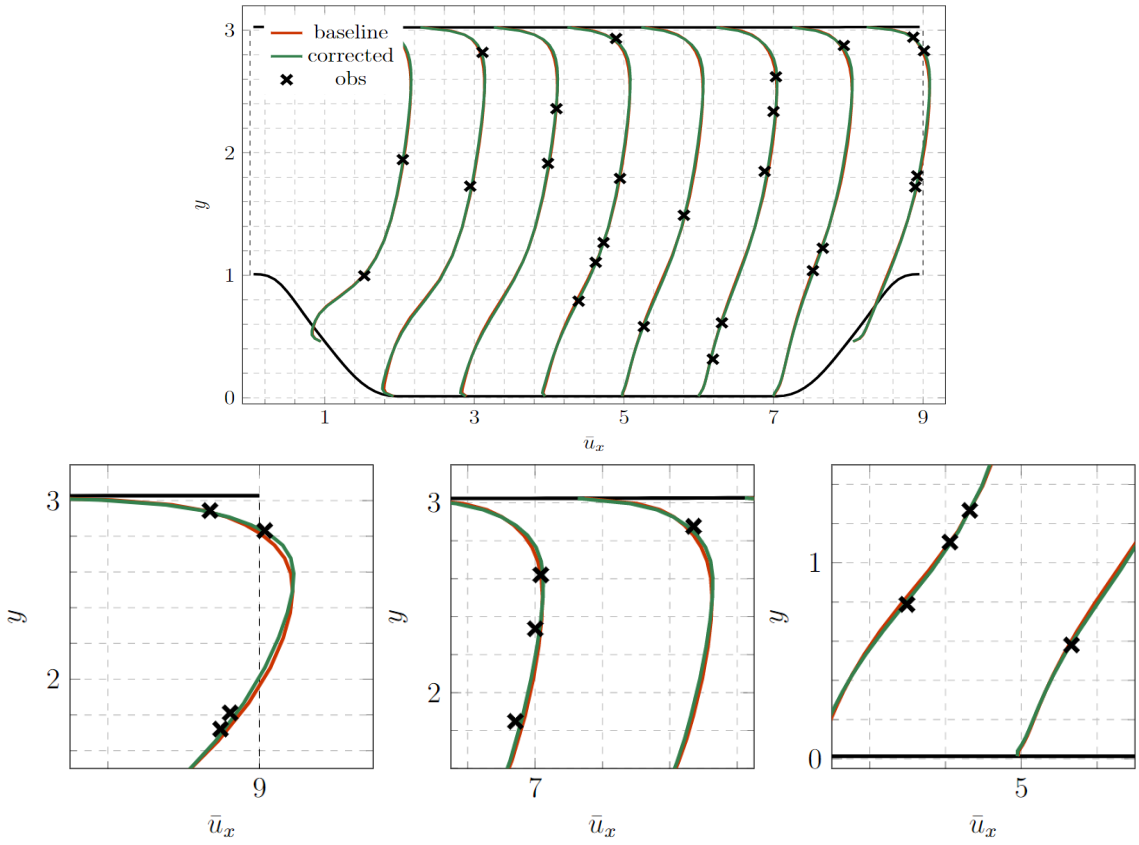

Figure 3.16: Above: $\bar{u}_x$ inferred profiles for each observations line.
Below: focus on some observations from different zones of the domain, as examples.

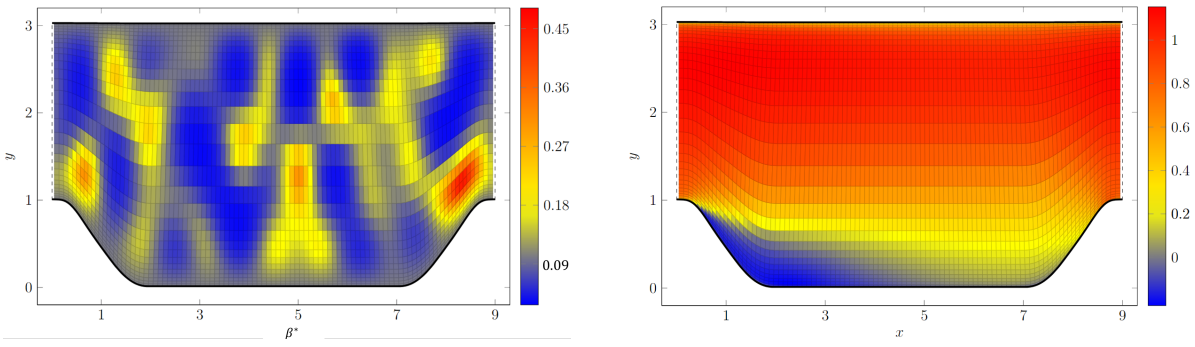

Figure 3.17: Left: β^* inferred field. Right: consequent $\bar{u}_x$ field.

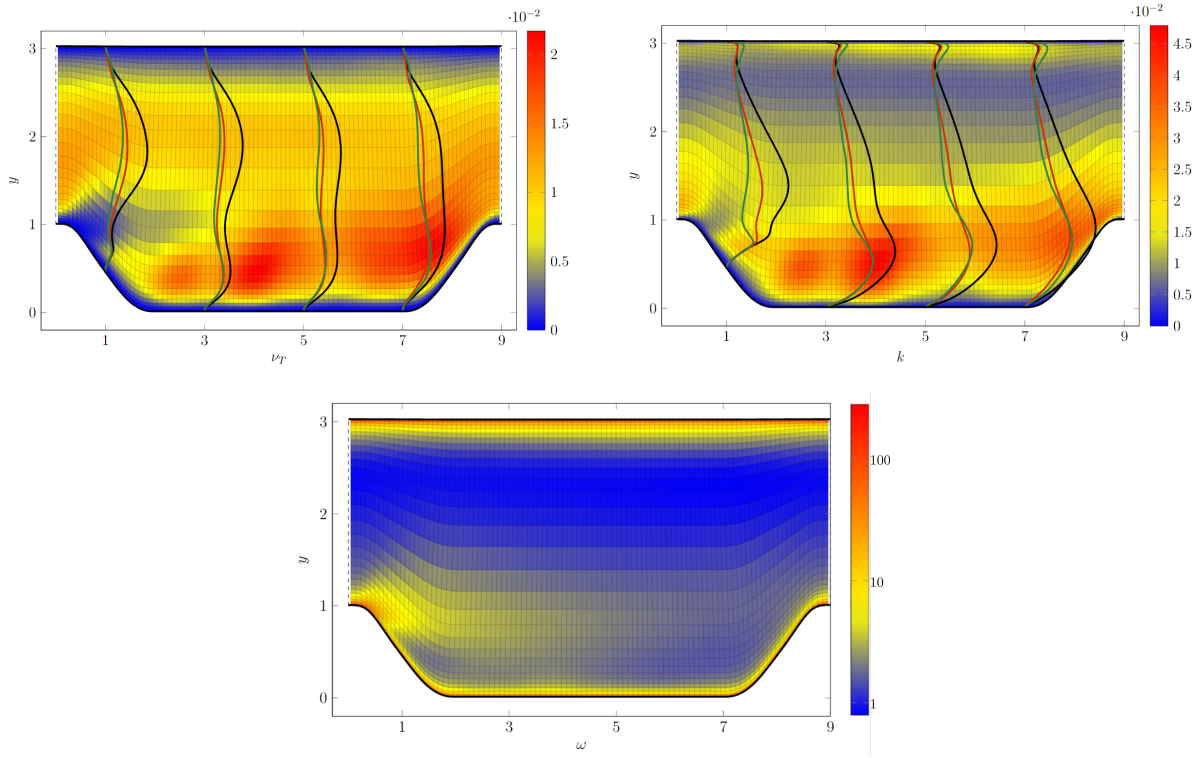

Figure 3.18: ν_T, k and ω posterior field and respective profiles in $x = 1, x = 3, x = 5, x = 7$: high refinement (black), baseline (red), corrected (green).
ω is in log-scale for clarity.

4 | Conclusions and further developments

In this work we analysed applications of EnKF algorithm to the optimisation of Eddy Viscosity Turbulence model for RANS equations, with particular attention to k-ω model: through Field Inversion techniques, we corrected mean longitudinal velocity field for turbulent flow cases. Kalman Filter has been preferred to gradient based techniques for its non intrusiveness and specifically Ensemble Kalman Filter (EnKF) has been used to reduce computational burden linked to evaluation of covariance matrices in the algorithm. We relied on DAFI library for EnKF, coupled with OpenFOAM (release: 9) for forward model solutions. Three different approaches have been proposed for Turbulent Channel Flow (TCF) and Periodic Hills Channel Flow (PHC) cases.

First, following [17] proposal, we considered ν_T as state to be inferred and directly provided to RANS equation as input: this has been possible exploiting *nutFoam* solver already implemented in DAFI. Remarkable corrections have been obtained, both for TCF and for PHC. Moreover, a second analysis has been done for the former case: considering geometry configuration, a simulation considering $\nu_T^{AVG}(y) := \frac{1}{N_x}\sum_{i=1}^{N_x}\nu_T(x_i, y)$ field has been run, leading to small worsening of mean longitudinal velocity field.

Then, we focus on model coefficients for k-ω model, with specific focus on β^*, and we tried to correct it preserving its constant nature: DAFI allows only field optimisations, thus we proposed a modification of pure EnKF algorithm, where sample mean of inferred state is considered inside the model at each iteration. Loosing accuracy by mean of this approximation, a slower convergence has been obtained, as well as negligible corrections in resulting velocity profile, in particular for TCF. This result could be expected, being TCF one of the benchmark cases used for model constant calibration: inferred values similar to default values confirm applicability of the method. Something different is obtained for the more complex case of PHC, where trying to infer a unique constant all over the domain failed, because of different nature of turbulence for different zones of the domain. For this reason we subdivided it in three characteristic zones, and a constant has been inferred

for each of them. Moreover, for computational reasons, a coarse grid has been used for prior profiles and for runs in EnKF iterations: then the results can be interpreted as a correction on β^* to tackle inaccuracy introduced by non refined mesh. As a drawback, this method suffers from instability when too many observations are included: in our simulations, no more than 8 observations have been considered.

Finally, to increase degrees of freedom for the optimisation, coefficient as scalar field over the domain have been considered: to do this, the new OpenFOAM k-ωField model has been implemented. This gave us a greater flexibility in term of number and locations of observations: if for TCF aligned measurements have been exploited again, for PHC the study included randomly placed observations. This last technique represents a middle ground between directly inferring ν_T doing without a turbulence model and inferring a coefficient constraint to be constant. As a consequence, in TCF case error correction stands between results obtained with the other two techniques, while this comparison can not be done with PHC since different mesh and observations have been considered.

A confrontation among all other quantities k, ω and ν_T shows how these fields are not subject to an interesting correction: this has been already found also by [17] and [24]. Only τ_w undergoes a small improvement in case β^* is inferred as a constant.

In this work only β^* has been considered as state to be optimized but attention could be generalized to other coefficients. Moreover, current version of DAFI allows to optimize coefficients one by one, while the workflow could be extended to simultaneous optimization of many values.
The study of the dependence of β^* and other values on mesh refinements can be deepened, thus allowing accurate results for low cost simulations. Finally, here, when dealing with constant coefficients, EnKF inferred scalar fields which were then forced to mean value to fulfill OpenFOAM: a library could be implemented to return an inferred constant value in a more natural way. Also, dealing with piecewise constant values, a methodology can be introduced to algorithmically establish subregions with different values of coefficients, which in this work have been defined only manually and qualitatively. The final goal is to automatically subdivide into several subdomains, each characterised by its own turbulence regime, where the constants of the model are calibrated independently.

Bibliography

[1] DAFI, . URL https://github.com/xiaoh/DAFI.

[2] DAFI user guide, . URL https://dafi.readthedocs.io/en/latest/.

[3] DNS simulations for turbulent channel flows. URL https://turbulence.oden.utexas.edu/.

[4] Openfoam user guide. URL https://www.openfoam.com/documentation/user-guide.

[5] CFD with open source software. URL http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/.

[6] wallShearStress in OpenFOAM. URL http://aboutcfd.blogspot.com/2017/05/wallshearstress-in-openfoam.html.

[7] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic programming: an introduction*, volume 1. Morgan Kaufmann, 1998.

[8] J. Boussinesq. Essai sur la théorie des eaux courantes. *Mémoires présentés par divers savants à l'Académie des Sciences*, pages 1–680, 1877.

[9] G. Evensen. *Data Assimilation : the Ensemble Kalman Filter*. Springer, 2007.

[10] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The computer journal*, 7:149–154, 1964.

[11] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, pages 35–45, 1960.

[12] B. Launder and D. B. Spalding. The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 1974.

[13] J. Nocedal and S. Wright. *Numerical optimization*. Springer, 2006.

[14] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Taylor Francis, 1980.

[15] A. Quarteroni, R. Sacco, F. Saleri, and P. Gervaio. *Matematica Numerica*. Springer.

[16] P. R. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. *30th Aerospace Sciences Meeting and Exhibit*, 1992.

[17] C. A. M. Strofer, X. Zhang, and X. H. Dafi:an Open-Source Framework for Ensemble-Based Data Assimilation and Field Inversion. *Communications in Computational Physics*, pages 1583–1622, 2021.

[18] C. A. M. Ströfer. Machine learning and field inversion approaches to data-driven turbulence modeling. Master's thesis, Virginia Polytechnic Institute and State University, 2021.

[19] L. Valdettaro. Lecture notes on turbulence, 2020. Mathematics Department, Politecnico di Milano.

[20] A. van Korlaar. Field inversion and machine learning in turbulence modeling. Master's thesis, Delft University of Technology, 2019.

[21] T. von Kármán. Mechanische Ähnlichkeit und Turbulenz. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen*, pages 58–76, 1930.

[22] D. C. Wilcox. *Turbulence Modeling for CFD*. DCW Industries, 2 edition, 1998.

[23] H. Xiao, J. L. Wu, J. X. Wang, R. Sun, and C. Roy. Quantifying and reducing model-form uncertainties in Reynolds-averaged Navier–Stokes simulations: A data-driven, physics-informed bayesian approach. *Journal of Computational Physics*, pages 115–136, 2016.

[24] X. Zhang, T. Gomez, and O. Coutier-Delgosha. Bayesian optimisation of RANS simulation with Ensemble-based variational method in convergent-divergent channel. *Journal of Turbulence*, pages 14–239, 2019.

A | Appendix A

In its original formulation, OpenFOAM provides classical version of k-ω model, where all coefficients are meant as constant. To adapt it to our purposes, a new model has been implemented allowing to work with fields: we will refer to it as k-ωField model and it relies directly on k-ω model original implementation.

In particular Listing A.1 and Listing A.3 show modifications to files, where the model is defined, both included in `src/MomentumTransportModels/momentumTransportModels/RAS/kOmegaFields` directory inside main OpenFOAM folder.

```
protected:

        // Model coefficients
//          dimensionedScalar Cmu_;
//          dimensionedScalar beta_;
//          dimensionedScalar gamma_;
//          dimensionedScalar alphaK_;
//          dimensionedScalar alphaOmega_;

        // Fields
            volScalarField k_;
            volScalarField omega_;

            volScalarField CmuField_;
            volScalarField betaField_;
            volScalarField gammaField_;
            volScalarField alphaKField_;
            volScalarField alphaOmegaField_;
```

```
public:
    typedef typename BasicMomentumTransportModel::alphaField alphaField;
    typedef typename BasicMomentumTransportModel::rhoField rhoField;

    // Constructors
    kOmegaFields
```

```
    (
        const alphaField& alpha,
        const rhoField& rho,
        const volVectorField& U,
        const surfaceScalarField& alphaRhoPhi,
        const surfaceScalarField& phi,
        const transportModel& transport,
        const word& type = typeName
    );

    // Destructor
    virtual ~kOmegaFields()
    {}

    // Member Functions
    // Return the effective diffusivity for k
    tmp<volScalarField> DkEff() const
    {
        return volScalarField::New
        (
            "DkEff",
            alphaKField_*this->nut_ + this->nu()
        );
    }

    // Return the effective diffusivity for omega
    tmp<volScalarField> DomegaEff() const
    {
        return volScalarField::New
        (
            "DomegaEff",
            alphaOmegaField_*this->nut_ + this->nu()
        );
    }
```

Listing A.1: kOmegaField.H

Among protected variables of class `kOmegaField`, coefficients are originally defined as `dimensionedScalar`, here are declared as scalar fields `volScalar Field`. Then, constructors, destructors and some public member functions are adapted to new variables. Note that in this case multiplication operation `*` already allowed operations between `dimensionedScalar` and `volScalarField` as well as between `volScalarField` and `volScalarField`, thus no modifications were needed.

Excluding further marginal adjustement in `kOmegaField.C`, the most relevant modifications are related to how coefficients are read. If in the original formulation they were just constant values, now they are passed as `volScalarField` and since we need to couple our model together with EnKF routine, they also need to allow to vary at eache iteration. For this reason, they are initialized in `caseDir/0` directory in the same way of other fields (`U`, `p`, `k`, `omega` and read at the initial *simpleFoam* iteration. Differently from other fields, they are not modified during the entire solver run. Their original initialization has been removed.

```
betaStarField_
(
    IOobject
    (
        "betaStarField_",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_
),

betaField_
(
    IOobject
    (
        "betaField_",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_
),

gammaField_
(
    IOobject
    (
        gammaField_",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
```

```
    ),
    this->mesh_
),

alphaKField_
(
    IOobject
    (
        alphaKField_",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_
),

alphaOmegaField_
(
    IOobject
    (
        "alphaOmegaField_",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_
)
```

```
// Turbulence specific dissipation rate equation
tmp<fvScalarMatrix> omegaEqn
(
    fvm::ddt(alpha, rho, omega_)
    + fvm::div(alphaRhoPhi, omega_)
    - fvm::laplacian(alpha*rho*DomegaEff(), omega_)
    ==
    gammaField_*alpha()*rho()*G*omega_()/k_()
    - fvm::SuSp(((2.0/3.0)*gammaField_)*alpha()*rho(),divU, omega_)
    - fvm::Sp(betaField_*alpha()*rho()*omega_(), omega_)
    + omegaSource()
    + fvModels.source(alpha, rho, omega_)
);

// Turbulent kinetic energy equation
```

```
tmp<fvScalarMatrix> kEqn
(
    fvm::ddt(alpha, rho, k_)
    + fvm::div(alphaRhoPhi, k_)
    - fvm::laplacian(alpha*rho*DkEff(), k_)
    ==
    alpha()*rho()*G
    - fvm::SuSp((2.0/3.0)*alpha()*rho()*divU, k_)
    - fvm::Sp(CmuField_*alpha()*rho()*omega_(), k_)
    + kSource()
    + fvModels.source(alpha, rho, k_)
);
```

Listing A.2: kOmegaField.C

Moreover, in `kOmegaField.C` we modified k-ω equations definitions: in addition to adapting terms, a new version of template function `fvm::SuSp()` has been implemented. In `src/finiteVolume/finiteVolume/fvm/fvmSup.C` it has been adapted to receive `volScalarField` as input, required by our adapted model.

```
template<class Type>
Foam::tmp<Foam::fvMatrix<Type>>
Foam::fvm::SuSp
(
    const volScalarField&,
    const volScalarField::Internal&,
    const GeometricField<Type, fvPatchField, volMesh>&
)
```

Listing A.3: fvmSup.C

Note that OpenFOAM denotes by `alphaK` what we refer to as σ^* and by `alphaOmega` what we refer to as σ.
Finally, the new defined model has been properly compiled following [5].

List of Figures

List of Tables