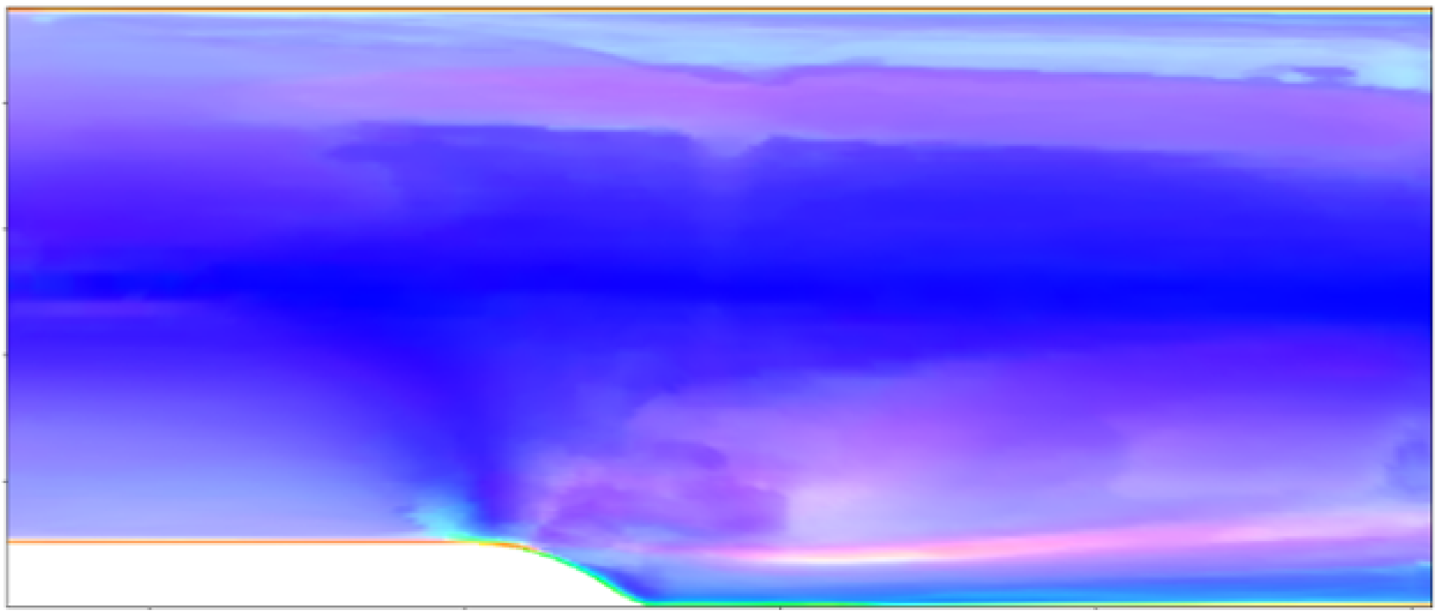
Master of Science Thesis



Machine Learning for Data-Driven RANS Turbulence Modelling

Mikael Kaandorp

February 15, 2018

Faculty of Aerospace Engineering

Delft University of Technology

Machine Learning for Data-Driven RANS Turbulence Modelling

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering
at Delft University of Technology

Mikael Kaandorp

February 15, 2018

Faculty of Aerospace Engineering · Delft University of Technology

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF AERODYNAMICS

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance the thesis entitled **"Machine Learning for Data-Driven RANS Turbulence Modelling"** by **Mikael Kaandorp** in fulfillment of the requirements for the degree of **Master of Science**.

Dated: February 15, 2018

Dr. R.P. Dwight

Dr.ir. M. Gerritsma

Dr.ir. E. van Kampen

M. Schmelzer, MSc

Preface

Before you lies my Master's thesis, titled 'Machine Learning for Data-Driven RANS Turbulence Modelling', which is the final result of my M.Sc. studies in aerodynamics at the Delft University of Technology. In this work, I will present to you the very interesting combination of machine learning, where we let computers learn certain patterns or functions given a set of data, and turbulence modelling, where we attempt to predict the seemingly chaotic motion of fluids, which is one of the great unsolved problems in physics.

I would like to thank my supervisor, dr. Richard Dwight, for giving me the opportunity to work on this very interesting topic. It allowed me to learn a great deal about data-driven techniques in physics, something which I believe will start playing an important role in the future. I appreciate the freedom which I was given during the entire process, which allowed me to work on this topic quite creatively, and I am really happy to say that this work is summarized in a paper for publication as well. Furthermore I would like to thank Martin Schmelzer for his help, and for the very useful conversations about our closely related topics. I would like to thank all my fellow aerodynamics students for accompanying me through this journey, including the members from the 'Periodic Hills club', with special mention to Javier Fatou Gómez for helping me out with the OpenFOAM code for the custom flow solver. I would like to thank my friends from around Delft and Castricum for taking my mind off things during this intensive period, my parents and sister for their support, and last but not least my girlfriend Lysanne for always being there for me.

I hope you will enjoy reading.

Mikael Kaandorp
Delft, the Netherlands
February 15, 2018

Abstract

The application of machine learning algorithms as data-driven turbulence modelling tools for Reynolds Averaged Navier-Stokes (RANS) simulations is presented. A novel machine learning algorithm, called the Tensor Basis Random Forest (TBRF) is introduced. This algorithm can be used to predict the Reynolds stress anisotropy tensor, guaranteeing Galilean invariance by making use of a tensor basis. By modifying a random forest algorithm to accept such a tensor basis, a robust, easy to implement, and easy to train algorithm is created for turbulence modelling, with few hyperparameters which need to be tuned. The individual decision trees in the TBRF are used in a robust outlier filter to improve predictive accuracy, and for uncertainty quantification of the predictions. The algorithm is trained on several flow cases using LES/DNS data, and used to predict the Reynolds stress anisotropy tensor on multiple test flow cases. Results are then propagated with a custom solver implemented in OpenFOAM to yield an improved flow field. Results are compared to a generic random forest, and the Tensor Basis Neural Network (TBNN) from Ling et al. [J. Fluid Mech, 807(2016):155-166, (2016)] on which the TBRF was inspired. Results show that the TBRF algorithm is able to accurately predict the anisotropy tensor for various flow cases, with most of the predictions being realizable and close to the DNS/LES reference data, and that it performs on par with the TBNN algorithm. Resulting mean flows for a square duct flow case and a backward facing step flow case show great resemblance to corresponding DNS and experimental data-sets.

Table of Contents

List of Figures

List of Tables

Nomenclature

Acronyms

BFS	Backward Facing Step
CBFS	Curved Backward Facing Step
CD	Converging-Diverging Channel
DNS	Direct Numerical Simulation
KDE	Kernel Density Estimate
LES	Large Eddy Simulation
MSE	Mean Squared Error
PH	Periodic Hills
RANS	Reynolds Averaged Navier-Stokes
RF	Random Forest
RMSE	Root Mean Squared Error
SD	Square Duct
TBDT	Tensor Basis Decision Tree
TBNN	Tensor Basis Neural Network
TBRF	Tensor Basis Random Forest
TKE	Turbulent Kinetic Energy

Greek Symbols

δ_{ij}	Kronecker delta	-
ϵ	Turbulence dissipation rate	$\mathrm{m}^2\ \mathrm{s}^{-3}$
η	Barycentric map ordinate	-
Γ	Regularization factor	-
γ	Blending parameter	-
$\hat{\Omega}_{ij}$	Normalized mean rotation rate tensor	-
λ	Invariant	-
λ_i	Anisotropy tensor eigenvalues	-
μ	Dynamic viscosity	$\mathrm{kg}\ \mathrm{m}^{-1}\ \mathrm{s}^{-1}$
ν	Kinematic viscosity	$\mathrm{m}^2\ \mathrm{s}^{-1}$
ν_t	Turbulent/eddy viscosity	$\mathrm{m}^2\ \mathrm{s}^{-1}$
ω	Specific turbulence dissipation rate	s^{-1}

Ω_{ij}	Mean rotation rate tensor	s^{-1}
ϕ_i	Reynolds stress eigenvalues	-
ρ	Density	$\mathrm{kg\ m}^{-3}$
σ	Standard deviation	-
τ	Kernel bandwidth parameter	-
τ_w	Wall shear stress	$\mathrm{kg\ m}^{-1}\ \mathrm{s}^{-2}$
$\theta_{med.}$	Median filter threshold	-
$\theta_{optim.}$	Threshold for using an optimization algorithm in the TBDT training process	-
ξ	Barycentric map abscissa	-
Latin Symbols		
$\hat{S}_{ij}$	Normalized mean strain rate tensor	-
$\mathbf{I}$	Identity matrix	-
$\mathbf{X}$	Data-set	-
$\mathcal{P}$	Production term	-
$\mathcal{T}$	Transport term	-
A_k	Antisymmetric turbulent kinetic energy gradient tensor	-
a_{ij}	Reynolds stress anisotropy tensor	$\mathrm{m}^2\ \mathrm{s}^{-2}$
b_{ij}	Normalized Reynolds stress anisotropy tensor	-
C	Barycentric map limiting state coefficient	-
D_{KDE}	Kernel Density Estimate based distance	-
g	Tensor basis coefficient	-
h	Flow geometry hill height	m
J	Minimization objective function	-
j	Decision tree splitting feature	-
k	turbulent kinetic energy	$\mathrm{m}^2\ \mathrm{s}^{-2}$
M	Mahalanobis distance	-
$M_{norm.}$	Normalized Mahalanobis distance	-
$N_{f,s}$	Number of features used for calculating the optimum split in the TBDT	-
$N_{s,l}$	Minimum amount of samples present in the TBDT leaf nodes	-
N_{trees}	Number of TBDT's in the TBRF	-
p	Pressure	$\mathrm{kg\ m\ s}^{-2}$
R_{ij}	Reynolds stress tensor	$\mathrm{m}^2\ \mathrm{s}^{-2}$
Re	Reynolds number	-
s	Decision tree splitting value	-
S_{ij}	Mean strain rate tensor	s^{-1}
t	Time step	-
T_{ij}	Basis tensor	-
u	Velocity	$\mathrm{m\ s}^{-1}$
y^+	Wall distance	-

Sub- and Superscripts

$\{\cdot\}'$	Fluctuation
$[\cdot]_j$	j-th feature of a data-set
$\overline{\{\cdot\}}$	Mean
$\{\cdot\}^{(m)}$	m-th component of the tensor basis
$\{\cdot\}^T$	Transpose
$\{\cdot\}_*$	Test sample
$\{\cdot\}_{1C}$	1-component limiting state turbulence
$\{\cdot\}_{2C}$	2-component limiting state turbulence
$\{\cdot\}_{3C}$	3-component limiting state turbulence
$\{\cdot\}_{norm.}$	Normalized quantity
$\{\cdot\}_{reatt.}$	Reattachment
$\{\cdot\}_{sep.}$	Separation

Chapter 1

Introduction

1.1 Background and relevance

Solving the Navier-Stokes equations for a turbulent flow is still one of the biggest issues in engineering. With a Direct Numerical Simulation (DNS), the result can be very accurate since no turbulence model is used. All length scales and time scales need to be resolved however, making this a very computationally expensive approach, which scales with Re^3 (Pope, 2000). In a Large Eddy Simulation (LES), the influence of the small scale turbulence turbulent motions is modelled, making this less computationally expensive. Solving the Reynolds-averaged Navier-Stokes (RANS) equations is still the most widely used method in engineering applications however (Emory et al., 2013) because of its relatively low computational costs. According to a study from Slotnick et al. (2014), it is expected that using RANS models will still be the norm in 2030, along with an increase in use of hybrid RANS/LES methods.

While higher-order models exist, linear eddy-viscosity models such as the $k-\epsilon$ model and $k-\omega$ model remain the most widely used. As a linear relation is assumed between the Reynolds stress and the mean strain rate tensor by using the Boussinesq hypothesis, much of the anisotropy which is present in real flows is not captured by these models. Some non-linear eddy-viscosity models have been introduced, such as the effective-viscosity model proposed in Pope (1975), and the cubic eddy-viscosity model from Craft et al. (1996). These models use higher order tensors derived from the mean strain rate tensor and mean rotation rate tensor in order to model the Reynolds stress more accurately. These more complex models have not seen widespread adoption however, since they do not always lead to improved performance, and often have worsened convergence properties (Ling et al., 2016b; Gatski and Speziale, 1993).

One possibility for improvement of RANS turbulence modelling would be the introduction of machine learning algorithms, where properties of a flow are predicted in a data-driven sense, by for example using LES, DNS, or experimental data. The last decades have already seen a lot of success with respect to the application of machine learning in many fields of science and technology. Examples are their use in image processing, e.g. road and vehicle detection in autonomous vehicles (Sivaraman and Trivedi, 2011), or analysing vast quantities of data to recognize different patterns and behaviours, e.g. analysing gene data for the possibility of existing tumours (Shipp et al., 2002).

Machine learning has been introduced into the field of turbulence modelling as well, of which two examples are the work done by Ling and Templeton (2015) and Ling et al. (2016b), where machine learning algorithms were used to create markers for RANS model uncertainty, and to model the Reynolds stress anisotropy tensor. As computational power keeps increasing, DNS data and high-fidelity LES data for higher Reynolds numbers becomes more readily available. While originally this data is mainly used to gain more physical insight in turbulent phenomena, it can also potentially be used for modelling turbulence by making use of data-driven machine learning algorithms such as artificial neural networks, random forests, and support vector machines.

1.2 Thesis objective

The aim of this thesis is to further develop machine learning algorithms for the use in data-driven turbulence modelling, by creating a random forest model which predicts the Reynolds Stress anisotropy tensor, and by looking at the possibility of using this model to quantify uncertainty in the output. Using machine learning algorithms in RANS turbulence modelling opens up new opportunities, as traditional methods of developing RANS models from physical understanding seems to be insufficient to improve the models which are currently used (Wang et al., 2017a).

The research is closely related to the work from Ling et al. (2016b), in which an artificial neural network was created to predict the Reynolds stress anisotropy tensor, with a tensor basis to ensure Galilean invariance of the predictions. While using this approach increases predictive capabilities of RANS models, artificial neural networks can be difficult and costly to train, are hard to interpret, and no confidence intervals for the predictions are given. Discrepancies of the Reynolds stress from RANS modelling compared to DNS results can largely be explained due to mean flow features as shown in Xiao et al. (2016), but part of it will remain uncertain. This would make a machine learning approach which takes in account this uncertainty attractive. The random forest algorithm might be a good alternative because of its simplicity and robustness, as well as the fact it consists out of a number of decision trees, perhaps making it suitable to predict uncertainty of the output.

Three main research questions are set-up for this thesis, with an additional seven sub-questions. These research questions are as follows:

- Q1: What limitations do the machine learning based approaches for data-driven RANS modelling such as presented in Ling et al. (2016b) and Ling et al. (2017b) have?

  - S1: Can the results from the models be reproduced?
  - S2: When do predictions of the models break down?

- Q2: Can the machine learning based approach presented in Ling et al. (2016b) be extended by making use of random forests?
  - S3: How should a tensor basis be incorporated in the random forest model in order to ensure Galilean invariance of the Reynolds stress anisotropy tensor prediction?
  - S4: Does this extension yield improvements in accuracy compared to a generic random forest when predicting the Reynolds stress anisotropy tensor?
  - S5: What are the benefits and downsides of using a random forest in comparison with an artificial neural network in order to predict the Reynolds stress anisotropy tensor?

- Q3: Can these machine learning based approaches be improved by including the amount of uncertainty of the prediction?
  - S6: What kind of extensions or improvements can be made to interpolate non-uniqueness in the target variable?
  - S7: Do the proposed extensions/improvements have significant benefits in predicting the Reynolds anisotropy stress tensor?

By answering the research questions, it will be attempted to fulfil the following research objective: "The research objective is to assess whether the use of machine learning algorithms as data driven turbulence modelling tools, such as done in Ling et al. (2016b), can be improved by making use of random forests with embedded invariance, and estimating the amount of uncertainty of the predictions".

In order to make the objective more tangible, several sub-goals have been set up: 1) Implement machine learning algorithms from existing libraries 2) Use these algorithms to create models for the Reynolds stress anisotopy tensor such as done in Ling et al. (2016b) and Ling et al. (2017b), in order to try to reproduce the results 3) Implement a new variant of the random forest model which includes a tensor basis to ensure Galilean invariance of the predicted Reynolds stress anisotropy tensor 4) Compare the results from this new random forest variant with results from generic random forests and the artificial neural network approach from Ling et al. (2016b) 5) Analyse whether it is possible to include uncertainty in the predictions, possibly by making use of the bagging principle of the random forest model.

As the field of machine learning in turbulence modelling is quite new, there are a lot of interesting possibilities, from which the topics specified in the research questions were selected as feasible for the timespan of the project. One additional topic which could be addressed after finishing this research could for example be taking in account non-local effects. So far only local mean flow quantities from RANS simulations have been used as features for the machine learning algorithms. Non-local effects are non-negligible in turbulent flows as indicated in Ling et al. (2016a) however. This means there might still be room for improvement in this aspect as well.

1.3 Thesis overview

This thesis is structured as follows. In Chapter 2 the topic of machine learning applied to RANS turbulence modelling will be introduced. This will be done by introducing different techniques for turbulence modelling, and by discussing some of the most commonly used machine learning algorithms, along with examples of their applications in related works. Afterwards, Chapter 3 will discuss the methodology used to set-up the machine learning framework for turbulence modelling. This includes the different approaches found in literature which were investigated, the implementation of the machine learning algorithms, and the numerical set-up used for the RANS simulations. In Chapter 4 the different flow cases which were used for training and testing the machine learning algorithms will be presented. Chapter 5 will present results from preliminary investigations, in which it was analysed how different machine learning approaches for turbulence modelling as found in literature relate to each other. In Chapter 6 the main results from this thesis will be presented, where the anisotropy tensor is predicted for different flow cases using the TBRF algorithm, and propagated into the flow field. Chapter 7 will present the analysis whether it would be possible to use the TBRF algorithm for quantifying uncertainty of predictions, and in Chapter 8 results from the TBRF algorithm will be further interpreted with respect to the importance of different input features and the tensor basis coefficients. Finally, Chapter 9 will present the conclusions and recommendations coming forth out of this research.

Chapter 2

Machine Learning Applied to RANS Turbulence Modelling

This chapter will give an overview of theory related to turbulence modelling and machine learning. Background on the Navier-Stokes equations and different methods of modelling turbulence will be given in Section 2.1. Theory regarding the Reynolds stress tensor from the RANS equations will be given in Section 2.2, which includes realizability conditions and invariant maps used to display different turbulent states. Invariance of the Navier-Stokes equations, a topic which is important to take in account when creating turbulence models, will be discussed in Section 2.3. In Section 2.4 an overview of different machine learning algorithms will be given, along with previous examples of their application in turbulence modelling.

2.1 Background on the Navier-Stokes equations and RANS

While solving the Navier-Stokes equations using a Direct Numerical Simulation (DNS) is possible, this is not feasible yet for industrial applications. The issue is the huge amount of scales which need to be resolved in order to capture the smallest and largest eddies in a turbulent flow. The Reynolds number of a certain turbulent flow is defined by a characteristic velocity $\mathcal{U}$, a characteristic length scale $\mathcal{L}$, and the kinematic viscosity ν, yielding

$$Re = \mathcal{U}\mathcal{L}/\nu. \tag{2.1}$$

The largest eddies in the flow have Reynolds numbers similar to the Reynolds number of the flow itself, where the length scale of these large eddies is defined by l_0, the characteristic velocity by u_0, and the characteristic time scale by τ_0. The large eddies break up in smaller eddies due to the energy cascade process, eventually becoming so small that molecular viscosity starts to dominate, dissipating the kinetic energy of the eddies. According to the Kolmogorov hypotheses, the smallest scales can be related to the kinematic viscosity and the dissipation rate ϵ. This yields the Kolmogorov scales, with a length scale η, a characteristic velocity u_η, and a characteristic time scale τ_η:

$$\eta = (\nu^3/\epsilon)^{1/4}, \tag{2.2}$$

$$u_\eta = (\epsilon\,\nu)^{1/4}, \tag{2.3}$$

$$\tau_\eta = (\nu/\epsilon)^{1/2}. \tag{2.4}$$

A relation for the ratios between the smallest and largest scales in a turbulent flow can be given. First the dissipation rate ϵ is assumed to be equal to the kinetic energy production rate. This way, ϵ can be related to the characteristic length scales of the larger eddies, using

$$\epsilon \sim u_0^3/l_0. \tag{2.5}$$

When substituting this relation in the Kolmogorov scales, it yields (Pope, 2000):

$$\eta/l_0 \sim Re^{-3/4} \tag{2.6}$$

$$u_\eta/u_0 \sim Re^{-1/4} \tag{2.7}$$

$$\tau_\eta/\tau_0 \sim Re^{-1/2}. \tag{2.8}$$

Looking at these relations, it is clear that the ratios of the smallest scales to the largest scales present in the flow scale with the Reynolds number. Directly solving the Navier-Stokes equations using finite volume methods requires refining the mesh up to a level where the cell sizes are fine enough to capture the small eddies, as well as having a domain which is large enough to capture the largest eddies. Furthermore the time step should be small enough to capture the motion of the smallest eddies as well. For increasing Reynolds numbers the simulation will therefore get more and more expensive. In the case of industrial applications, where the Reynolds number may increase to the order of millions, it is infeasible to resolve the flow using DNS.

Not resolving the very smallest eddies and modelling them instead makes solving the Navier-Stokes equations more feasible for higher Reynolds number flows. This is done in the Large Eddy Simulation (LES). Using a filtering operation, the velocity is broken up into a resolved part and a subgrid-scale component. The filter annihilates the smallest eddies above a certain cut-off wavenumber; in the case of LES the largest eddies containing 80% of the turbulent kinetic energy are resolved. In case less of the turbulent kinetic energy is accounted for, the simulation is called a very-large eddy simulation (VLES). The filtered Navier-Stokes equations need a closure model for the subgrid-scale stress tensor, for which a commonly used model is for example the Smagorinsky model (Pope, 2000).

Finally, one can solve the Reynolds-Averaged Navier Stokes (RANS) equations, where one decomposes the velocity u into a mean quantity $\bar{u}$ and a fluctuating quantity u':

$$u = \bar{u} + u', \tag{2.9}$$

where the mean quantity is calculated by taking the average value over time:

$$\bar{u} = \frac{1}{T}\int_0^T u(x,t)dt. \tag{2.10}$$

This is also done for the pressure, using the same notation with a mean pressure $\bar{p}$ and a fluctuating pressure p'. The Navier-Stokes equation in its incompressible form without any body forces is given by:

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2}. \tag{2.11}$$

The Reynolds decomposed velocity and pressure can be substituted into the Navier-Stokes equation and the continuity equation. Some important relations for the Reynolds averaging operations are that the mean of a fluctuating quantity, i.e. $\overline{u'}$, is zero; but that the mean of two fluctuating quantities multiplying each other, i.e. $\overline{u'u'}$, is not zero. Eventually, one arrives at the Reynolds-average Navier-Stokes equations, which in their incompressible form without body forces are written as:

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = \frac{\partial}{\partial x_j} \left[-\bar{p}\delta_{ij} + \nu \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) - \overline{u_i' u_j'} \right]. \tag{2.12}$$

Although the equations are similar to the Navier-Stokes equations, there is an additional term, $\overline{u_i' u_j'}$, which is called the Reynolds stress, denoted by R_{ij} in its tensor form. A closure model is required for the Reynolds stress term, for which a multitude of different turbulence models are available.

Diagonal components of the Reynolds stress (normal stresses) and the off-diagonal components of the Reynolds stress (shear stresses) can be separated, yielding

$$R_{ij} = \frac{2}{3} k \delta_{ij} + a_{ij}, \tag{2.13}$$

where a_{ij} is the Reynolds stress anisotropy tensor, $k := \frac{1}{2}\text{trace}(R_{ij})$ is the turbulent kinetic energy, and δ_{ij} is the Kronecker delta. It is the anisotropic part of the Reynolds stresses which is important: only this part is effective in transporting momentum, and the isotropic stresses can be absorbed in the modified mean pressure term (Pope, 2000). The non-dimensional Reynolds stress anisotropy tensor, b_{ij}, is defined as:

$$b_{ij} = \frac{R_{ij}}{2k} - \frac{1}{3}\delta_{ij}. \tag{2.14}$$

This is the main quantity of interest in the remainder of this thesis, and will be referred to as the anisotropy tensor.

Different categories exist within RANS turbulence models, which will be discussed in the following paragraphs. Information is obtained from (Pope, 2000) unless specified otherwise.

2.1.1 Turbulent-Viscosity Models

The first class of RANS turbulence models are turbulent-viscosity models. Often a linear relation is assumed between the anisotropy tensor a_{ij} and the mean strain rate tensor S_{ij}, which is defined by

$$S_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right). \tag{2.15}$$

The linear relation models the anisotropy tensor similarly to the diffusive term in the Navier-Stokes equations:

$$a_{ij} \approx -\nu_t\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) = -2\nu_t S_{ij}. \tag{2.16}$$

In this equation, ν_t is called the turbulent-viscosity, or eddy-viscosity. Linear turbulent-viscosity models are based on the assumption of a simple shear flow, where turbulent characteristics and velocity gradients change relatively slowly compared to the mean flow. Examples of linear eddy-viscosity models often used in engineering are the $k-\epsilon$, $k-\omega$, and Spalart-Allmaras turbulence models (Launder and Spalding, 1974; Wilcox, 2008; Spalart and Allmaras, 1992). These models have different methods of calculating the turbulent viscosity.

In one-equation models, the transport equation for one turbulent quantity is solved. This can for example be the turbulent kinetic energy k. In the case of models based on the Prandtl mixing length theory, one can for example model the eddy-viscosity using

$$\nu_t = ck^{1/2}l_m, \tag{2.17}$$

where c is a constant, and l_m the mixing length. One commonly used one-equation model is the Spalart-Allmaras turbulence model, where a transport equation for a modified eddy-viscosity is solved.

The $k-\epsilon$ and $k-\omega$ models are examples of two-equation models, where transport equations for k in combination with either ϵ or ω are solved. Afterwards, one can calculate the eddy-viscosity using $\nu_t = C_\mu k^2/\epsilon$ or $\nu_t = C_\mu k/\omega$. In these relations C_μ is an empirically determined coefficient, for which $C_\mu = 0.09$ is often used. Instead of using the similarity relation in 2.5 for ϵ, one can relate it to the turbulent kinetic energy k and a mixing length l_m using $\epsilon = C_\mu k^{3/2}/l_m$. Combining this relation with (2.17), one can eliminate the mixing length, and get an expression for C_μ in terms of k, ϵ, and ν_t. The value for C_μ can then be determined using experimental or numerical data, where it can be observed that this quantity is approximately constant at a value of 0.09.

The transport equation for the turbulent kinetic energy can be derived from the Navier-Stokes equations, yielding

$$\frac{Dk}{Dt} + \nabla \cdot \mathcal{T} = \mathcal{P} - \epsilon, \tag{2.18}$$

where $\mathcal{T}$ and $\mathcal{P}$ are the transport and production terms respectively, and where $\frac{D}{Dt}$ denotes the material derivative.

In case of a turbulent-viscosity model, a closure relation is necessary for the transport and production term. As an example, the exact definition of the transport term, and the closure relation used in the $k-\epsilon$ model are given by:

$$\mathcal{T}_i = \frac{1}{2}\overline{u_i'u_j'u_j'} + \overline{u_i'p'}/\rho - 2\nu\overline{u_j's_{ij}'} \approx \frac{\nu_t}{\sigma_k}\nabla k. \tag{2.19}$$

A constant σ_k can be seen in the closure model for the transport term, which is set to a value of 1.0 by examining empirical data for free turbulent flows (Launder and Spalding, 1974).

The exact definition of the production term, and the closure relation normally used in eddy viscosity models is given by:

$$\mathcal{P} = -\overline{u_i'u_j'}\frac{\partial \bar{U}_i}{\partial x_j} \approx \nu_t \left(\frac{\partial \bar{U}_i}{\partial x_j} + \frac{\partial \bar{U}_j}{\partial x_i}\right)\frac{\partial \bar{U}_i}{\partial x_j}. \tag{2.20}$$

As can be seen this term is the product of the Reynolds stress with the velocity gradient tensor, which means the closure model can straightforwardly use the Reynolds stress as calculated by the eddy-viscosity hypothesis.

For the transport equation of the rate of dissipation ϵ, even more empirical relations are used. In the $k-\epsilon$ turbulence model, it is given by:

$$\frac{D\epsilon}{Dt} = \nabla \cdot \left(\frac{\nu_t}{\sigma_\epsilon}\nabla\epsilon\right) + C_{\epsilon 1}\frac{\mathcal{P}\epsilon}{k} - C_{\epsilon 2}\frac{\epsilon^2}{k}. \tag{2.21}$$

Three coefficients need calibration in this transport equation by using for example numerical or experimental data. When using the $k-\omega$ turbulence model, the transport equation for the turbulence frequency ω looks similar to that of ϵ. In this case one would use:

$$\frac{D\omega}{Dt} = \nabla \cdot \left(\frac{\nu_t}{\sigma_\omega}\nabla\omega\right) + C_{\omega 1}\frac{\mathcal{P}\omega}{k} - C_{\omega 2}\omega^2, \tag{2.22}$$

which also contains a number of coefficients which need to be calibrated.

Both the $k-\epsilon$ and $k-\omega$ turbulence models have specific well known shortcomings, which are for example discussed in Menter (1993). First of all the $k-\epsilon$ turbulence model is known to be insensitive against adverse pressure gradients, leading to delay of separation. The model also poses some issues when using it in a low Reynolds number simulation, where the flow field is resolved through the viscous sublayer, instead of using wall functions to represent the flow in the boundary layer. One example of a modified $k-\epsilon$ model which can be used in a low Reynolds number simulation is the one from Launder and Sharma (1974). In this case damping functions are used in the viscous sublayer, and a modified version of ϵ is used which goes towards zero at the wall, making the implementation of boundary conditions easier. The $k-\omega$ turbulence model generally performs better in the case of adverse pressure gradients, and can directly be used in a low Reynolds number simulation. Its major shortcoming is its strong dependence on the freestream values for ω however. In Menter (1993) the advantages of both models are combined by blending both models together (resulting in the $k-\omega$ SST turbulence model), using the $k-\omega$ turbulence model closer to the wall.

Flow cases in which the simple shear flow assumption used for the linear turbulent-viscosity model breaks down are for example strongly swirling flows, and flows with significant streamline curvature (Pope, 2000). It is possible to create higher order turbulent-viscosity models, which use more tensors than just the strain rate tensor S_{ij} to get a relation for a_{ij}. One example is the work presented in Pope (1975). Here it is assumed that the Reynolds stresses are uniquely related to the rates of strain and local scalar quantities (i.e. transport terms, accounting for convection of stresses by the mean and fluctuating flow are negligible), along with an assumption of high Reynolds number flow. This results in $b_{ij} = f(\hat{S}_{ij}, \hat{\Omega}_{ij})$, where

$\hat{S}_{ij}$ is the normalized mean strain rate tensor, and $\hat{\Omega}_{ij}$ is the normalized mean rotation rate tensor, which are defined by

$$\hat{S}_{ij} = \frac{1}{2}\frac{k}{\epsilon}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right), \quad \hat{\Omega}_{ij} = \frac{1}{2}\frac{k}{\epsilon}\left(\frac{\partial \bar{u}_i}{\partial x_j} - \frac{\partial \bar{u}_j}{\partial x_i}\right). \tag{2.23}$$

Eventually an expression for b_{ij} is obtained in terms of a tensor basis by making use of the Cayley-Hamilton theorem, where 10 scalars $g^{(m)}$ multiply 10 basis tensors $T_{ij}^{(m)}$ which are a function of the mean strain rate tensor and the mean rotation rate tensor:

$$b_{ij} = \sum_{m=1}^{10} g^{(m)} T_{ij}^{(m)}(\hat{S}_{ij}, \hat{\Omega}_{ij}). \tag{2.24}$$

More information on this tensor basis will be given in Section 2.3, as it forms an important basis for the research presented in this thesis.

Another example of a non-linear eddy-viscosity model based on the idea from Pope (1975) is the model from Craft et al. (1996). This model was developed to capture streamline curvature effects, one of the shortcomings of linear eddy-viscosity models. In this case the linear, quadratic, and cubic basis tensors in (2.24) are retained, resulting in an expression for a_{ij} which includes the linear relation between a_{ij} and S_{ij}, plus seven additional terms with the higher order tensors. These seven additional terms all include different coefficients which were calibrated.

2.1.2 Reynolds Stress Models

The second class of RANS turbulence models which will briefly be discussed are the Reynolds stress models. In Reynolds stress models, no eddy-viscosity is used. Instead the Reynolds stresses are transported according to the Navier-Stokes equations, resulting in

$$\frac{D}{Dt}\overline{u_i'u_j'} + \frac{a}{\partial x_k}\mathcal{T}_{kij} = \mathcal{P}_{ij} + \mathcal{R}_{ij} - \epsilon_{ij}, \tag{2.25}$$

where $\mathcal{T}_{kij}$ is the Reynolds-stress flux, $\mathcal{P}_{ij}$ is the production term, $\mathcal{R}_{ij}$ is the pressure-rate-of-strain tensor, and ϵ_{ij} is the dissipation. Closure models are required for $\mathcal{T}_{kij}$, $\mathcal{R}_{ij}$, and ϵ_{ij}. Since the Reynolds stress tensor is symmetric, this means 6 equations are solved for the different Reynolds stresses. The turbulent kinetic energy k can directly be obtained from the Reynolds stress. A transport equation for ϵ is solved as well in order to obtain length and time scales.

Of the terms which require a closure model, $\mathcal{R}_{ij}$ is the most important, and plays the role of redistributing energy among the Reynolds stresses. It is given by

$$\mathcal{R}_{ij} = \overline{\frac{p'}{\rho}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)}. \tag{2.26}$$

It can be shown that the fluctuating pressure can be divided into a rapid, slow , and harmonic contribution. Different limiting states of turbulence can be identified. In the first case, decaying homogeneous anisotropic turbulence, it is assumed that turbulence becomes less anisotropic as it decays. In this case only the slow contribution of $\mathcal{R}_{ij}$ ($\mathcal{R}_{ij}^{(s)}$) plays a role. In order to relate $\mathcal{R}_{ij}^{(s)}$ to b_{ij}, the Cayley-Hamilton theorem can be used to write $\mathcal{R}_{ij}^{(s)}$ in terms of tensors created from b_{ij}, similarly to the expansion in (2.24). In the simplest case only the first term of the series can be retained, resulting in a linear relation between $\mathcal{R}_{ij}^{(s)}$ and b_{ij}, as proposed in the model from Rotta (1951). The other limiting case is the rapid-distortion limit. In this case the mean-velocity-gradient terms dominate, and only the rapid contribution of $\mathcal{R}_{ij}$ is used ($\mathcal{R}_{ij}^{(r)}$).

Most of the time both $\mathcal{R}_{ij}^{(s)}$ and $\mathcal{R}_{ij}^{(r)}$ play a role and need to be taken in account however. Examples of commonly used Reynolds stress models are the LRR-IP model from Launder et al. (1975), and the SSG turbulence model from Speziale et al. (1991). Instead of using the Cayley-Hamilton theorem to write $\mathcal{R}_{ij}$ as a function of tensors based on b_{ij} only, a tensor basis can be created using b_{ij}, S_{ij} and Ω_{ij}. In both the LRR-IP model and the SSG model, $\mathcal{R}_{ij}$ can written as a combination of 5 of these basis tensors, multiplied by 5 coefficients which have different values for both models.

The $\mathcal{T}_{kij}$ term can be decomposed into three fluxes, caused by viscous diffusion, pressure transport, and turbulent convection, from which the viscous diffusion term is negligible and in closed form. In for example the LRR-IP model only the turbulent convection ($\overline{u_i'u_j'u_k'}$) is modelled by using k, ϵ, and the Reynolds stresses $\overline{u_i'u_j'}$. Finally the ϵ_{ij} term is often modelled by using $\frac{2}{3}\delta_{ij}\epsilon$ (assuming local isotropy).

2.2 Realizability of the Reynolds stress and invariant maps

As explained in the previous section, the Reynolds stress term in the RANS equations can be modelled in a multitude of different ways. Although in theory Reynolds stress models should be better able to predict the Reynolds stress, turbulence models assuming a linear relation between the anisotropy tensor a_{ij} and the mean strain rate tensor S_{ij} remain the most commonly used. The more complicated non-linear eddy-viscosity models have not seen widespread adoption since they do not always lead to improved performance, and often have worsened convergence properties (Ling et al., 2016b; Gatski and Speziale, 1993).

This use of an eddy-viscosity which linearly relates the strain rate tensor to the Reynolds stress introduces a modelling error. Several different maps exist in which all realizable turbulent states can be plotted. These maps can be useful to give an indication of the limited turbulent states which linear eddy-viscosity models are able to represent. Realizability constraints can be derived by looking at the properties of the Reynolds stress.

For a given matrix $\mathbf{A}$, positive semi-definiteness is satisfied when

$$\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0 \tag{2.27}$$

for any non-zero column vector x with real numbers. The outer product of the turbulent fluctuation vector $\mathbf{u}'$ with itself can be taken, which yields:

$$\mathbf{u}' \otimes \mathbf{u}' = \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} \begin{bmatrix} u' & v' & w' \end{bmatrix} = \begin{bmatrix} u'u' & u'v' & u'w' \\ v'u' & v'v' & v'w' \\ w'u' & w'v' & w'w' \end{bmatrix} = u'_i u'_j. \tag{2.28}$$

Any matrix or tensor formed this way satisfies positive semi-definiteness, since

$$\mathbf{x}^T \mathbf{u}' \mathbf{u}'^T \mathbf{x} = (\mathbf{x}^T \mathbf{u}')^2 \geq 0. \tag{2.29}$$

The Reynolds stress is obtained from $u'_i u'_j$ by taking the temporal average over n samples, yielding $\overline{u'_i u'_j} = \frac{1}{n} \sum_n u'_i u'_j$, which is still positive semi-definite since all the samples adhere to this property. The Reynolds stress tensor being positive semi-definite means that all its eigenvalues, and therefore also its determinant and trace will be non-negative, as well as having non-negative diagonal components. This yields:

$$R_{\alpha\alpha} \geq 0, \quad det(R_{ij}) \geq 0, \tag{2.30}$$

where for the Greek indices the summation convention is not used. Additionally, it can be shown from (2.27) using orthonormal sets of basis vectors for $\mathbf{x}$, that

$$R^2_{\alpha\beta} \leq R_{\alpha\alpha} R_{\beta\beta}, \quad \forall\, \alpha \neq \beta \tag{2.31}$$

must hold, the Cauchy-Schwarz inequality.

Since the Reynolds stress is a real and symmetric tensor, it is possible to perform an eigenvalue decomposition. The Reynolds stress can be broken up as

$$R_{ij} = 2k \left(\frac{1}{3} \delta_{ij} + v_{in} \Phi_{nl} v_{jl} \right), \tag{2.32}$$

where v_{ij} is the matrix containing the eigenvectors, and Φ_{ij} is the diagonal matrix containing the eigenvalues ϕ_i. Looking at this representation, one can identify an amplitude (the turbulent kinetic energy), an orientation (the eigenvectors in v_{ij}) and a shape (the eigenvalues in Φ). The eigenvectors can be used to form a coordinate system, called the principal coordinate system. With respect to this coordinate system the Reynolds stress will be diagonal. Using this fact, Equation (2.30), and the definition of the turbulent kinetic energy, it can easily be shown that the eigenvalues of the Reynolds stress will therefore fall between 0 and $2k$. Eigenvalues of the Reynolds stress tensor (ϕ_i) can be related to eigenvalues of the Reynolds stress anisotropy tensor (λ_i) by:

$$\lambda_i = \frac{\phi_i}{2k} - \frac{1}{3}. \tag{2.33}$$

From this it follows that the eigenvalues and diagonal components of b_{ij} fall between $-1/3$ and $+2/3$. Furthermore using (2.31) it can be shown that the off-diagonal components of b_{ij} fall between $-1/2$ and $+1/2$, which is the case when $R_{\alpha\beta}$ reaches either $-k$ or $+k$.

Two independent invariants of the anisotropy tensor can be identified: II ($b_{ij}b_{ji}$) and III ($b_{ij}b_{in}b_{jn}$). On the II-III plane all realizable states of turbulence can be plotted, which all fall within a triangular shape since the components of the tensor fall within the previously mentioned limits. This fact was used to create a triangular invariant map in Lumley and Newman (1977). Based on the eigenvalues of the anisotropy tensor and their limits, all realizable states can be plotted on a triangular eigenvalue map as well as introduced in Lumley (1979).

The barycentric map was introduced in Banerjee et al. (2007). From the anisotropy tensor three limiting states can be identified, corresponding to 1-component turbulence (turbulence in one direction, one eigenvalue of b_{ij} is non-zero), 2-component turbulence (restricted to a plane, two eigenvalues of b_{ij} are non-zero), and 3-component turbulence (isotropic turbulence, three eigenvalues of b_{ij} are non-zero). These three limiting states correspond to one of the corners in the barycentric map. The anisotropy tensor is expressed in terms of its three limiting states, using basis tensors b_{1c}, b_{2c}, and b_{3c}, giving:

$$b_{ij} = C_{1c}\, b_{1c} + C_{2c}\, b_{2c} + C_{3c}\, b_{3c}. \tag{2.34}$$

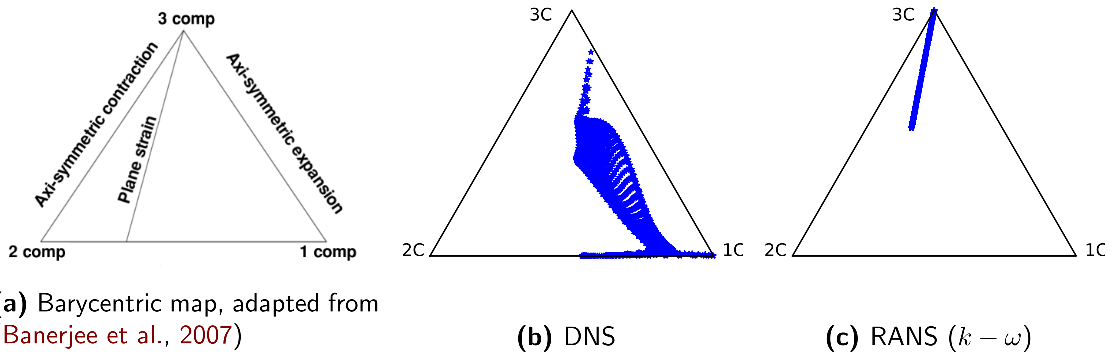
The coefficients (C_{1c}, C_{2c}, C_{3c}) in this tensor basis can directly be calculated from the anisotropy tensor eigenvalues using (Banerjee et al., 2007):

$$C_{1c} = \lambda_1 - \lambda_2, \quad C_{2c} = 2(\lambda_2 - \lambda_3), \quad C_{3c} = 3\lambda_3 + 1. \tag{2.35}$$

In order plot the results on a 2-dimensional plane, the three limiting states can be assigned coordinates corresponding to three vertices of a triangle. The coordinates of this 2-dimensional map will be denoted by ξ and η. After giving the vertices of the triangle a location on the $\xi - \eta$ plane (ξ_{1c},η_{1c}; ξ_{2c},η_{2c}; ξ_{3c},η_{3c}), different turbulent states can be plotted in the triangle using:

$$\xi = C_{1c}\, \xi_{1c} + C_{2c}\, \xi_{2c} + C_{3c}\, \xi_{3c}, \quad \eta = C_{1c}\, \eta_{1c} + C_{2c}\, \eta_{2c} + C_{3c}\, \eta_{3c}. \tag{2.36}$$

The resulting barycentric map is presented in Figure 2.1a. For a 2D linear eddy-viscosity RANS simulation, the predicted Reynolds stress tensor will lie entirely along the line indicated by plane strain, while its true value could lie in the entire triangle. Figure 2.1b and Figure 2.1c present the eigenvalue locations on the barycentric map of a square duct flow case for DNS and RANS ($k - \omega$) respectively. As can be seen there is a significant difference.



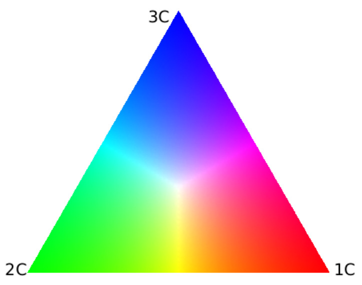
(a) Barycentric map, adapted from (Banerjee et al., 2007)

(b) DNS

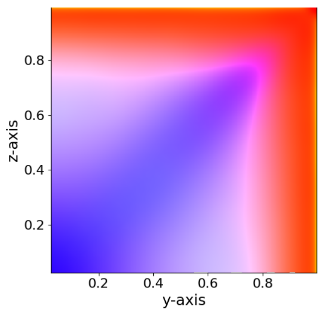
(c) RANS ($k - \omega$)

Figure 2.1: Barycentric map, along with results for a turbulent square duct at Re = 3,500. DNS data from Pinelli et al. (2010)

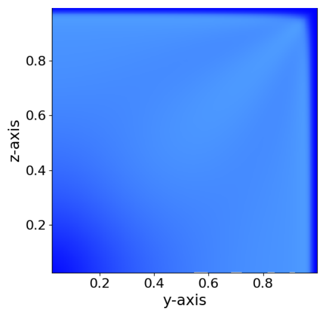
Proximity to one of the corners in the barycentric map can be expressed in a Red-Green-Blue (RGB) map, in order to visualize the type of turbulence locally in the flow field. The coefficients C_{1c}, C_{2c}, and C_{3c} in (2.34) add up to one. This means these coefficients can directly be translated into an RGB array, as for example demonstrated in Tracey et al. (2013). Figure 2.2a presents the colormap applied to the barycentric map as used in this thesis. The RGB-array created from C_{1c}, C_{2c}, and C_{3c} was normalized using its maximum value in order to increase the brightness of the colormap. Figure 2.2b and Figure 2.2c present the use of this colormap for the square duct flow case, corresponding to the barycentric map presented in Figure 2.1b and Figure 2.1c. It can clearly be seen that the DNS data shows one-component turbulence close to the wall, which goes to three component turbulence in the centerline of the duct, whereas the RANS simulation only represents turbulence near the three-component limit.



(a) Colormap



(b) DNS



(c) RANS $(k-\omega)$

Figure 2.2: RGB colormap used to represent the turbulent states for a square duct flow case. Only the upper right quadrant of the square duct is shown, with the mean flow moving out of plane. DNS data from Pinelli et al. (2010).

2.3 Invariance of the Navier-Stokes equations

The Navier-Stokes equations are Galilean invariant: this means that the laws of motion should be the same in any inertial frame (bodies on which the net force is zero are at rest or moving at a constant velocity in an inertial frame). Taking the reference frame, and specifying a fixed rotation, fixed or constant translation, or reflection, should therefore not have any influence.

Taking the Reynolds-averaged Navier-Stokes Equations, and transforming an original frame of reference using a rotation or reflection matrix $\mathbf{Q}$, means that all vector and tensor quantities in the original frame, e.g. the velocity $\mathbf{u}$ or Reynolds stress $\mathbf{R}$, are transformed to $\mathbf{Qu}$ and $\mathbf{QRQ}^T$ in the new frame of reference. This also holds for the anisotropy tensor b_{ij} which is the quantity to be predicted by the machine learning algorithms in this thesis.

Invariance plays an important role when creating a regressor in the context of fluid dynamics. For example, when creating a regressor for a scalar such as p, the outcome of the function should not depend on the orientation of the coordinate system. In case of a scalar function f, which uses a tensor $\mathbf{S}$ and a vector $\mathbf{v}$ as inputs, the following should hold (Ling et al., 2016a):

$$f(\mathbf{S},\mathbf{v}) = f(\mathbf{QSQ}^T,\mathbf{Qv}), \tag{2.37}$$

where $\mathbf{Q}$ can be any rotation matxrix (orthogonal matrix with det $=1$). When creating a function τ which predicts a *tensor* quantity, this tensor quantity is aligned with a certain coordinate system. This means that the following should hold (Speziale et al., 1991):

$$\mathbf{Q}\,\tau(\mathbf{S},\mathbf{v})\,\mathbf{Q}^T = \tau(\mathbf{QSQ}^T,\mathbf{Qv}). \tag{2.38}$$

By deriving invariant input features for the regression function from vector and tensor quantities in the flow field, one can guarantee (2.37). An example is presented in (Ling et al., 2016a), where it is shown that the Hilbert Basis Theorem can be used to derive a finite set of all independent invariants (called an integrity basis) from a collection of tensors and vectors. In Wang et al. (2017a) this approach is used to derive a set of 47 invariants based on $\nabla\bar{\mathbf{u}}$, $\nabla\mathbf{k}$, and $\nabla\mathbf{p}$.

When predicting a tensor quantity such as b_{ij}, one can guarantee (2.38) by using both invariant input features, and by using an integrity basis for b_{ij} which is invariant to the full orthogonal group. The propery $\mathbf{Q}^{-1} = \mathbf{Q}^T$ holds for all matrices in this group, which contains all rotation and reflection matrices (Ling et al., 2016a). In Pope (1975) an integrity basis was derived for b_{ij} in order to create a more general eddy-viscosity model. As explained earlier in Section 2.1.1, it is assumed that the Reynolds stresses are uniquely related to the rates of strain and local scalar quantities, resulting in $b_{ij} = f(\hat{S}_{ij},\hat{\Omega}_{ij})$.

An infinite tensor polynomial can be formed from $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$ to construct the expression for b_{ij}. It can be shown using the Cayley-Hamilton theorem, that there is a limited number of linearly independent tensors which can be formed from $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$. In case of a three-dimensional rank-2 tensor $\mathbf{S}$, it can be shown from the Cayley-Hamilton theorem that

$$\mathbf{S}^3 - \mathrm{I}_s\mathbf{S}^2 + \mathrm{II}_s\mathbf{S} - \mathrm{III}_s\mathbf{I} = 0, \tag{2.39}$$

where I_s is the trace of $\mathbf{S}$, II_s is $\frac{1}{2}\left(\mathrm{trace}(\mathbf{S})^2 - \mathrm{trace}(\mathbf{S}^2)\right)$, and III_s is the determinant of $\mathbf{S}$. As can be seen $\mathbf{S}^3$ can be written as a combination of $\mathbf{S}^2$, $\mathbf{S}$, and the identity tensor $\mathbf{I}$. Pre multiplying this equation by $\mathbf{S}$ shows that $\mathbf{S}^4$ can be represented by a linear combination of $\mathbf{S}^2$, $\mathbf{S}$ and $\mathbf{I}$ as well, and so on for $\mathbf{S}^n$ for $n \le 3$. A slightly generalized version of this approach is used to derive 10 linearly independent basis tensors which can be formed from $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$. These ten basis tensors $T_{ij}^{(m)}$ are multiplied with ten scalar coefficients $g^{(m)}$ to obtain an expression for b_{ij}:

$$b_{ij} = \sum_{m=1}^{10} T_{ij}^{(m)}(\hat{S}_{ij},\hat{\Omega}_{ij})\; g^{(m)}(\lambda_1,...,\lambda_5), \tag{2.40}$$

where the scalar coefficients $g^{(m)}$ are a function of the tensor invariants λ. From $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$ five independent invariants can be derived. The basis tensors and invariants created from $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$ are given by:

$$\begin{aligned}
\mathbf{T}^{(1)} &= \hat{\mathbf{S}} & \mathbf{T}^{(6)} &= \hat{\mathbf{\Omega}}^2\hat{\mathbf{S}} + \hat{\mathbf{S}}\hat{\mathbf{\Omega}}^2 - \tfrac{2}{3}\mathbf{I}\cdot\mathrm{trace}(\hat{\mathbf{S}}\hat{\mathbf{\Omega}}^2)\\
\mathbf{T}^{(2)} &= \hat{\mathbf{S}}\hat{\mathbf{\Omega}} - \hat{\mathbf{\Omega}}\hat{\mathbf{S}} & \mathbf{T}^{(7)} &= \hat{\mathbf{\Omega}}\hat{\mathbf{S}}\hat{\mathbf{\Omega}}^2 - \hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}\hat{\mathbf{\Omega}}\\
\mathbf{T}^{(3)} &= \hat{\mathbf{S}}^2 - \tfrac{1}{3}\mathbf{I}\cdot\mathrm{trace}(\hat{\mathbf{S}}^2) & \mathbf{T}^{(8)} &= \hat{\mathbf{S}}\hat{\mathbf{\Omega}}\hat{\mathbf{S}}^2 - \hat{\mathbf{S}}^2\hat{\mathbf{\Omega}}\hat{\mathbf{S}}\\
\mathbf{T}^{(4)} &= \hat{\mathbf{\Omega}}^2 - \tfrac{1}{3}\mathbf{I}\cdot\mathrm{trace}(\hat{\mathbf{\Omega}}^2) & \mathbf{T}^{(9)} &= \hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}^2 + \hat{\mathbf{S}}^2\hat{\mathbf{\Omega}}^2 - \tfrac{2}{3}\mathbf{I}\cdot\mathrm{trace}(\hat{\mathbf{S}}^2\hat{\mathbf{\Omega}}^2)\\
\mathbf{T}^{(5)} &= \hat{\mathbf{\Omega}}\hat{\mathbf{S}}^2 - \hat{\mathbf{S}}^2\hat{\mathbf{\Omega}} & \mathbf{T}^{(10)} &= \hat{\mathbf{\Omega}}\hat{\mathbf{S}}^2\hat{\mathbf{\Omega}}^2 - \hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}^2\hat{\mathbf{\Omega}}
\end{aligned} \tag{2.41}$$

$$\lambda_1 = \text{trace}(\hat{\mathbf{S}}^2) \quad \lambda_2 = \text{trace}(\hat{\mathbf{\Omega}}^2) \quad \lambda_3 = \text{trace}(\hat{\mathbf{S}}^3) \quad \lambda_4 = \text{trace}(\hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}) \quad \lambda_5 = \text{trace}(\hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}^2). \tag{2.42}$$

2.4 Brief overview of machine learning algorithms

Before going into detail about previous work regarding the application of machine learning algorithms to turbulence modelling, some background regarding different kinds of algorithms will briefly be presented, as well as standard practices used for training and validating machine learning models.

2.4.1 Classes of algorithms

Different distinctions can be made for machine learning algorithms. In Russell and Norvig (2010) three different classes are identified, depending on the feedback they get:

- Supervised Learning: An algorithm is given a set of input features, and a corresponding set of output features, for which the algorithm then creates a model. A very basic example would be linear least-squares regression, where a linear model is created to fit given data. Depending on the output, supervised learning algorithms can be divided into classifiers (given an input, the algorithm decides what class the output belongs to), and regressors (given an input, the algorithm returns a value for the output).
- Reinforcement Learning: in this case the algorithm produces certain actions, and given these actions the algorithm receives rewards or punishments. The goal is to maximize the future rewards over its lifetime.
- Unsupervised Learning: an algorithm tries to find patterns in input data, e.g. clustering of data

This thesis will focus on supervised learning algorithms, and more specifically the regression algorithms. A multitude of different supervised learning algorithms exist, for which some will be discussed in Section 2.5.

2.4.2 Training, validation, and the bootstrap method

Theory in this subsection was obtained from Hastie et al. (2008), unless specified otherwise. When using machine learning algorithms to model a certain process, two different goals need to be met: selecting the right model, and assessing the performance of the selected model. When there is enough data available, a good choice is dividing the dataset into three parts:

training data, used to train the model; validation data, used to tune hyperparameters and used to obtain the an estimate on the prediction error which can be used to select the best model; and test data, giving an indication on generalization error of the selected model.

The training error is not a good measure for the model accuracy. Making the machine learning model more complex might lead to a lower training error, but can result in overfitting and poor generalization of the predictions. Typically more complex models have a lower bias (deviation of the average from the true mean) but a high variance (expected squared deviation of the prediction around its mean).

A typical split for dividing the total available data into training, validation, and test data could for example be 50%, 25%, 25% respectively. This case where training data and validation data is kept separate is called holdout cross-validation (Russell and Norvig, 2010). After using the validation data to select the best model, the selected algorithm can be trained on the training data and validation data for future cases. When there is not much data available different techniques can be used to replace the validation step: (K-fold) cross-validation and bootstrap methods.

Cross-validation K-fold cross-validation can be performed when the data-set is not large enough to split it into three distinct parts. In this case, the training data is split into K-parts, of which $K-1$ parts are used for training, and 1 part for validation. This process is repeated K times, so that each part of the training data is used for validation once. The K estimates are used to estimate the prediction error of the model. One variation of the K-fold cross validation is the leave-one-out cross validation (LOOCV): in this case all data, except from one data point, is used for training, and this is repeated until every data point has been used for validation.

Cross validation can be used to determine hyperparameters in machine learning algorithms. In this case the cross-validation error is calculated for various choices of the hyperparameters, and the parameters minimizing the cross-validation error are then chosen. Different methods for choosing the best hyperparameters are available, such as Bayesian optimization as done in Ling et al. (2016b), or by performing a simple (but costly) grid search.

Bootstrap Methods When using bootstrap methods, random equally sized subsets from the data are obtained, which is repeated a number of times. The model is fitted with each of the bootstrap datasets. The random samples from the dataset are obtained 'with replacement', which means that samples can be present multiple times in the training data, or not at all. An estimate on the prediction error can be obtained by using samples not present in the bootstrap training set.

Bootstrap methods can also be used to combine a number of approximately unbiased regressors/classifiers. Using bootstrap methods to create a number of these regressors/classifiers to predict a certain value (by for example voting) is called bagging, and can be used to reduce the variance of the output. By using multiple bootstrap replicates of the data to train a certain model, it is furthermore possible to make a prediction on the variance of the output.

2.5 Supervised learning and examples used in turbulence modelling

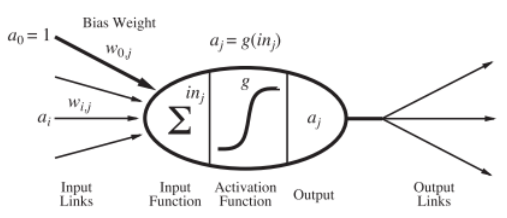
In this section a brief overview of the most relevant machine learning algorithms is given: the artificial neural network, methods based on the decision tree, and support vector machines. Additionally some kernel-based methods which are able to be used stochastically will be discussed. For all algorithms examples as found in literature for their application in turbulence modelling scenarios will be presented as well.

2.5.1 Artificial neural networks

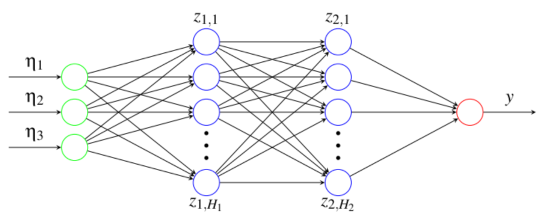
Background Artificial neural networks consist of nodes which are connected to each other. Each node has a certain activation function associated to it, each connection a certain weight. Each node can receive multiple inputs, consisting of activations a_i with weights $w_{i,j}$. The inputs often include one dummy input $a_0 = 1$ with weight $w_{0,j}$ (called the bias). The weighted sum of the inputs in_j is calculated using

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{2.43}$$

for each node (Russell and Norvig, 2010). Afterwards, the activation function produces an output by evaluating in_j. Different activation functions can be used, such as sigmoids, or linear activation functions. These outputs can then be fed to a subsequent node. An illustration of such an artificial neuron is presented in Figure 2.3a. Networks between the nodes can be created in different ways, for example by using a feed-forward network. Nodes are put in layers, and each layer receives its input from the previous layer. In Figure 2.3b an example is shown of a feed-forward network from Singh et al. (2016). There is an input layer, and an output layer, connected by so called hidden layers, in this case 2. Neural networks like this are known as multilayer perceptrons (MLP) (Kubat, 2015).



(a) Artificial neuron, from (Russell and Norvig, 2010)



(b) ANN with 2 hidden layers, from (Singh et al., 2016)

Figure 2.3: Overview of the artificial neural network

Training of networks is done by back-propagation. The error at the output nodes is calculated with the training data and propagated back to the nodes in the previous layer, after which weights of the connections are updated. Further information on this process can be found in Russell and Norvig (2010) or Hastie et al. (2008). Presenting each training point to the neural network is called an epoch (Kubat, 2015). During training the optimization process loops through a certain amount of epochs until a suitable error level is reached.

The artificial neural network is one of the most complex to properly tune machine learning algorithms. This is due to the large amount of hyperparameters which can be chosen. While the amount of nodes in the input layer and output layer are problem specific, the amount of hidden layers and nodes per hidden layer can be chosen arbitrarily, as well as the activation functions. Other issues are selecting the starting weights of the connections and the necessity to properly scale the input data (e.g. normalizing to a standard deviation of one and zero mean) (Hastie et al., 2008).

The architecture of artificial neural networks is very flexible, which is why they come in many different forms other than the MLP. One of the many possibilities is using local connectivity instead of connecting the neurons to all neurons in the subsequent layer (Hastie et al., 2008), as done in case of the convolutional neural network, which is often used in image processing. Other possibilities are the use of Bayesian Neural Networks which could be used to treat uncertainties (Bhat and Prosper, 2005), and the Tensor Basis Neural Network from Ling et al. (2016b).

Examples in turbulence modelling Various examples of the use of artificial neural networks can be found in the context of turbulence modelling. The most relevant work is presented in Ling et al. (2016b), where the tensor basis derived in Pope (1975) as presented in (2.41) is translated into an artificial neural network architecture. The input invariants form the input layer of the neural network, after which this input traverses a number of hidden layers. The final hidden layer has ten nodes, which multiply ten nodes in an additional input layer containing the basis tensors. This results in an expression for b_{ij} equivalent to that presented in (2.40). The work from Ling et al. (2016b) will be discussed in more detail in Section 3.5, since it forms an important baseline for this research. The major advantage of the tensor basis approach in Ling et al. (2016b) is the fact that predictions for b_{ij} are Galilean invariant, which will mean that the algorithm is likely to generalize well when using data-sets from different flows for training and testing. Reproducing results from this paper is worthwhile however, since there are still significant discrepancies for b_{ij} with respect to the DNS data, and it is not clear how well these predictions will propagate into the flow field, since the figures in this paper do not show much detail.

One of the first works regarding the application of machine learning in turbulence modelling is the one from Milano and Koumoutsakos (2002). Here, a comparison is made between using an artificial neural network and proper orthogonal decomposition to reconstruct near wall turbulent flow. In Duraisamy et al. (2015) an inverse problem is solved to find a corrective term introduced in the $k-\omega$ turbulence model, in order to improve the modelling of transition. This corrective term is learned by an artificial neural network, to be able to correct an unseen flow field later. In Singh et al. (2016) a similar approach is used to learn a corrective term in the Spalart-Allmaras turbulence model, to improve predictions of the lift/drag/stall angle of airfoils.

2.5.2 Tree based methods

Decision Tree Decision trees base their output on a number of 'if-then' tests, which maximize the information gained at each split (Russell and Norvig, 2010; Ling and Templeton, 2015). A common algorithm used to create decision trees is the CART algorithm (Classification And Regression Tree), which is used in for example the `python` machine learning library `scikit-learn` (Scikit-learn, 2017). This algorithm can both be used to create classification and regression trees. The algorithm chooses for each input variable the optimal splitting point. The optimal combination is chosen by evaluating a so called loss function, which can for example be the sum of squared error with respect to the training data in the case of regression. The optimal combination which minimizes the specified loss function is then used as splitting variable and splitting point. In each splitted domain, this splitting step is repeated (Hastie et al., 2008). An important parameter which can be changed is the tree depth (the amount of layers which the decision tree has). Making the decision tree deeper enables it to approximate a certain function with more accuracy, but at the risk the model does not generalize well, i.e. overfitting of the data (Ling and Templeton, 2015).

Decision trees have a lot of benefits: models are fast to construct, are relatively easy to interpret, deal well with outliers, and perform internal feature selection (Hastie et al., 2008). Pre-processing of the data is not necessary. Downsides are overfitting and learning non-linear boundaries (Ling and Templeton, 2015). High variance can be a problem, as small changes in the data might lead to a completely different series of splits in the tree because of its hierarchical nature. Using many trees (ensemble methods) and taking the resulting average can be used to reduce this variance. Another limitation is its lack of smoothness. Techniques exist to overcome this problem such as using Multivariate Adaptive Regression Splines (MARS) (Hastie et al., 2008).

AdaBoost Decision Tree In the AdaBoost procedure, multiple decision trees are trained on modified sets of the data. The first decision tree is trained on unmodified data. Afterwards, the training data is given weights, with higher weights for data points which were predicted incorrectly, and lower weights for the data points which were predicted correctly. This modified training set is used for the next decision tree. This process can be repeated a number of times. The final prediction is given by a weighted vote of all the different decision trees. The AdaBoost procedure can be used on different kinds of machine learning algorithms, but is popular to use with decision trees: it increases their predictive performance, while keeping the beneficial properties mentioned in the previous paragraph (Hastie et al., 2008).

Random Forest Similarly to the AdaBoost Decision Tree method, random forests use a collection of decision trees, and let them vote on the outcome. Bootstrap samples are taken from the training data, on which the individual decision trees are trained. In contrast with normal decision trees, some random features are selected to choose the best splitting variable from (instead of the entire feature set). The random selection of splitting features is done in order to reduce the correlation between the trees. This in turn reduces the variance of the random forest prediction (Hastie et al., 2008).

Examples in turbulence modelling Some examples can be found in literature for the use of the AdaBoost decision tree algorithm in turbulence modelling. The first example is its use in Ling and Templeton (2015). Here it was one of the algorithms trained to predict areas of high RANS uncertainty. This was done by training the algorithm to identify three error metrics using LES/DNS data where linear eddy-viscosity models break down: the existence of a negative eddy-viscosity, a larger degree of anisotropy, and the deviation of the eddy-viscosity when one would use the cubic eddy-viscosity model from Craft et al. (1996) instead. In the same paper the random forest algorithm was used, which showed better performance compared to the AdaBoost decision tree algorithm, possibly due to the fact that it is more robust against noisy training inputs and responses. Another example of the use of the AdaBoost decision tree algorithm is Singh et al. (2017), where an inverse problem was solved to infer a corrective term in the Spalart-Allmaras turbulence model for the modelling of shock-boundary layer interactions.

The random forest algorithm has been the most popular algorithm so far for the use in turbulence modelling. Besides its aforementioned use in Ling and Templeton (2015), it has been used in Ling et al. (2017b) to improve predictions for the Reynolds stress anisotropy tensor of a jet-in-crossflow case. One limitation from this paper is that only the eigenvalues are predicted, and the eigenvectors are not taken in account at all. In for example Wang et al. (2017b), Wu et al. (2016), Wang et al. (2017a) the random forest algorithm is used to correct the Reynolds stress from RANS simulations by predicting discrepancies of the eigenvalues, eigenvectors, and turbulent kinetic energy with respect to LES/DNS data. Discrepancies of the eigenvalues are learned by making use of the barycentric map, discrepancies of the eigenvectors are learned by using three Euler angles defining the orientation of the eigenvectors with respect to the coordinate system. While the three aforementioned papers show potential with regards to the application of machine learning to RANS turbulence modelling, one limitation is that it is not demonstrated whether this approach of correcting the eigenvectors is really Galilean invariant. Only one flow case is considered in Wang et al. (2017a), which means it is hard to say how well this idea generalizes when training and testing on different flows.

2.5.3 Support Vector Machines

Background The support vector machines (SVM) algorithm is a maximum margin separator: it creates the largest possible decision boundary. An example of such a maximum margin separator which could be created when making a decision boundary for two different classes is presented in figure 2.4. Normally a linear separating hyperplane is made, but when this is not able to separate the classes, the SVM method creates higher dimensional spaces in which such a separation hyperplane can be made by using kernel functions (Russell and Norvig, 2010).

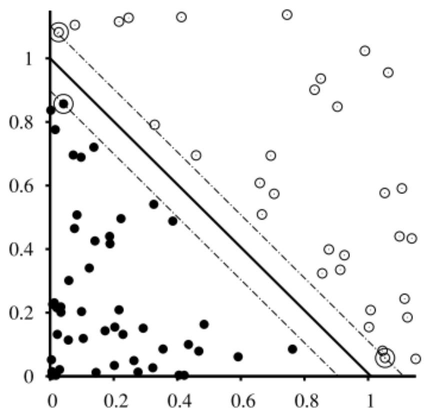


Figure 2.4: Support Vector Machine, maximum margin separator (reproduced from Russell and Norvig (2010))

Support vector machines do not use the entire data-set to construct the separating hyperplane. The decision boundary is defined by a small subset of the training data, called the 'support vectors'. This makes this method more efficient than for example k-nearest neighbours algorithms. The support vectors can be selected by looking at points which lie on the wrong side of the decision boundary (classification), or by looking at the residuals of the training points (regression).

Examples in turbulence modelling The use of support vector machines is relatively unexplored in the context of turbulence modelling. One example where it is used is the aforementioned paper from Ling and Templeton (2015), to predict areas of high RANS uncertainty. In this paper it is noted that a downside of the algorithm is the importance of feature selection before using it in comparison with for example the random forest algorithm. In this paper the cross-validated classifier error was reduced by 33% after performing feature selection.

2.5.4 Other (stochastic) kernel-based methods

Finally, two other algorithms as found in literature will be discussed, which are able to take in account confidence bounds for the parameters of interest. The first one is found in Tracey et al. (2013), and is based on the kernel regression algorithm. A kernel function, denoted by $k(x_i, x_j)$ determines the closeness of two feature vectors. In Tracey et al. (2013) a squared-exponential is selected as the kernel, $k(x_i, x_j) = \exp(-(x_i - x_j)^2/\tau)$, where τ is a bandwidth parameter. In the case of kernel regression, the expected value of the response, y_*, as a function of a test input x_* is given by evaluating

$$y_*(x_*) = \sum_i^n w(x_*, x_i) y_i, \tag{2.44}$$

where $w(x, x_i)$ is evaluated for the test input given n training samples, using

$$w(x_*, x_i) = \frac{k(x_*, x_i)}{\sum_{j=1}^{n} k(x_*, x_j)}. \tag{2.45}$$

In Tracey et al. (2013) this expression is used to predict the mean of the response, $\mu(x_*)$. Furthermore, the standard deviation is predicted by evaluating

$$\sigma(x_*) = \sqrt{\sum_{i}^{n} w_i \left(y_i - \mu\left(x_*\right)\right)^2}. \tag{2.46}$$

The bandwidth parameter τ was then selected using k-fold cross validation, by looking at the sum of the log-likelihoods of the held-out data. The algorithm was trained on the anisotropy tensor eigenvalues using the barycentric map.

Another algorithm used in the context of turbulence modelling is the Gaussian process regression algorithm. In Zhang and Duraisamy (2015) it is used to learn a corrective term in the k-equation for a turbulence modelling scenario, and to learn a corrective term for modelling transition. These corrective terms are found by solving an inverse problem using Bayesian inversion. In Duraisamy et al. (2015) and Parish and Duraisamy (2016) Gaussian processes are used in a similar fashion to learn a corrective term for turbulence modelling and transition modelling.

In the case of Gaussian process regression, the predictions from the algorithm are also stochastic, i.e. what is the probability of the response $\mathbf{y}_*$ given the training data $\mathbf{y}$, or $p(\mathbf{y}_*|\mathbf{y})$. It is assumed that the training data has a certain amount of Gaussian noise ϵ, i.e. $y = f(x) + \epsilon$, where ϵ has a mean of zero, and a variance of σ_n^2. Using Gaussian process regression can be beneficial when the response is noisy, or when confidence bounds are wanted for the predictions.

The kernel function $k(x_i, x_j)$, also called the covariance function, is evaluated among all combinations of training data and test data. When having n training points, and n_* test points, four different matrices are set up containing the covariances between the different points: $K(X, X)$ is a $n \times n$ matrix, containing the covariances of the training data, $K(X, X_*)$ is a $n \times n_*$ matrix containing the covariances between the training data and test data, and $K(X_*, X_*)$ is a $n_* \times n_*$ matrix containing the covariances of the test data. It can be shown, see for example Rasmussen and Williams (2006), that the conditional probability of a set of test points $\mathbf{y}_*$ is given by:

$$\begin{aligned} p\left(\mathbf{y}_*|\mathbf{x}, \mathbf{y}, \mathbf{x}_*\right) \sim \mathcal{N}\big(&K(X, X_*)\left[K(X, X) + \sigma_n^2\mathbf{I}\right]^{-1}\mathbf{y}, \\ &K(X_*, X_*) - K(X_*, X)\left[K(X, X) + \sigma_n^2\mathbf{I}\right]^{-1}K(X, X_*)\big). \end{aligned} \tag{2.47}$$

As can be seen, both kernel regression and Gaussian processes are quite costly, which is a major limitation of these approaches. In both cases all training data needs to be stored. This is not such a nice property, since this means all data needs to be transferred to each machine on which predictions need to be made. Furthermore these methods will suffer relatively a lot from the 'curse of dimensionality' (Wang et al., 2017b), making them less suitable when introducing more variables.

Chapter 3

Methodology for the Machine Learning Framework

In this chapter the methodology used to implement the machine learning framework will be presented. To start off, preliminary investigations into existing machine learning frameworks for turbulence modelling as found in literature will be presented in Section 3.1. From these preliminary investigations the most appropriate method for further development was selected. An overview of this selected framework, using a random forest with a tensor basis, will be given in Section 3.2. Input features used for the machine learning algorithms will be presented in Section 3.3. Extrapolation metrics which can be used as a priori confidence parameters for the machine learning predictions will be discussed in Section 3.4. The machine learning algorithms used in this research will be discussed afterwards. These are the Tensor Basis Neural Network (TBNN) from Ling et al. (2016b) as presented in Section 3.5, and the newly developed Tensor Basis Decision Tree (TBDT) and Tensor Basis Random Forest (TBRF) as presented in Section 3.6 and Section 3.7 respectively. Post-processing implemented for the results of the TBRF algorithm will be discussed in Section 3.8, and finally the numerical set-up used for the RANS simulations and propagating the predictions by the machine learning algorithms will be discussed in Section 3.9.

3.1 Preliminary investigations into existing machine learning frameworks

In this section, an overview will be given on different approaches which have been investigated for the use of machine learning algorithms in RANS turbulence modelling scenarios. This includes the general lay-out of how supervised machine learning algorithms can be used, different approaches of predicting the Reynolds stress, and different approaches of propagating predictions of the Reynolds stress into the flow field.

3.1.1 General overview

Figure 3.1 presents a flow diagram of the general lay-out how supervised learning algorithms can be used in a turbulence modelling scenario. A supervised learning algorithm always includes two different phases: the training phase, in which a map is created from the input to the response; and the testing phase, in which predictions are made for a previously unseen test case. Multiple options are available for the response on which the machine learning algorithm is trained. Three different options for the response, based on quantities related to the Reynolds stress, will be discussed in Section 3.1.2. Just obtaining a prediction for the response in an unseen test case is not useful. It should be investigated whether the predicted response really leads to improved quantities of interest, in this case for example the entire flow field or a specific quantity such as the reattachment point. The methods for propagating the predicted response into the flow field will be discussed in Section 3.1.3.

It should be noted that in theory there are two approaches for using machine learning algorithms which predict quantities related to the Reynolds stress: a corrective approach, and an iterative approach, from which the first one is used in this thesis.

In the corrective approach, the machine learning algorithm is used only once to give an estimate of the true value of for example b_{ij} or R_{ij}, after which it is propagated. It essentially creates a map using input features from RANS, to the corresponding ground truth (DNS/LES). It is not a replacement for the turbulence model, but should be seen as a supplement to increase accuracy of a given RANS simulation.

In the iterative approach, the machine learning algorithm could be trained using input features from DNS/LES data, as well as the corresponding responses (b_{ij}) from DNS/LES data. The map which the machine learning algorithm creates from the input features to the anisotropy tensor, could be used to replace the turbulence model. The machine learning algorithm would be called at each iteration in the flow solver, illustrated by the dashed line in Figure 3.1. This idea is not explored further in this thesis (one reason being the difficulty this method would bring with respect to numerical stability), but would be an interesting topic for further research.

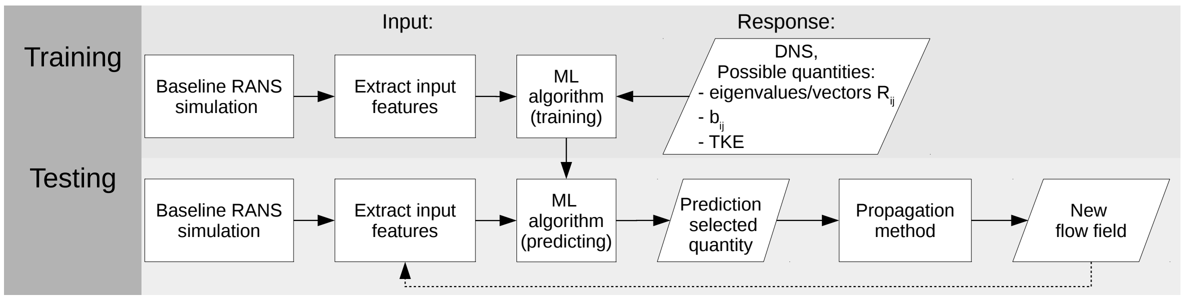


Figure 3.1: Flow diagram for the general lay-out of different machine learning frameworks in literature

3.1.2 Reynolds stress representation

A preliminary investigation is done on the Reynolds stress representation in order to compare different approaches used so far in literature for machine learning. As was shown in Equation (2.32), the Reynolds stress can be separated in terms of the turbulent kinetic energy (representing the amplitude), its eigenvalues (representing the shape), and its eigenvectors (representing the orientation). Three different approaches can be identified by looking at literature:

1. Predict new values for the Reynolds stress eigenvalues. Substitute these eigenvalues back in the original RANS prediction for the Reynolds stress, and propagate this into the flow field. This is for example done in Tracey et al. (2013).

2. Predict new values for the normalized Reynolds stress anisotropy tensor b_{ij}. This would be equivalent to predicting new values for the eigenvectors and eigenvalues of the Reynolds stress, see (2.32). The turbulent kinetic energy can be modelled by making use of a (modified) k-equation when propagating the predictions into the flow field. This is the approach used in Ling et al. (2016b).

3. Predict new values for the Reynolds stress eigenvalues and eigenvectors, and predict new values for the turbulent kinetic energy. This gives a complete description for the Reynolds stress when substituting these predictions in (2.32), which can then be propagated into the flow field. This is for example done in Wang et al. (2017a).

From these approaches, the first approach is the easiest to implement. Only scalar quantities are predicted (the eigenvalues), which means that predictions are the same in any frame of reference. The orientation of the Reynolds stress is completely neglected however, resulting in limited improvement over the baseline RANS simulation.

For the other two approaches, invariance is more difficult since the orientation of the frame of reference needs to be taken in account when predicting tensor quantities. One solution is using a tensor basis as proposed in Ling et al. (2016b). The other approach would be correcting the Reynolds stress eigenvalues, and correcting the Reynolds stress eigenvectors by looking for example at the Euler angles as done in Wang et al. (2017a). Here the angles of the RANS and DNS eigenvectors were calculated with respect to the x-y-z coordinate system. Afterwards the discrepancy between RANS and DNS was learned in order to make corrections on a new flow case. The set of eigenvectors of the Reynolds stress, denoted by $\mathbf{V}$, are orthogonal to each other since the Reynolds stress is a symmetric tensor. This means one can use a rotation matrix $\mathbf{Q}$ to go from the x-y-z coordinate system to the unit eigenvectors. Representing the x-y-z- coordinate system by the identity matrix $\mathbf{I}$, this means that

$$\mathbf{V} = \mathbf{Q}\mathbf{I}, \tag{3.1}$$

from which the rotation matrix $\mathbf{Q}$ can be obtained. The angles between the x-y-z coordinate system and $\mathbf{V}$, denoted by α_x, α_y, and α_z are calculated using the relations (Slabaugh, 1999):

$$\alpha_x = \mathtt{atan2}(\mathbf{Q}_{2,1}, \mathbf{Q}_{2,2}), \tag{3.2}$$

$$\alpha_y = \mathtt{atan2}\left(-\mathbf{Q}_{2,0}, \sqrt{\mathbf{Q}_{2,1}^2 + \mathbf{Q}_{2,2}^2}\right), \tag{3.3}$$

$$\alpha_z = \mathtt{atan2}(\mathbf{Q}_{1,0}, \mathbf{Q}_{0,0}), \tag{3.4}$$

where `atan2` is the four-quadrant inverse tangent.

The three angles between $\mathbf{V}$ and the x-y-z coordinate system can be learned by the machine learning algorithm as a function of the input features. Afterwards it can make predictions for the eigenvector orientations of an unseen data-set. The values for the angles are substituted back into three rotation matrices, yielding

$$\mathbf{V} = \begin{bmatrix} \cos(\alpha_z) & -\sin(\alpha_z) & 0 \\ \sin(\alpha_z) & \cos(\alpha_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\alpha_y) & 0 & \sin(\alpha_y) \\ 0 & 1 & 0 \\ -\sin(\alpha_y) & 0 & \cos(\alpha_y) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha_x) & -\sin(\alpha_x) \\ 0 & \sin(\alpha_x) & \cos(\alpha_x) \end{bmatrix} \mathbf{I} \tag{3.5}$$

It is clear however that this representation depends on the frame of reference, making this approach unsuitable when training and testing on different flow cases. As of writing this thesis, some authors of the paper from Wang et al. (2017a) have come up with alternative methods to describe the orientation of the eigenvectors which might be independent on the frame of reference, see Wu et al. (2017a).

A weakness of the third approach is that the turbulent kinetic energy is predicted by the machine learning algorithm as well. The transport equation for the turbulent kinetic energy is not used at all anymore in this case. While it is reasonable to assume that features based on for example the strain rate and rotation rate of the flow relate to the anisotropy tensor (see the derivation of turbulence models such as Pope (1975)), predicting the turbulent kinetic energy purely based on data as well reduces the turbulence model even more to a black box model. The approach in Ling et al. (2016b) is more elegant, in the sense that it relies more on physical understanding of the turbulence problem, and that it could potentially be used in an iterative manner, see the discussion in Section 3.1.1.

3.1.3 Propagation of the Reynolds stress

After obtaining predictions for either the Reynolds stress eigenvalues, the Reynolds stress anisotropy tensor, or the full Reynold stress, this prediction should be propagated in order to obtain an improved flow field. Different approaches exist in order to do this, which were investigated to select the most suitable one to be implemented in the OpenFOAM framework.

It was attempted to obtain a solver which is able to propagate predictions for the Reynolds stress tensor or Reynolds stress anisotropy tensor by adjusting turbulence models in OpenFOAM and the `simpleFOAM` solver from OpenFOAM. Implementations were first tested by propagating the DNS data for the relevant quantities. In case this proved to be stable enough, an attempt was made to propagate predictions from a machine learning algorithm.

Approach 1 In the first approach, the Reynolds stress is adjusted and directly injected in the momentum equation. One example where this is done is Wang et al. (2017a), where the `simpleFoam` solver in OpenFOAM is adjusted, in order to directly calculate a velocity field from a given Reynolds stress field. No turbulence model is used, and no transport equations for turbulent quantities need to be calculated, since the Reynolds stress can directly be put in the momentum equation. The authors mention that numerical stability could be an issue for propagations, but that it should be trivial to solve by including artificial diffusion. As of writing this thesis however, some of the same authors mention that this propagation is still an unsolved challenge, and that they are working on a condition number to increase stability (Wu et al., 2017b). In Emory and Iaccarino (2014) the Reynolds stress is perturbed and propagated as well. An extra source term is added to the momentum equation, corresponding to the perturbation of the Reynold stress. It is noted that this source term can be non-smooth, which is why a filtering algorithm was implemented for stability. Using this first approach means that the turbulent kinetic energy would need to be predicted, or for example taken from the RANS simulation.

In order to implement the first approach, the steps from Wang et al. (2017a) were followed. In the `simpleFoam` solver, the momentum equation was adjusted to accept a given Reynolds stress field.

The steady-state, incompressible Reynolds Averaged Navier-Stokes equations without any body forces can be written down as:

$$\bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = \frac{\partial}{\partial x_j}\left[-\bar{p} + \nu\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) - R_{ij}\right]. \tag{3.6}$$

In the SIMPLE algorithm, Equation (3.6) is iteratively solved by separating part containing the pressure gradient term from the part containing the velocity. In the `simpleFoam` solver, the pressure equation is solved in `pEqn.H`. The file `UEqn.H` contains the rest of the momentum equation. In this file, the term `turbulence->divDevReff(U)` calculates:

$$\frac{\partial}{\partial x_j}\left((\nu + \nu_t)\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right)\right). \tag{3.7}$$

The `turbulence->divDevReff(U)` function references to another file, which depends on the turbulence model which is used. For example, the calculation of this term will be different when using a linear eddy viscosity model which uses the Boussinesq approximation, versus a Reynolds stress transport model which calculates R_{ij} based on extra transport equations. The OpenFOAM implementation of Reynolds stress transport models can be used as a template to modify the `UEqn.H` file. In the `UEqn.H` file the `turbulence->divDevReff(U)` term can be removed and replaced directly with the implementation from the Reynolds stress transport turbulence model.

In the `ReynoldsStress.C` OpenFOAM source file, which is a template for the OpenFOAM

Reynolds stress transport models, the `divDevReff(U)` term effectively solves:

$$\frac{\partial}{\partial x_j} R_{ij} + \frac{\partial}{\partial x_j}\left(\nu_t \frac{\partial \bar{u}_i}{\partial x_j}\right) - \frac{\partial}{\partial x_j}\left((\nu+\nu_t)\frac{\partial \bar{u}_i}{\partial x_j}\right) - \frac{\partial}{\partial x_j}\left(\nu \frac{\partial \bar{u}_i}{\partial x_j}\right). \tag{3.8}$$

As can be seen, the turbulent viscosity appears in this expression. Although at first sight this is strange, since the Reynolds stress transport model does not use a turbulent viscosity, it is included in there for stability reasons. The third term in this equation is solved implicitly, while the rest is solved explicitly. This implicit term has the possibility to increase the diagonal dominance of the matrix system to be solved, increasing stability of the solver (Greenshields, 2015), which is the reason the turbulent viscosity is still included in this equation.

The `divDevReff(U)` term from the Reynolds stress transport model template was copied to modify the `simpleFoam` solver. The implementation in the C++ code is presented in appendix B. Different implementations were attempted: one without using ν_t in Equation (3.8), which seems to be the approach used by Wang et al. (2017a) since they mention no turbulence models were involved in their propagations, and an implementation with using ν_t, possibly increasing stability of the solver. In the latter case ν_t was calculated using either the $k-\omega$ turbulence model, or using the Launder-Sharma turbulence $k-\epsilon$ model, which is the low Reynolds number implementation for the $k-\epsilon$ turbulence model.

Approach 2 In the second approach, as found in Ling et al. (2016b), predictions for the Reynolds stress anisotropy tensor are implemented in the momentum equation, and in the production term for the turbulent kinetic energy. This is done using the corrective approach, i.e. the machine learning algorithm is called only once. The benefit of this method is that the turbulent kinetic energy does not need to be predicted or taken from the RANS simulation.

In order to implement the second approach, similar modifications can be made to the `simpleFoam` solver as for the first approach. Instead of inserting the Reynolds stress, the Reynolds stress anisotropy tensor is used in combination with the turbulent kinetic energy as calculated by the turbulence model. This means that Equation (3.8) changes to:

$$\frac{\partial}{\partial x_j}\left(\frac{2}{3}k\delta_{ij} + 2kb_{ij}\right) + \frac{\partial}{\partial x_j}\left(\nu_t \frac{\partial \bar{u}_i}{\partial x_j}\right) - \frac{\partial}{\partial x_j}\left((\nu+\nu_t)\frac{\partial \bar{u}_i}{\partial x_j}\right) - \frac{\partial}{\partial x_j}\left(\nu \frac{\partial \bar{u}_i}{\partial x_j}\right). \tag{3.9}$$

A turbulence model needs to be selected to be used for calculating the turbulent kinetic energy and ν_t. Options which were attempted are the $k-\omega$ turbulence model, and the Launder-Sharma $k-\epsilon$ turbulence model, both suitable for low Reynolds number applications. The production term of the k-equation was modified as specified in Ling et al. (2016b). The production term, see Equation (2.20), changes to:

$$\mathcal{P} = -\left(\frac{2}{3}k\delta_{ij} + 2kb_{ij}\right)\frac{\partial \bar{U}_i}{\partial x_j} \tag{3.10}$$

Implementation of the second approach in the C++ code of the OpenFOAM framework is presented in Appendix B.

Approach 3 In the third approach which was attempted, which has not been presented in literature before, the solver blends machine learning predictions for the Reynolds stress anisotropy tensor with predictions from a standard RANS turbulence model. This approach will be explained in more detail in Section 3.9, as this was selected to be the most suitable method and forms an essential part of the machine learning framework.

Other approaches Another approach which will not be further elaborated on, was using Reynolds stress models already implemented in OpenFOAM as a baseline, e.g. the LRR model from Launder et al. (1975). In this case the transport equation for R_{ij} was removed, and substituted with the Reynolds stress field from the machine learning algorithm. The value for k was then directly obtained from R_{ij}, and a transport equation for ϵ was solved. It did not prove to be more stable than the previously mentioned approaches, so further efforts were not pursued.

Approaches found in literature which were not implemented inlude some of the iterative approaches. For example, in Tracey et al. (2013) the machine learning model predicting the Reynolds stress eigenvalues is coupled to the flow solver in an iterative fashion. At each iteration, the eigenvalues are corrected by making use of the barycentric map after which they are injected back in the Reynolds stress tensor. The approach used in this paper is not very consistent, since the input features for training come from the RANS simulation, and the response from DNS (see the discussion in Section 3.1.1).

3.1.4 Preliminary conclusions

From the preliminary investigations, it was decided that the approach from Ling et al. (2016b), where b_{ij} is represented using a tensor basis is the most promising for further research. This is based on the discussion in the end of Section 3.1.2, and preliminary results which will be presented in Section 5.1 and Section 5.2.2. It was decided an useful contribution to the field of machine learning in turbulence modelling would be implementing the tensor basis from Pope (1975) in a random forest algorithm, similar to the TBNN from Ling et al. (2016b). The majority of the remainder of this thesis will focus on the implementation and results from this adjusted random forest, which will be called the Tensor Basis Random Forest (TBRF).

It was decided that the third approach for propagating the Reynolds stress is the most suitable. This approach, which will be called the continuation solver in the remainder of this report, will be elaborated upon in Section 3.9. Results from which it was decided this approach is the most suitable will be presented in Section 5.2.1.

3.2 Overview of the TBRF framework

A flow diagram of the Tensor Basis Random Forest (TBRF) framework used in this thesis is shown in Figure 3.2. Two different phases can be identified: the training phase and the testing phase.

In the training phase the machine learning algorithm is supplied with invariant input features from a RANS flow field. The $k - \omega$ turbulence model is used for the baseline simulation from which these invariant input features are derived. The machine learning algorithm is also supplied with the response (the anisotropy tensor, b_{ij}) corresponding to the input features. This response can be seen as the ground truth, and comes from either DNS or highly resolved LES. Each mesh cell of the baseline RANS simulation yields a set of input features, and a response which is linearly interpolated from the DNS/LES data.

There is no restriction on the amount of different flow cases which can be used to train the machine learning algorithm. Computational effort increases with increasing amount of samples for the TBRF algorithm however ($\mathcal{O}(n \texttt{ log } n)$), which means random samples are taken from the total available data in order to keep the training time within reasonable bounds.

In the testing phase predictions for the anisotropy tensor are made for an unseen flow case. Input features for the machine learning algorithm are extracted from the baseline RANS simulation. The machine learning algorithm then gives an estimated value for the response, which will be denoted by $b_{ij,ML}$. The response can be propagated in the flow field using the continuation solver, which will be further discussed in Section 3.9. When the machine learning algorithm works properly, the value for $b_{ij,ML}$ should be an improvement over the anisotropy as predicted by the $k - \omega$ model, which then hopefully leads to a more accurate flow field.

As can be seen the approach used for the TBRF framework is a corrective approach as explained in the previous section, where the machine learning algorithm is used only once to give an estimate on the true value of b_{ij}.

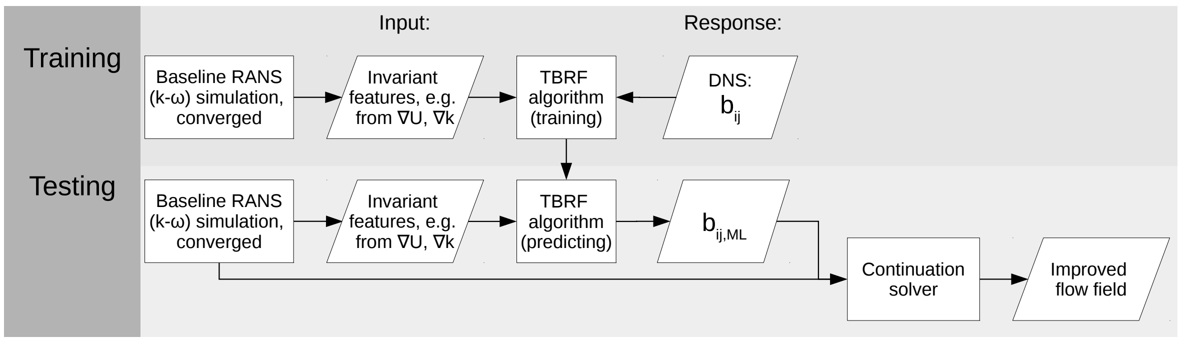

Training
Testing
Input:
Response:
Baseline RANS (k-ω) simulation, converged
Invariant features, e.g. from ∇U, ∇k
TBRF algorithm (training)
DNS: b_ij
Baseline RANS (k-ω) simulation, converged
Invariant features, e.g. from ∇U, ∇k
TBRF algorithm (predicting)
b_ij,ML
Continuation solver
Improved flow field

Figure 3.2: Flow diagram for the TBRF framework

3.3 Input Features and pre-processing

When sticking to the approach from Ling et al. (2016b), which in turn is based on Pope (1975), five features serve as input to the machine learning algorithm, which are all based on the rates of strain in the flow. These five features form a mathematical basis from which all ten basis tensor series coefficients are derived. It is possible to include more features to predict the basis tensor series scalar coefficients however, while still using the integrity basis from $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$ for the basis tensors. Instead of $g^{(m)}(\lambda_1, ..., \lambda_5)$ the tensor basis coefficients can be a function of k different inputs, $g^{(m)}(\lambda_1, ..., \lambda_k)$.

It was decided not to only use the five input invariants from Ling et al. (2016b). Only using these five invariants as features for training the machine learning algorithm does not yield enough information as observed during preliminary investigations. It was found that the features are too similar, due to the fact that they mirror each other, effectively leading to only 2 significantly different features in for example the square duct flow case.

The random forest will be using the full set of invariants derived from $\hat{\mathbf{S}}$, $\hat{\mathbf{\Omega}}$, and $\nabla\mathbf{k}$, which were obtained from Wang et al. (2017a). In order to use the turbulent kinetic energy gradient it is first normalized using $\sqrt{k}/\epsilon$, and then transformed to an antisymmetric tensor:

$$\mathbf{A_k} = -\mathbf{I} \times \nabla\mathbf{k}. \tag{3.11}$$

Predictions for the anisotropy still adhere to Galilean invariance as long as the extra input features themselves are Galilean-invariant. For example, one can take a certain reference frame and transform it with a rotation/reflection matrix $\mathbf{Q}$. First one has the input features from Ling et al. (2016b): $\lambda(\hat{\mathbf{S}}, \hat{\mathbf{\Omega}})$. These transform to $\lambda(\mathbf{Q}\hat{\mathbf{S}}\mathbf{Q}^T, \mathbf{Q}\hat{\mathbf{\Omega}}\mathbf{Q}^T)$ in the new frame of reference, and are not changed with respect to the original frame of reference since the input features are invariant to the proper orthogonal group. Additionally, features are added using the turbulent kinetic energy gradient tensor $\mathbf{A_k}$. These input features should not change in a different reference frame, i.e. $\lambda(\mathbf{A_k}) = \lambda(\mathbf{Q}\mathbf{A_k}\mathbf{Q}^T)$ should be true. When this is the case, one can ensure (2.37), which in this case would translate to:

$$g^{(m)}\left(\lambda\left(\hat{\mathbf{S}}, \hat{\mathbf{\Omega}}, \mathbf{A_k}\right)\right) = g^{(m)}\left(\lambda\left(\mathbf{Q}\hat{\mathbf{S}}\mathbf{Q}^T, \mathbf{Q}\hat{\mathbf{\Omega}}\mathbf{Q}^T, \mathbf{Q}\mathbf{A_k}\mathbf{Q}^T)\right)\right). \tag{3.12}$$

This means that in different inertial frames the predictions for $g^{(m)}$ will be the same, after which the integrity basis (the basis tensors $T_{ij}^{(m)}$) ensures that b_{ij} has its correct orientation with respect to the reference frame.

Furthermore nine extra scalar features which are more physically interpretable such as the wall-distance based Reynolds number are used, which were obtained from Wu et al. (2016). All features which are used are presented in Table 3.1, where feature set 1 (FS1) is based on the mean strain rate tensor and mean rotation rate tensor, feature set 2 (FS2) are the extra features when including the turbulent kinetic energy gradient, and feature set 3 (FS3) are the features obtained from Wu et al. (2016). For the features in FS3 an normalization factor is included, whereas the tensors in FS1 and FS2 are already normalized using k and ϵ. From the available features the ones with a low variance ($<1\times10^{-4}$) were discarded, since these either did not contain any information at all, or showed very spurious patterns. See Appendix F for more details.

Table 3.1: Features used for the machine learning algorithms, from Wang et al. (2017a) and Wu et al. (2016). For features with an * all cyclic permutations of labels of anti-symmetric tensors need to be taken in account. For FS1 and FS2 the trace of the tensor quantites is taken.

Feature set:	**Features:**	**Normalization factor:**	**Explanation:**
FS1	$\hat{\mathbf{S}}^2$, $\hat{\mathbf{S}}^3$, $\hat{\mathbf{\Omega}}^2$, $\hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}$, $\hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}^2$, $\hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}\hat{\mathbf{\Omega}}\hat{\mathbf{S}}^2$	-	invariant set based on $\hat{\mathbf{S}}$ and $\hat{\mathbf{\Omega}}$
FS2	${\mathbf{A_k}}^2$, ${\mathbf{A_k}}^2\hat{\mathbf{S}}$, ${\mathbf{A_k}}^2\hat{\mathbf{S}}^2$, ${\mathbf{A_k}}^2\hat{\mathbf{S}}\mathbf{A_k}\hat{\mathbf{S}}^2$, $\hat{\mathbf{\Omega}}\mathbf{A_k}$, $\hat{\mathbf{\Omega}}\mathbf{A_k}\hat{\mathbf{S}}$, $\hat{\mathbf{\Omega}}\mathbf{A_k}\hat{\mathbf{S}}^2$, $\hat{\mathbf{\Omega}}^2\mathbf{A_k}\hat{\mathbf{S}}^*$, $\hat{\mathbf{\Omega}}^2\mathbf{A_k}\hat{\mathbf{S}}^{2*}$, $\hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}\mathbf{A_k}\hat{\mathbf{S}}^{2*}$	-	added invariants when including $\nabla\mathbf{k}$
FS3	$\frac{1}{2}(\|\|\hat{\Omega}_{ij}\|\|^2 - \|\|\hat{S}_{ij}\|\|^2)$	$\|\|\hat{S}_{ij}\|\|^2$	Ratio of excess rotation rate to strain rate
	k	$\frac{1}{2}\bar{u}_i\bar{u}_i$	turbulence intensity
	$\min\left(\frac{\sqrt{k}d}{50\nu}, 2\right)$	-	wall-distance based Reynolds number
	$\bar{u}_k\frac{\partial p}{\partial x_k}$	$\sqrt{\frac{\partial p}{x_j}\frac{\partial p}{x_j}\bar{u}_i\bar{u}_i}$	pressure gradient along streamline
	$\frac{k}{\epsilon}$	$\frac{1}{\|\|S\|\|}$	ratio of turbulent time scale to mean strain time scale
	$\sqrt{\frac{\partial p}{x_i}\frac{\partial p}{x_i}}$	$\frac{1}{2}\rho\frac{\partial\bar{u}_k^2}{\partial x_k}$	ratio of pressure normal stresses to shear stresses
	$\bar{u}_i\frac{\partial k}{\partial x_i}$	$\|R_{jk}S_{jk}\|$	ratio of convection to production of TKE
	$\|\|R_{ij}\|\|$	k	ratio of total to normal Reynolds stresses
	$\left\|\bar{u}_i\bar{u}_j\frac{\partial\bar{u}_i}{\partial x_j}\right\|$	$\sqrt{\bar{u}_l\bar{u}_l\bar{u}_i\frac{\partial\bar{u}_i}{\partial x_j}\bar{u}_k\frac{\partial\bar{u}_k}{\partial x_j}}$	non-orthogonality between velocity and its gradient

The machine learning algorithm essentially creates a map from a RANS flow field—which might contain structural discrepancies due to modelling errors—to a corresponding DNS field for b_{ij} which serves as the ground truth. While in a conventional turbulence model the five invariants derived in Pope (1975) might be sufficient to calculate the basis tensor series coefficients, the machine learning algorithm has to predict the anisotropy tensor using only one "iteration", based on input features from a RANS simulation (i.e. which can be structurally different from features had they been derived from the ground truth, DNS). It was therefore decided to include more than just the five features from Pope (1975), possibly allowing the algorithm to learn more about areas where RANS simulations show large discrepancies with respect to DNS.

While it might seem inconsistent with the approach from Pope (1975) to use basis tensors derived from $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$, but using invariant features using more flow quantities, this is not necessarily so when looking at turbulence models created in the past. For example, one of the additional features supplied to the machine learning algorithm, which is not based on $\hat{S}_{ij}$ or $\hat{\Omega}_{ij}$, is the wall-distance based Reynolds number. The wall distance is used in some of the most frequently used turbulence models, such as the $k-\omega$ SST turbulence model from Menter (1993), or the Spalart-Allmaras turbulence model from Spalart and Allmaras (1992). In the transport equations for k and ω there are plenty of parameters which are empirically determined from experiments, and furthermore both equations depend on relations of each other, e.g. the transport equation for ω in Menter (1993) contains terms based on k and $\nabla \mathbf{k}$, and the transport equation for k contains terms based on ω.

Before using the features, they are normalized to have a mean of zero, and a standard deviation (σ) of one. Furthermore features in the flow field which fall above a threshold of $\pm 5 \cdot \sigma$ are seen as outliers and removed. Scaling the input features is especially important when using the data for training an artificial neural network, since this determines the scaling of the weights in the bottom layer of the network, which has a large influence on the quality of the algorithm (Hastie et al., 2008). After training the machine learning algorithms, the standard deviation and the mean of the training data are saved. This is then later used to scale the test data with.

3.4 A Priori Confidence

Given a training data-set $\mathbf{X}$ and a test data-set $\mathbf{X}_*$, one can estimate the probability that samples from the test data-set were present in the distribution of samples from the training data-set. This was used to set up a priori confidence parameters for machine learning in Ling and Templeton (2015) and Wu et al. (2016). In Wu et al. (2016), a correlation study was done for two confidence parameters and the error on the barycentric map. The two confidence parameters analysed were the Mahalanobis distance, and a Kernel Density Estimate (KDE) distance. When these confidence parameters have higher values, this means that the test data is less similar to the training data, i.e. there is more extrapolation, which often will result in higher errors. This can be useful for identifying why a given prediction shows higher errors in a certain domain of the flow field.

In case of the Mahalanobis distance, it is assumed that the data adheres to a multivariate Gaussian distribution. This of course does not necessarily need to be the case, as there is no reason to assume the training data will not have multiple modalities. For the training data, the mean of the features over all samples is calculated ($\overline{\mathbf{X}}$), as well as the covariance matrix $\mathbf{\Sigma}$. The Mahalanobis distance of a certain test sample $\mathbf{X}_{*,i}$, denoted by M_i, is then given by:

$$M_i = \sqrt{\left(\mathbf{X}_{*,i} - \overline{\mathbf{X}}\right)^T \mathbf{\Sigma}^{-1} \left(\mathbf{X}_{*,i} - \overline{\mathbf{X}}\right)}. \tag{3.13}$$

The Mahalanobis distance can be normalized in order to yield a range from 0 (no extrapolation) to 1 (high extrapolation). The value for the normalized Mahalanobis distance at a test sample i, $M_{norm,i}$, is calculated by evaluating

$$M_{norm.,i} = 1 - f_{dm}, \tag{3.14}$$

where f_{dm} is the fraction of training points with a larger Mahalanobis distance than the test point.

The assumptions used by the KDE distance are less strict than that of the Mahalanobis distance. Instead of assuming the entire data-set adheres to a multivariate Gaussian distribution, it assumes that each sample comes from a multivariate Gaussian distribution. Each sample of the training data serves as the location for a Gaussian kernel. The sum of all the kernels then forms the approximation for the probability density function of the training data. The Kernel Density Estimate evaluated for a certain test sample, KDE_i, is given by:

$$\text{KDE}_i = \frac{1}{n\tau}\sum_{j=1}^{n} K\left(\frac{\mathbf{X}_{*,i} - \mathbf{X}_j}{\tau}\right), \tag{3.15}$$

where n is the amount of training samples, τ is the bandwidth of the Gaussian kernel, and K denotes the Gaussian kernel. In order to get a distance out of the KDE distribution, it is compared to a multivariate uniform distribution spanning the space between the minimum and maximum values from the training samples. The KDE distance, denoted by D_{KDE} is given by:

$$D_{KDE,i} = 1 - \frac{\text{KDE}_i}{\text{KDE}_i + 1/\prod_j\left(\max([\mathbf{X}]_j) - \min([\mathbf{X}]_j)\right)}, \tag{3.16}$$

where $[\mathbf{X}]_j$ denotes the j-th feature of data-set $\mathbf{X}$.

For estimating the bandwidth τ multiple approaches are available. One which will be used is called Scott's rule (Scott, 1992). It was found however that this estimation for the bandwidth does not always yield good results. In some cases the estimated bandwidth is too narrow, which results in only predicting high uncertainty at very local areas in the flow, yielding not much usable information.

3.5 Tensor Basis Neural Network (TBNN)

In Ling et al. (2016b) a neural network architecture was modified in such a way that it accepts an extra input consisting of the tensor basis series derived in Pope (1975). The idea of the basis tensor series is directly translated into the artificial neural network architecture, which is shown in Figure 3.3. The five invariants form the first input layer, shown in red. This input is transformed through a number of hidden layers, resulting in an output in the final hidden layer, shown in orange. This layer will consist of ten nodes, since ten scalar coefficients multiply the ten basis tensors. These ten basis tensors are inserted in the tensor input layer, shown in blue. The final hidden layer and tensor input layer are merged to form the predictions for b_{ij}.

The implementation of the TBNN was obtained from Ling et al. (2017a). The neural network is implemented in `python` using the `lasagne` library, which serves as a wrapper for the lower-

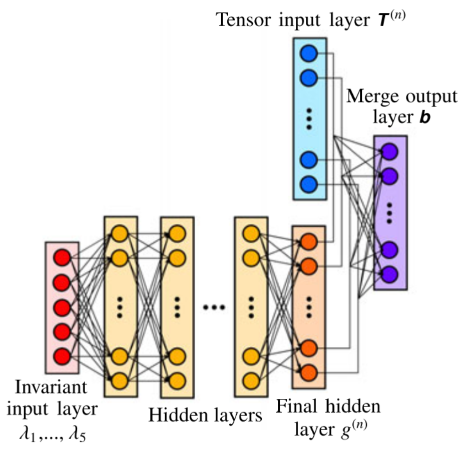

Tensor input layer $T^{(n)}$
Merge output layer b
Invariant input layer $\lambda_1, ..., \lambda_5$
Hidden layers
Final hidden layer $g^{(n)}$

Figure 3.3: Tensor Basis Neural Network architecture (reproduced from from Ling et al. (2016b))

level `Theano` library.

Training of artificial neural networks is not straightforward due to the large amount of hyperparameters which need to be tuned. The amount of hidden layers and nodes per layer were optimized in (Ling et al., 2016b), and were used for the TBNN in this thesis. The TBNN uses a leaky ReLU activation function, which is an adjusted version of the rectified linear unit activation function and has a slightly negative slope for negative values. The TBNN is trained using back-propagation, in which the weights of the neural network are optimized using a gradient-descent algorithm. More specifically, the TBNN is trained using the Adam algorithm, which is a stochastic gradient descent method introduced in Kingma and Ba (2015). Various parameters can be changed for training, such as the learning rate, the learning rate decay, and the mini-batch size used for the stochastic gradient descent algorithm. The minimum learning rate was set to 2.5×10^{-5}, the learning rate which was found to be optimal in Ling et al. (2016b). The mini-batch size was set to 1,000; which was found to be an appropriate size for convergence while at the same time keeping the training time reasonably low.

In order to avoid overfitting, part of the data-set is kept separate in a validation data-set, which was checked every ten epochs for the validation error. Early stopping was used, which terminates training of the algorithm when the training error reduces, but the validation error starts increasing. Since the validation error as a function of the epoch has a noisy behaviour, a moving average of five samples was taken to determine when the early stopping should be activated. From the available training data 80% was used for training, and 20% was used for the validation data-set. Since training of the TBNN is random in the sense that the starting weights of the network are randomly chosen, the TBNN was trained five times from which the best performing network was selected by looking at the validation error.

3.6 Tensor Basis Decision Tree (TBDT)

Random forests consist of a collection of individual decision trees. In case of the Tensor Basis Random Forest (TBRF), this is the Tensor Basis Decision Tree (TBDT), for which its implementation is explained in this section. Decision trees base their predictions on a series of if-then tests on the input. Multiple decision tree algorithms exist, of which the CART (Classification And Regression Tree) algorithm is a common example. This algorithm serves as the baseline for the Tensor Basis Decision Tree algorithm, and will therefore be briefly explained in Section 3.6.1, after which the derivation of the TBDT algorithm is presented. Afterwards an overview of measures taken to increase efficiency of the TBDT algorithm will be presented in Section 3.6.2. Finally, Section 3.6.3 will present a flow diagram of the TBDT algorithm as a summary.

3.6.1 Derivation of the TBDT algorithm

In the training phase of the CART decision tree, the feature space is repeatedly split into two separate areas. In both of these areas a constant value is used to approximate the response. The data is split using a certain splitting feature j, where a splitting value s is selected which is the threshold for dividing the data into the two separate areas which will be called bins. The data-set is denoted as $\mathbf{X} \in \mathbb{R}^{N\times p}$, where N is the total amount of observations in the data-set, and p is the amount of features of which the data-set consists. The two bins in which the data is split will be denoted by R_L (left bin) and R_R (right bin), and are given by

$$R_L(j,s) = \{\mathbf{X} \,|\, [\mathbf{X}]_j \leq s\} \quad R_R(j,s) = \{\mathbf{X} \,|\, [\mathbf{X}]_j > s\}. \tag{3.17}$$

This splitting is done as efficiently as possible, meaning the least amount of mismatch between the response, and the constant value approximation of the response in both bins is required. This minimization problem can be written as

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_L(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_R(j,s)}(y_i-c_2)^2\right], \tag{3.18}$$

where y_i denotes the response. In case of regression c_1 and c_2 are found by taking the average of the response in R_L and R_R respectively, effectively minimizing the variance in both bins. When the optimum splitting feature are value are found (e.g. by evaluating all combinations of features and values present in the data-set), a decision tree node is created, in which the optimum splitting feature and value are saved. This splitting process starts with all data, resulting in the creation of the root node. The splitting process can then be repeated with the data in R_L and R_R, resulting in the creation of more nodes, after which this process can be repeated a large number of times until a stopping criterion is reached. This can be a maximum tree depth (amount of layers which the decision tree has), or by looking at the amount of data a node has. When a node has less samples than a certain threshold, this node becomes a terminal or leaf node. Growing a tree until every sample has its own leaf node is called a fully grown tree.

Later, when testing the decision tree on a different data-set, given data points can be compared to the splitting features and values as prescribed in the tree nodes. Each data point will follow a cascade of decisions as given by the decision tree nodes, eventually resulting in a prediction for the response in a leaf node. As the CART algorithm assumes a constant value in R_L and R_R, the resulting function which fits the data will be piecewise constant.

The TBDT algorithm is comparable with the CART decision tree algorithm, but instead of approximating the response with a constant value in R_L and R_R, the algorithm finds a constant value for the tensor basis coefficients $g^{(m)}$ (see (2.40)) in both bins. In each of the two bins in which the data is split, this tensor basis approximation of the given anisotropy tensors from DNS should give the minimum total mismatch possible. This is achieved by solving

$$\min_{j,s}\left[\min_{g_L^{(m)}\in\mathbb{R}^{10}}\sum_{x_n\in R_L(j,s)}\left|\left|\sum_{m=1}^{10}\mathbf{T}_n^{(m)}g_L^{(m)}-\mathbf{b}_n\right|\right|^2+\min_{g_R^{(m)}\in\mathbb{R}^{10}}\sum_{x_n\in R_R(j,s)}\left|\left|\sum_{m=1}^{10}\mathbf{T}_n^{(m)}g_R^{(m)}-\mathbf{b}_n\right|\right|^2\right], \tag{3.19}$$

where the subscripts n indicate different observations (i.e. at all the grid points of the different flow cases), $\mathbf{b}_n$ is the response of the anisotropy tensor which comes from the DNS/LES data-sets, and $g_L^{(m)}$ and $g_R^{(m)}$ indicate the tensor basis series coefficients in the left and right bin respectively. The norm of the tensor is defined as the Euclidean norm:

$$||c_{ij}|| := \sqrt{\sum_{i,j} c_{ij}^2}. \tag{3.20}$$

Since the basis tensors $\mathbf{T}^{(m)}$ and training features $\mathbf{X}$ come from the RANS simulations, the responses $\mathbf{b}_n$ have been interpolated on the RANS grid from the DNS/LES data-sets. Finding the tensor basis coefficients $g_L^{(m)}$ and $g_R^{(m)}$ which minimize the two terms inside of the square brackets in (3.19) amounts to solving two least squares problems. This is repeated for every combination of j and s until an optimum is found.

Just like for the CART algorithm, the procedure in Equation 3.19 is repeated at each node which is created as a result of the splitting process, until a stopping criterion is reached. In case of the TBDT algorithm, the tensor basis series coefficients $g^{(m)}$ are constant in each bin. This does not mean that the function fitting the data is piecewise constant, since the basis tensors $\mathbf{T}^{(m)}(\hat{\mathbf{S}},\hat{\mathbf{\Omega}})$ vary for each observation.

By using matrix and vector notations for $\mathbf{T}_n^{(m)}$, $g^{(m)}$, and $\mathbf{b}_n$ as

$$\hat{\mathbf{T}}_n=\begin{pmatrix}T_{n,11}^{(1)} & T_{n,11}^{(2)} & \cdots & T_{n,11}^{(10)}\\ T_{n,12}^{(1)} & T_{n,12}^{(2)} & \cdots & T_{n,12}^{(10)}\\ \vdots & \vdots & \ddots & \vdots\\ T_{n,33}^{(1)} & T_{n,33}^{(2)} & \cdots & T_{n,33}^{(10)}\end{pmatrix}, \quad \mathbf{g}=\begin{pmatrix}g^{(1)}\\ g^{(2)}\\ \vdots\\ g^{(10)}\end{pmatrix}, \quad \mathbf{b}_n=\begin{pmatrix}b_{n,11}\\ b_{n,12}\\ \vdots\\ b_{n,33}\end{pmatrix}, \tag{3.21}$$

the total minimization problem J which needs to be solved in order to minimize the mismatch between the response and the tensor basis approximation can be written as

$$J=||\hat{\mathbf{T}}_1\mathbf{g}-\mathbf{b}_1||^2+||\hat{\mathbf{T}}_2\mathbf{g}-\mathbf{b}_2||^2+...+||\hat{\mathbf{T}}_N\mathbf{g}-\mathbf{b}_N||^2. \tag{3.22}$$

This needs to be solved with the observations in both R_L and R_R. This is a least squares problem, for which the coefficients $\mathbf{g}$ can be found by setting the derivative of J with respect to $\mathbf{g}$ to zero, which yields

$$\mathbf{g} = (\hat{\mathbf{T}}_1^T\hat{\mathbf{T}}_1 + ... + \hat{\mathbf{T}}_N^T\hat{\mathbf{T}}_N)^{-1}(\hat{\mathbf{T}}_1^T\mathbf{b}_1 + ... + \hat{\mathbf{T}}_N^T\mathbf{b}_N), \tag{3.23}$$

which can be solved within R_L and R_R in order to obtain $g_L^{(m)}$ and $g_R^{(m)}$.

The problem of finding the tensor basis coefficients $g^{(m)}$ is not well-posed: there is a risk that the coefficient values will be sensitive to the data. Regularization is added to the problem to avoid this. In this case, a type of L^2 regularization is added, which is called ridge regression. Expression (3.22) changes to

$$J = ||\hat{\mathbf{T}}_1\mathbf{g} - \mathbf{b}_1||^2 + ||\alpha\mathbf{I}\,\mathbf{g}||^2 + ||\hat{\mathbf{T}}_2\mathbf{g} - \mathbf{b}_2||^2 + ||\alpha\mathbf{I}\,\mathbf{g}||^2 + ... + ||\hat{\mathbf{T}}_N\mathbf{g} - \mathbf{b}_N||^2 + ||\alpha\mathbf{I}\,\mathbf{g}||^2, \tag{3.24}$$

where α is a constant which multiplies the identity matrix $\mathbf{I}$. It can be shown that this will yield

$$\mathbf{g} = (\hat{\mathbf{T}}_1^T\hat{\mathbf{T}}_1 + \Gamma\mathbf{I} + ... + \hat{\mathbf{T}}_N^T\hat{\mathbf{T}}_N + \Gamma\mathbf{I})^{-1}(\hat{\mathbf{T}}_1^T\mathbf{b}_1 + ... + \hat{\mathbf{T}}_N^T\mathbf{b}_N) \tag{3.25}$$

instead of (3.23) in order to find the basis tensor series coefficients, where the coefficients Γ are equal to α^2. The value for Γ can be treated as a hyperparameter of the tensor basis decision tree.

3.6.2 Increasing the efficiency of the TBDT algorithm

As the algorithm iterates over all the features and samples to find the optimum splitting feature and value, it is important that this step is implemented efficiently in order to save training costs. At the very beginning of training the TBDT, the least squares matrices presented in (3.25) are set up once. At each splitting step the samples are binned into a certain node. This is kept track of for each sample, by having an array for each sample indicating the path it follows (i.e. a 0 is added in case of a split into the left node, a 1 is added in case of a split into the right node). When solving (3.25) at a certain node, the indices of the samples present in the node are easily retrieved this way, and only the relevant least-squares submatrices are used.

In order to make splitting of the data at a given splitting value more efficient, the data is sorted in advance. The sorting algorithm requires $\mathcal{O}(n \log n)$ operations after which the data is easily split into two parts, instead of $\mathcal{O}(n^2)$ operations which would be the case without the pre sorting.

Instead of the brute-force approach for finding the splitting feature and value, the possibility to use an optimization algorithm instead is implemented as well. This option can be enabled at a certain threshold of the amount of samples to speed up the computations. The Brent optimization algorithm was chosen, as this offers a good trade-off between speed (by using inverse quadratic interpolation) and robustness (by falling back on the golden section search in the worst case) (Brent, 1973).

In Figure 3.4 an example is shown where data for a square duct is split at the very first

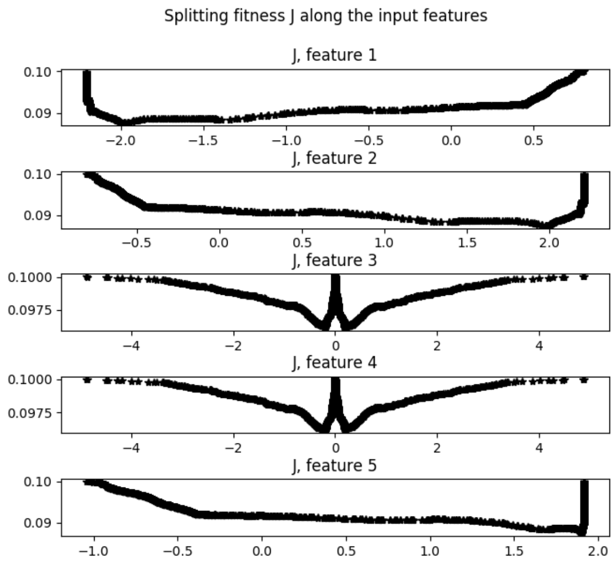


Figure 3.4: Example of the splitting fitness, taken from the square duct flow case with 5 different features

node (root node), using 5 possible features. The splitting fitness is plotted (see (3.22)), which means a lower value is better as it gives a lower mismatch to the b_{ij} from DNS/LES. As can be seen the fitness is relatively smooth and not very noisy, making the use of an optimization algorithm instead of a brute force search viable. In this case either feature 1, feature 2, or feature 5 should be selected for creating the split, since these features give an equally low value at either $[\mathbf{X}]_1 \approx -2.0$, $[\mathbf{X}]_2 \approx 2.0$ or $[\mathbf{X}]_5 \approx 1.89$. In Appendix C.2 further reflection on the use of the Brent optimization algorithm will be given.

It was further attempted to speed up the training phase of the TBDT algorithm by implementing parallel feature selection, where the splitting fitness for each feature is calculated in parallel. The first attempt was done using the `joblib` library, and a second attempt was done using the `multiprocessing` library from `python`. A simplified version of the decision tree algorithm was implemented, and the speed-up for using multiple cores was kept track of. The test code can be found in Appendix C. Unfortunately the parallelization did not result in any significant speed-up, and the bottleneck causing this is yet to be identified. Running the test code on 1, 2, 3, and 4 cores resulted in the calculations taking 106.7, 92.3, 92.6, and 121.4 seconds respectively. The cores were monitored, from which it was observed that no significant amount of time passed until the calculations started on each core, and cores were utilized close to 100%. By using intermediate output from the script it was found that all programs indeed ran in parallel. Implementation of the parallel feature selection was not pursued further. Instead it is trivial to train the trees in a random forest in parallel, by starting multiple `python` kernels and using different random seeds to ensure the trained TBDT's are different.

3.6.3 Flow diagram of the TBDT algorithm

A flow diagram which summarizes the TBDT algorithm is presented in Figure 3.5. In this figure, standard processes are represented by squares, decisions by the diamond shapes, and data by the parallelograms.

The main routine is indicated by the grey box, in which the optimal splitting feature and value is calculated and saved. In the flow diagram the brute-force approach is represented, where all combinations of features and samples are evaluated. In the case of using an optimization algorithm as explained in the previous subsection, this would only be done for a number of samples as chosen by the specific algorithm. As can be seen, after deciding which split is the most optimal, two nodes (left and right nodes) are created, each containing part of the data present in the parent node. The algorithm then checks whether these nodes adhere to certain criteria for further splitting, namely whether the maximum tree size is not reached yet, and whether there are enough samples left for further splitting. When this is the case, the given node is split into two child nodes, which are added to a queue for further splitting. This process is repeated until no nodes are present in the queue.

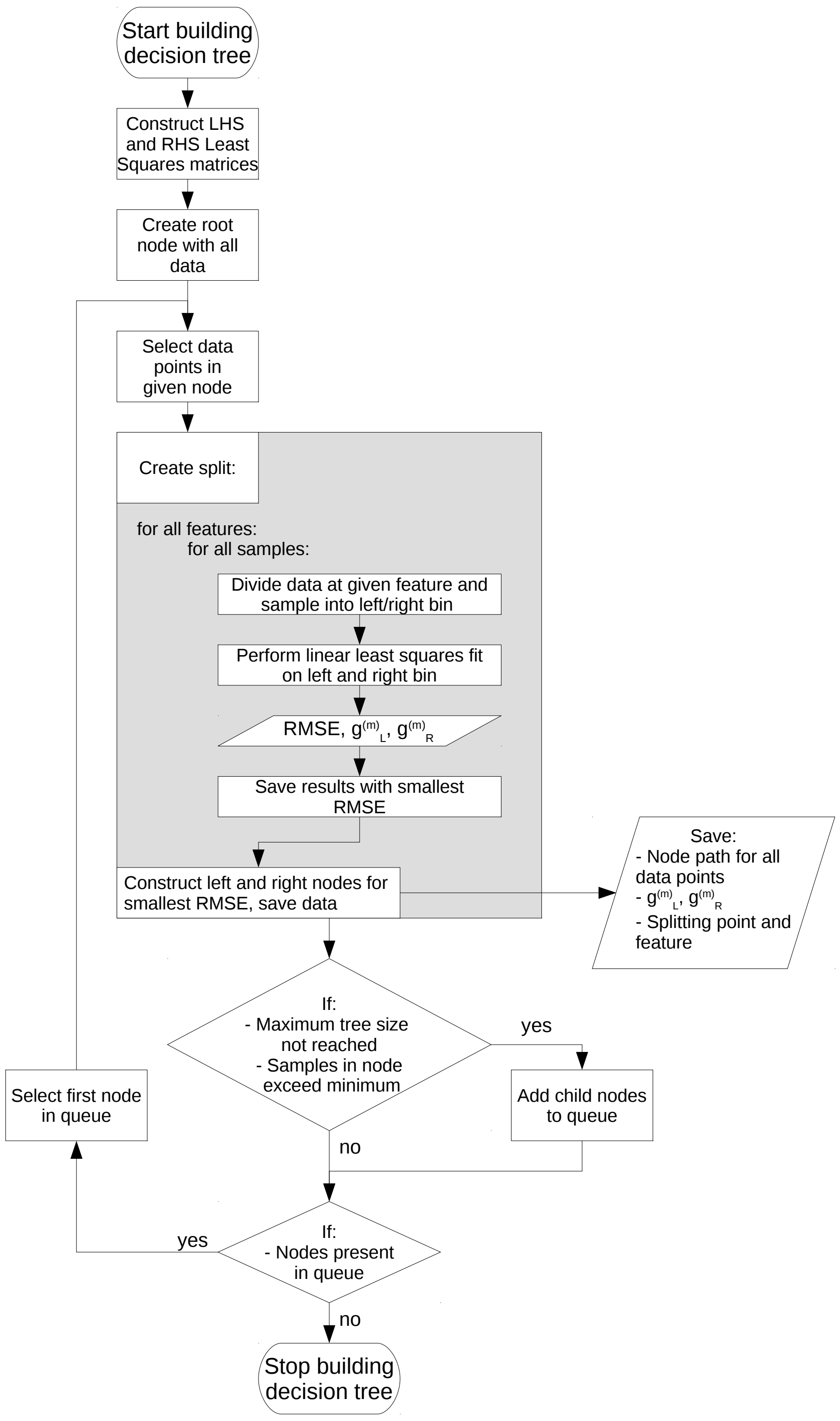

Start building decision tree
Construct LHS and RHS Least Squares matrices
Create root node with all data
Select data points in given node
Create split:
for all features:
for all samples:
Divide data at given feature and sample into left/right bin
Perform linear least squares fit on left and right bin
RMSE, $g^{(m)}_L$, $g^{(m)}_R$
Save results with smallest RMSE
Construct left and right nodes for smallest RMSE, save data
Save:
- Node path for all data points
- $g^{(m)}_L$, $g^{(m)}_R$
- Splitting point and feature
If:
- Maximum tree size not reached
- Samples in node exceed minimum
yes
Add child nodes to queue
no
Select first node in queue
yes
If:
- Nodes present in queue
no
Stop building decision tree

Figure 3.5: Flow diagram for the TBDT algorithm

3.7 Tensor Basis Random Forest (TBRF)

In the tensor basis random forest, multiple tensor basis decision trees are combined. The idea behind the random forest algorithm is that the individual trees are trained on so called bootstrap aggregated data-sets created from the original data-set. This procedure is called bagging. It means that from the original data-set, random samples are taken with replacement, leading to some of the original samples being present in the bagged data-set multiple times, and some not at all. Bagging works well when combining a number of high-variance, low-bias estimators, such as the decision tree. By averaging many noisy, but unbiased models, the variance of the prediction is reduced.

It can be shown that the variance of the average of a set of B predictors, $\sigma^2_{avg.}$, is given by:

$$\sigma^2_{avg.} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{3.26}$$

when assuming that the different predictors are identically distributed and have a positive pairwise correlation ρ, with a variance of σ^2 (Hastie et al., 2008). As can be seen, the second term in this Equation vanishes when using a large amount of decision trees. Furthermore, it can be seen that the variance of the average gets smaller as the correlation between the decision trees reduces.

Reducing the correlation between the decision trees is done by introducing some randomness in the individual trees: at each split, not all features, but a subset of the available features is used for creating the optimal split. This subset is selected randomly from all available features. The amount of features present in this subset, which will be denoted by $N_{f,s}$, can be treated as a hyperparameter. A commonly used default value for $N_{f,s}$ when using the standard random forest algorithm for regression is the total amount of features divided by three.

Several benefits can be identified in using random forest algorithms. First of all, the available data can be divided into a training data-set and validation data-set in a natural way. Since random samples are taken from the available with replacement until the original size of the data-set is reached, a number of samples will not be present in the training data-set for each decision tree, called out-of-bag (OOB) samples. For each observation a random forest can be constructed in which this observation was not used for training, which can then be used to give a validation error, or OOB error. During training of the trees this OOB error can be observed to determine when training can be stopped. Using the OOB error is similar to performing N-fold cross validation (Hastie et al., 2008).

Using the OOB error means that one of the hyperparameters of the random forest, the amount of trees, does not have to be set beforehand but can be optimized during training. One other hyperparameter, the amount of samples per leaf, can be set to one in order to have maximum tree diversity, and was observed to perform the best for the TBRF algorithm as well. The TBDT algorithm has the possibility to include regularization, but it was observed that this can be set to a very low value, and instead post-processing the results with an outlier filter, which will be descibed in Section 3.8.1. Next to having few hyperparameters to tune, the random forest algorithm is relatively straightforward to implement, and can be trained in parallel easily by growing a few trees at the same time, or performing the search for the optimal splitting feature and value described in Section 3.6 in parallel.

3.8 Post-Processing

In this section the methods used for post-processing the predictions by the TBRF will be described. This includes a median outlier filter, and enforcing realizability of the predicted anisotropy tensor.

3.8.1 Median outlier filter

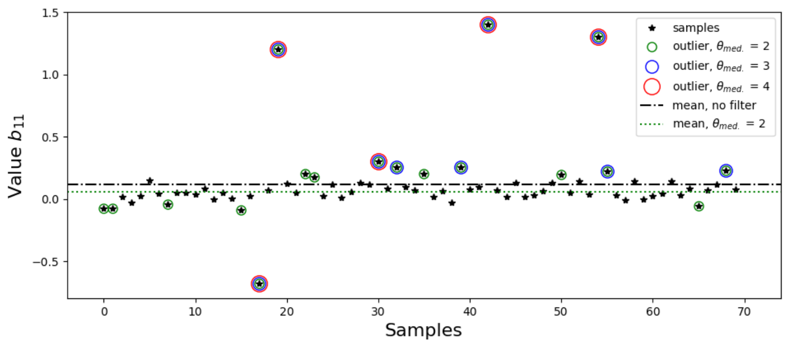
Each decision tree in the tensor basis random forest yields a prediction. During testing of the tensor basis decision tree it was found that some of the predictions of the anisotropy tensor lie way out of their possible bounds. While regularization can fix a large part of this problem, introducing too much regularization was also seen to decrease the accuracy in other areas of the flow. It was decided to implement an outlier filter on the individual decision tree predictions to increase accuracy of the tensor basis random forest.

A median outlier filter was implemented to detect the outliers. The expression

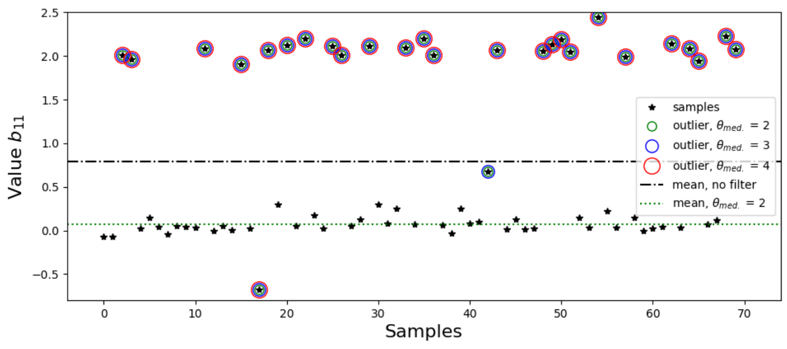
$$\frac{|y - \text{med}(y)|}{\text{med}\left(|y - \text{med}(y)|\right)} > \theta_{med.} \tag{3.27}$$

is evaluated to check whether samples from the signal fall above a certain threshold, $\theta_{med.}$, and where med denotes the median. In case the expression is true, the sample is seen as an outlier and removed. Although the outlier filter threshold can be seen as a hyperparameter of the random forest, it can be tuned after the trees have been grown, since it does not affect the tree splitting process unlike for example including regularization. It was observed a threshold ranging from 2 to 3 is appropriate for most test cases, and that almost no regularization is necessary when using the filter.

As an example, predictions for the b_{11} component of the anisotropy tensor are given in Figure 3.6a for 70 different TBDT's. For different thresholds of $\theta_{med.}$, points are marked which the median filter has identified as outliers. The green, blue, and red circles mark the outliers as identified with $\theta_{med.} = 2$, 3, and 4 respectively. Four major outliers can be seen, from which three have values much higher than most of samples. These major outliers have a relatively large influence when one would calculate b_{11} by taking the mean over all the TBDT's. Instead, when for example using the outlier filter with $\theta_{med.} = 2$, the mean gives a much better description of the predictions. In order to show the robustness of the implemented filter, 25 random samples have been shifted upwards with a value of 2 in Figure 3.6b. Taking the mean of the unfiltered signal in this case would lead to a strongly non-realizable anisotropy tensor where b_{11} is far above its limit of $\frac{2}{3}$. As can be seen the mean as given by the signal filtered with $\theta_{med.} = 2$ gives a good estimate for b_{11}, since the majority of the samples are still giving reasonable predictions. Of course when the majority of the predictions would be far off, the median filtered signal would also be inaccurate. A possible measure which could be implemented to avoid this would be looking at the realizability bounds for the components of b_{ij}.



(a) Predictions for b_{11}



(b) Predictions for b_{11}, perturbed

Figure 3.6: Illustration of the median filter identifying outliers

This outlier filter is used for the predictions of the decision trees at a certain location in the flow field. Results are further smoothened spatially using a Gaussian filter, before they are propagated into the flow field. For this the `ndimage.filters` packages are used from the `scipy` library. These packages include filters used for multi-dimensional image processing, such as a Gaussian filter and a median filter.

3.8.2 Enforcing realizability

Results can be forced within bounds of realizability by shifting the eigenvalues of the anisotropy tensor and the individual components within an acceptable range. The same approach as used in Ling et al. (2016b) was used for this, as obtained from Ling et al. (2017a). First the diagonal components of the anisotropy are checked whether they are larger than $\frac{1}{3}$.

When this is not the case, the minimum diagonal component is shifted to $\frac{1}{3}$ by scaling it with the appropriate factor. The other diagonal components are also scaled with this factor to maintain a zero trace tensor. Afterwards, the Cauchy-Schwarz inequality is checked for the off-diagonal components. In case one of the off-diagonal components is greater than Equation 2.31 allows with respect to its corresponding diagonal components, it is shifted to the upper limit.

Afterwards the limits on the eigenvalues are checked. First an eigenvalue decomposition on the anisotropy tensor is done. The eigenvalues are checked whether they adhere to the bounds as prescribed by the realizability map from Lumley (1979):

$$\lambda_1 \leq (3|\lambda_2| - \lambda_2)/2, \quad \lambda_1 \leq 1/3 - \lambda_2, \tag{3.28}$$

where λ_1 and λ_2 are the two largest eigenvalues of b_{ij}. When this is not the case, all eigenvalues are scaled with a factor bringing the eigenvalues within the prescribed bounds. Afterwards the anisotropy tensor is reconstructed with the matrix containing the eigenvectors and the matrix containing the new eigenvalues.

The last step where the limits of the eigenvalues are checked can undo the constraints from the first step. The function which enforces realizability should therefore be called iteratively for convergence to a realizable state.

3.9 Numerical set-up and propagating the anisotropy tensor

The open source CFD toolbox OpenFOAM is used in order to calculate the RANS flow fields. The $k - \omega$ turbulence model is used, together with the `simpleFoam` solver (Semi-Implicit Method for Pressure Linked Equations) with a second order accurate numerical scheme.

Propagating the prediction of the anisotropy tensor into the flow field affects numerical stability of the solver. A custom solver was implemented in the OpenFOAM framework, using the `simpleFoam` solver as a baseline, together with modifications in the $k - \omega$ turbulence model. The `simpleFoam` solver is uncoupled, and separately solves the momentum equation and pressure equation.

The incompressible Navier-Stokes equations without any body forces can be written down as

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = \frac{\partial}{\partial x_j} \left[-\bar{p} + \nu S_{ij} - R_{ij} \right]. \tag{3.29}$$

The prediction for b_{ij} from the machine learning algorithm ($b_{ij,ML}$) is introduced into the momentum equation. This is done by using a blending parameter γ in the custom solver to determine how much of the anisotropy tensor as predicted by the machine learning is used, and how much from original turbulence model is left out. This results in a modified expression for the momentum equation:

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = \frac{\partial}{\partial x_j} \left[-\bar{p} + \nu S_{ij} - (1-\gamma)\, R_{ij,EV} - \gamma\, R_{ij,ML} \right], \tag{3.30}$$

where $R_{ij,EV}$ is the Reynolds stress as calculated by using the eddy viscosity hypothesis, and $R_{ij,ML}$ is the Reynolds stress as calculated by using $b_{ij,ML}$:

$$R_{ij,EV} = -\nu_t \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right), \quad R_{ij,ML} = \frac{2}{3} k \, \delta_{ij} + 2k \, b_{ij,ML}. \tag{3.31}$$

This blending parameter is slowly increased during the simulation. This idea is based on the continuation method, where first a simple problem is solved, after which the solution is used as the initial condition for a more difficult problem, see e.g. Knoll and Keyes (2004). In this case the initial condition comes from the flow field as calculated by the standard $k - \omega$ turbulence model.

The turbulent kinetic energy in the expression for $R_{ij,ML}$ comes from a k-equation which is solved at each iteration by a slightly modified version of the $k - \omega$ turbulence model. The production term in the k-equation and ω-equation is modified to slowly introduce $b_{ij,ML}$. The full production term without any simplifications is given by:

$$\mathcal{P} = -R_{ij} \frac{\partial \bar{u}_i}{\partial x_j}. \tag{3.32}$$

The same blending parameter γ as used in the momentum equations can be used, to combine the Reynolds stress from the eddy viscosity hypothesis with the Reynolds stress using $b_{ij,ML}$. The full expression for the production term is then calculated using

$$\mathcal{P} = -(1 - \gamma) \, R_{ij,EV} \frac{\partial \bar{u}_i}{\partial x_j} - \gamma \, R_{ij,ML} \frac{\partial \bar{u}_i}{\partial x_j}. \tag{3.33}$$

A linear ramping function is used for γ, which is a function of the current time step t. The maximum value of γ (γ_{max}) needs to be specified, the starting value of γ (γ_0), the total amount of iterations (t_{total}), and the amount of iterations before the end at which γ remains constant ($t^\star$). The blending parameter at a specified time step is then calculated by evaluating

$$\gamma = \frac{\gamma_{max} - \gamma_0}{t_{total} - t^\star} \cdot t + \gamma_0. \tag{3.34}$$

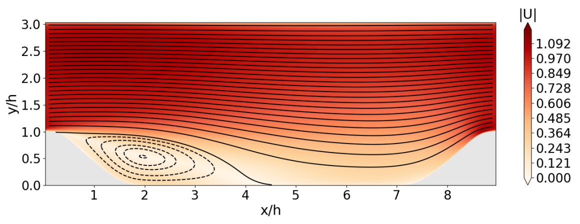
Too high values for γ can result in numerical instability of the flow solver. The maximum value of γ depends on the quality of the predictions of b_{ij} and the flow case itself. It was observed that typically the value for γ can be increased to about 0.8.

Chapter 4

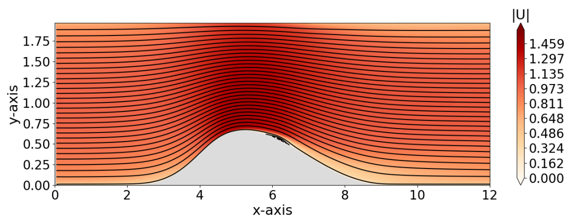
Flow Cases Used in the Machine Learning Framework

Five different flow cases are used in the framework to train and test the machine learning algorithms. For all flow cases DNS data or highly resolved LES data was available. The different cases are presented in Figure 4.1. For the periodic hills five different Reynolds numbers are available in the DNS data-sets, ranging from $Re = 700$ to $Re = 10,595$. For the square duct multiple Reynolds numbers are available as well, with a total of sixteen ranging from $Re = 1,100$ to $Re = 3,500$. For the converging diverging channel the only Reynolds number available is $Re = 12,600$, for the curved backward facing step $Re = 13,700$, and for the backward facing step $Re = 5,100$.

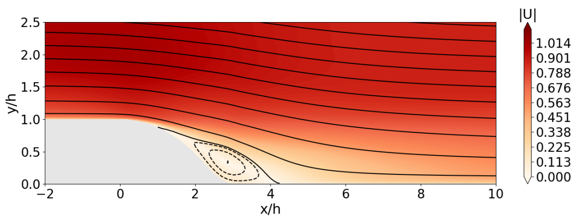
For all flow cases a description will be given, a mesh convergence study will be presented for the RANS meshes used in OpenFOAM, and the available DNS/LES data will be presented and discussed. Section 4.1 will give the overview for the periodic hills, Section 4.2 will present the converging-diverging channel, Section 4.3 will present the curved backward facing step, Section 4.4 will present the backward facing step, and finally Section 4.5 will present the square duct.



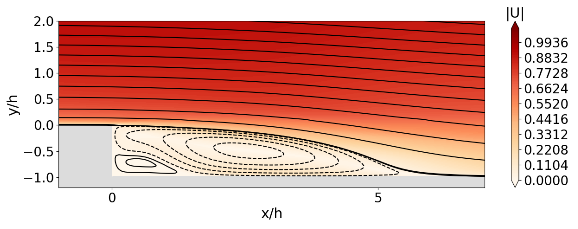
(a) Periodic Hills, Re = 10,595



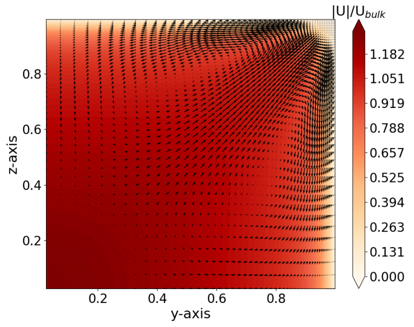
(b) Converging-Diverging Channel, Re = 12,600



(c) Curved Backward Facing Step, Re = 13,700



(d) Backward Facing Step, Re = 5,100



(e) Square Duct, Re = 3,500

Figure 4.1: Different flow cases used in the framework: velocity magnitude and streamlines. Clockwise rotating regions of separation are indicated by the dashed lines.

4.1 Periodic Hills

Description Data for the Periodic Hill flow case comes from Breuer et al. (2009), in which DNS data and highly resolved LES data was presented, along with a supporting experimental study using PIV measurements. For the lower Reynolds numbers ($Re = 700$ to $Re = 5,600$) DNS simulations were done, for the highest Reynolds number ($Re = 10,595$) a highly resolved LES simulation was done. The Reynolds number of the flow case is based on the bulk velocity at the inlet and the hill height.

This flow case was originally proposed by Mellen et al. (2000), as a computationally affordable test case involving separation, strong circulation and natural reattachment. Several properties of the periodic-hills flow case make it interesting for our problem at hand: it features a recirculation bubble, non-parallel shear layers, and mean flow curvature, which are all known to pose challenges for RANS based turbulence models (Wang et al., 2017b). Reattachment of the flow is strongly influenced by wall modelling and subgrid-scale models, making it an interesting test case for LES and RANS models (Breuer et al., 2009).

The flow case features no-slip upper and lower walls. As the flow case is periodic, a source term is added to the momentum equation in streamwise direction. The geometry and boundary conditions were implemented in OpenFOAM, where empty boundary conditions were used for the x-y planes.

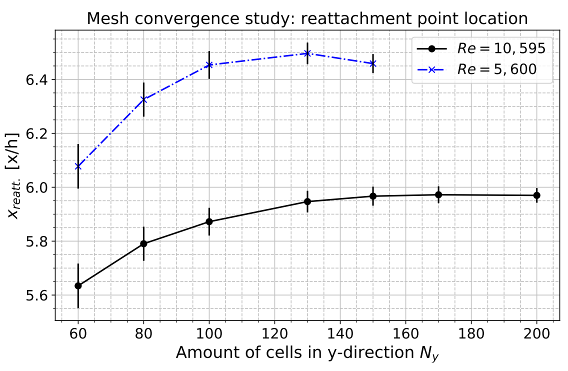
Mesh convergence In Figure 4.2 results for the mesh convergence study of the periodic hills flow case are presented. The amount of cells in x-direction (streamwise direction, N_x) and amount of cells in y-direction (N_y) were varied simultaneously, by multiplying the amount of cells in both directions with the same factor. For $N_y = 150$ the corresponding value for N_x is 140.

Figure 4.2a presents the location for the reattachment point $x_{reatt.}$ for various levels of mesh refinement. Results are shown for $Re = 10,595$ and $Re = 5,600$, both cases which were used for training in the machine learning framework. Error bars are plotted, based on the approximate mesh cell length. This gives the most conservative error possible from the fact that the wall shear stress is given at the cell centres of the mesh. As can be seen for $Re = 10,595$, the reattachment point converges to a steady value close to $x/h = 6$. From $N_y = 150$ further refinement barely has any influence on the reattachment point. This corresponds to an average y^+ value of 0.88 at the bottom wall as given by the `yPlus` tool in OpenFOAM. A similar pattern can be observed for $Re = 5,600$, for which further refinement than $N_y = 130$ does not seem to be necessary, also taking in account the error bars. At the bottom wall an average value of y^+ of 0.62 is calculated for this level of refinement.

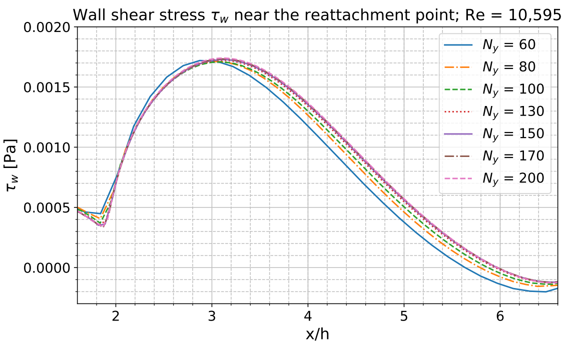
Figure 4.2b presents the wall shear stress close to the reattachment point for $Re = 10,595$. Convergence to a single line is observed. From $N_y = 130$ onwards the curves almost overlap. From these results a mesh refinement level of $N_y = 150$ was selected as a suitable refinement level with respect to convergence of the results, and taking in account the error for quantities of interest such as the reattachment point due to finite mesh cell sizes.

DNS/LES data-sets For the periodic hills flow, the six distinct components of the anisotropy tensor are presented in Figure 4.3. Since the flow is homogeneous in the z-direction, the b_{23} and b_{13} components are zero. There is a significant amount of anisotropy present in the flow which would not be captured by a linear eddy viscosity model. Most notably this can be seen when looking at the b_{33} component. In case of the linear eddy viscosity hypothesis, this component would be related to $\partial \bar{u}/\partial z$ and $\partial \bar{w}/\partial x$, which are both zero because of the aforementioned homogeneity of the flow in the z-direction.

The type of turbulence in the flow domain of the periodic hills is visualized using the RGB-map on the barycentric map, see Figure 4.4. Results for the highest Reynolds number ($Re = 10,595$) and the lowest Reynolds number ($Re = 700$) are visualized. As can be seen, there is



(a) $x_{reatt.}$



(b) τ_w near the reattachment point

Figure 4.2: Mesh convergence study for the Periodic Hills

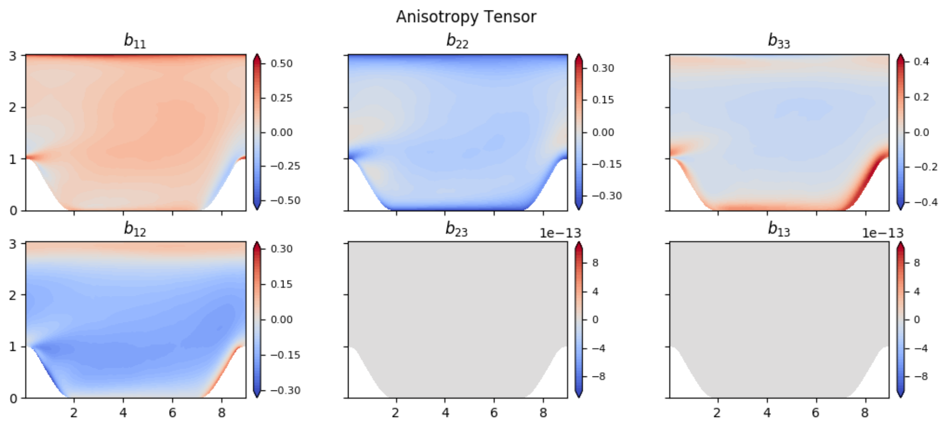
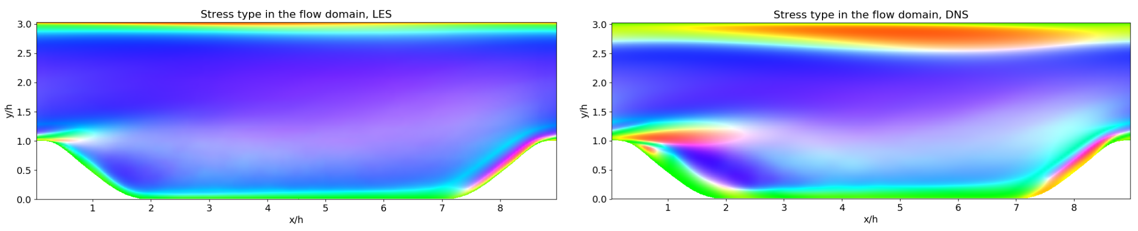


Figure 4.3: Components of b_{ij} for the periodic hills (LES: $Re = 10,595$)

mainly 2-component turbulence close to the upper and lower walls (the green colors, see the colormap in Figure 2.2a). Some 1-component turbulence is present along the upper wall (red colors), and at the converging part of the channel around $x/h = 8.0$.

The strongly 1-component behaviour of the turbulence around $x/h = 8.0$ is explained to be caused by the so called splatting effect, see Fröhlich et al. (2005). The large scale eddies generated in the shear layer are convected downstream onto the upward slope, causing a high level of fluctuations in the z-direction. This also means that eddy viscosity models, which do not take in account such transport effects, will not be able to capture this high anisotropy due to splatting. Some 1-component turbulence can also be observed just after the hill crest, convecting into the shear layer. This is due to a high streamwise stress, caused by a strong shear strain in the boundary layer. Due to the presence of the wall, the wall-normal stresses and stresses in z-direction are relatively low.

Since a periodic boundary condition is applied to the inlet and outlet, this means the 1-component stresses due to the splatting effect are transported towards the inlet. This, in



(a) $Re = 10,595$ **(b)** $Re = 700$

Figure 4.4: Stress type visualized using the RGB-colormap for the periodic hills (LES: $Re = 10,595$, and DNS: $Re = 700$)

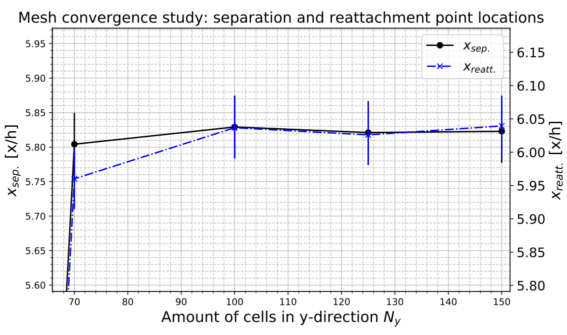
combination with the high fluctuations in streamwise direction due to the high streamwise stress, results in high 2-component stresses on top of the hill above the 1-component region. Both of these effects are most notably present for the $Re = 700$ case.

4.2 Converging-Diverging Channel

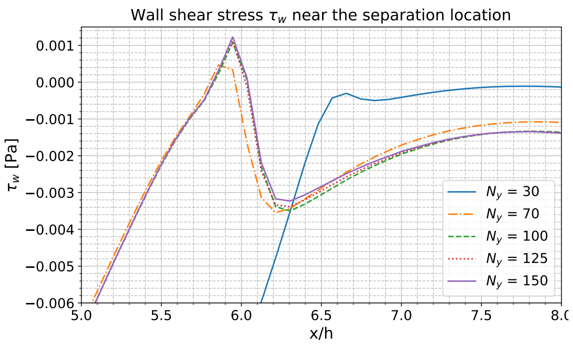
Description In Laval and Marquillie (2011) results from DNS simulations are presented for a converging-diverging channel. The Reynolds number of 12,600 is based on the channel half-height and the maximum velocity at the inlet. This flow case was simulated in order to investigate the behavior of turbulence in presence of of an adverse pressure gradient with and without curvature, and to gather data as a reference for RANS/LES models. The flow case features a small separation bubble just at the back of the bump. Flat channel flows at equivalent Reynolds numbers were used to generate the channel inlet boundary conditions. The channel features no-slip upper and lower walls.

The same procedure was used to generate the boundary conditions in OpenFOAM: at the same Reynolds number a channel flow was simulated using the given turbulence model ($k-\omega$). After convergence, the velocity profile and turbulent quantities from the channel flow were used as the inlet boundary conditions for the flow case. Geometry for the channel was obtained from Rumsey (2017).

Mesh convergence For the converging-diverging channel a mesh refinement study was done by changing the amount of cells in y-direction (N_y), i.e. in the wall-normal direction. Figure 4.5 presents the results for this study. In Figure 4.5a the separation location and reattachment point are plotted for the different levels of refinement. Just like for the periodic hills, error bars are included based on the mesh cell width, in order to give a conservative error due to the finite cell sizes of the mesh. For $N_y = 30$ no viable results were found. For further refinement results are quite similar. More details can be obtained by looking at Figure 4.5b. Here it can be observed that for the coarsest mesh results are completely different compared to the finer meshes. For $N_y = 70$ the wall shear stress is still clearly different compared to the finer meshes. For $N_y = 100$ and further on results start to overlap. A mesh refinement level of $N_y = 100$ was therefore selected as appropriate.



(a) $x_{sep.}$ (left y-axis), and $x_{reatt.}$ (right y-axis)



(b) τ_w near the reattachment point

Figure 4.5: Mesh convergence study for the converging-diverging channel

DNS data-set Figure 4.6 presents the six distinct components of the anisotropy tensor b_{ij} for the converging-diverging channel. Just like for the periodic hills flow, the b_{23} and b_{13} components are zero. Again the DNS data gives significant amounts of anisotropy for the b_{33} component, which will yield a zero magnitude in case of a linear eddy-viscosity model. Large magnitudes for the anisotropy tensor components can be observed on the hill.

The RGB-colormap as generated by using the barycentric map is presented in Figure 4.7 for the converging-diverging channel. As can be seen, 1-component turbulence is present around the separation region on top of the hill, and and the upper wall above that. Since the 1-component turbulence is present on the upper wall as well, Laval and Marquillie (2011) explain this behavior not by the strong stresses due to separation as was the case for the periodic hills, but due to the intensity of the adverse pressure gradient and curvature effects.

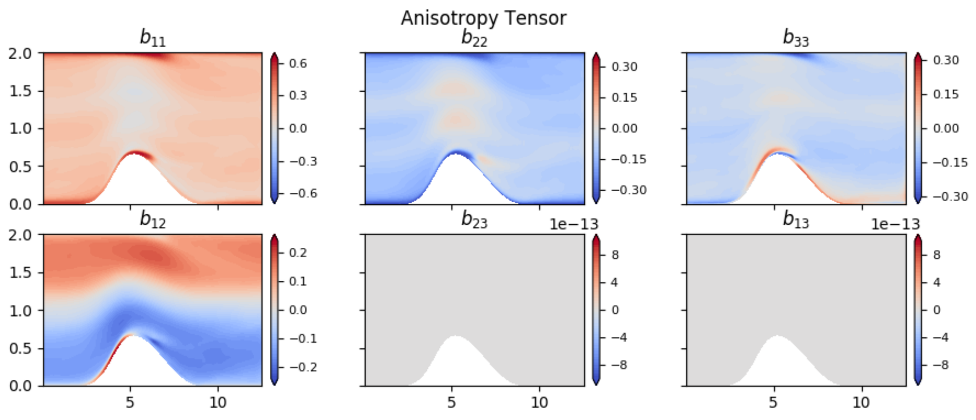


Figure 4.6: Components of b_{ij} for the converging-diverging channel (DNS)

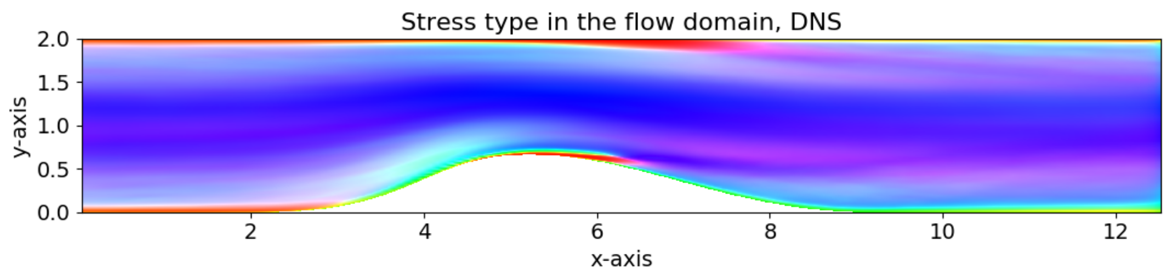


Figure 4.7: Stress type visualized using the RGB-colormap for the converging-diverging channel (DNS)

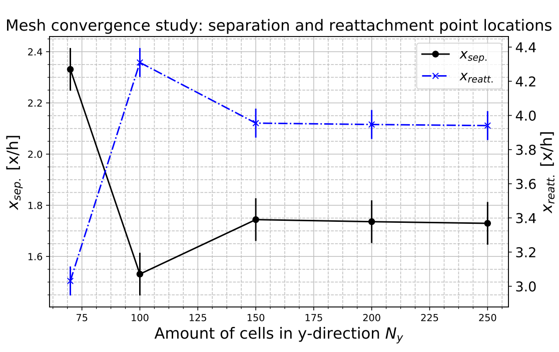
4.3 Curved Backward Facing Step

Description In Bentaleb et al. (2012) a highly resolved LES simulation is presented for a turbulent boundary layer separating from a rounded step. The Reynolds number of $13,700$ is based on the step height and the center-channel inlet velocity. The goal of this simulation was to obtain details about the separation process, and properties of the separation region and reattachment. Just like the periodic hills flow case, both the separation location and reattachment point can be taken as quantities of interest. Separation on such a curved surface poses a challenge for RANS turbulence modelling: even though there is a strong adverse pressure gradient, the shear stress grows as well, which increases mixing of the flow which in turn helps against separation (Bentaleb et al., 2012). Just as for the converging-diverging channel, the flow case uses a fully developed boundary layer obtained from a channel flow for the inlet boundary conditions. Furthermore both the upper and lower walls feature no-slip boundary conditions.

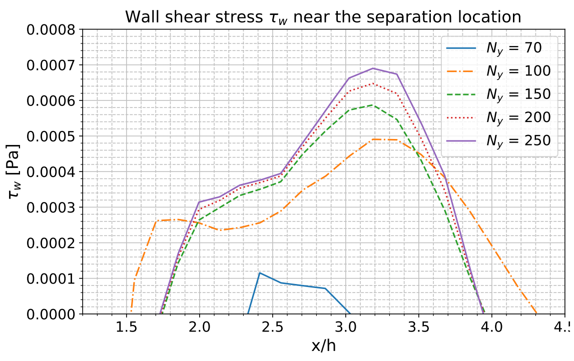
Geometry of the flow case was obtained from Rumsey (2017). Similarly to the converging-diverging channel, a channel flow at equivalant Reynolds number was simulated beforehand in order to generate the boundary conditions for the OpenFOAM set-up (velocity and turbulent quantities).

Mesh convergence Results for the mesh convergence study are presented in Figure 4.8. The amount of cells in y-direction (N_y), i.e. in wall-normal direction, was varied, while keeping the amount of cells in x-direction fixed at $N_x = 140$.

Changes in both the separation location and reattachment point can be observed for $N_y = 70$ and $N_y = 100$ when looking at Figure 4.8a. From $N_y = 150$ and onwards no major changes are observed. The same conclusion can be made when analysing the wall shear stress near the separation location as shown in Figure 4.8b. For $N_y = 150$ and finer meshes the profiles are relatively similar. From these results, a refinement level of $N_y = 150$ was selected as sufficient.



(a) $x_{sep.}$ (left y-axis), and $x_{reatt.}$ (right y-axis)



(b) τ_w near the reattachment point

Figure 4.8: Mesh convergence study for the curved backward facing step

LES data-set The LES data-set from Bentaleb et al. (2012) was analysed for its correctness before using it in the machine learning framework. Figure 4.9 presents the six distinct components of the anisotropy tensor, and in Figure 4.10 the stress type in the flow domain is plotted by using the RGB map on the barycentric map (see Section 2.2).

Some possible inconsistencies can be noticed near the horizontal centerline of the duct. A mixture of mainly 2-component stresses, along with some 1-component stresses is shown here. Some noise-like patterns of 1-component stresses can be observed as well. Both of these observations are not expected at this location. One possible explanation is the boundary condition used for the LES simulation. The momentum-thickness Reynolds number Re_θ at the inlet is 1,190. A channel flow was simulated, from which data was stored at the streamwise position for which Re_θ = 1,190. This data was used for the inlet boundary condition of the curved backward facing step. It might be possible that a laminar flow was used for the channel, which was tripped in order to generate a turbulent boundary layer. This could lead to inconsistencies at locations where the flow is technically still laminar further away from the wall. While data for the anisotropy tensor in the middle of the channel might therefore be unrealiable, data closer to the wall is still expected to be of good quality since extensive validation was performed as shown in Bentaleb et al. (2012). It was decided to use the data only for validating results closer to the wall, and not using it for training the machine learning algorithms.

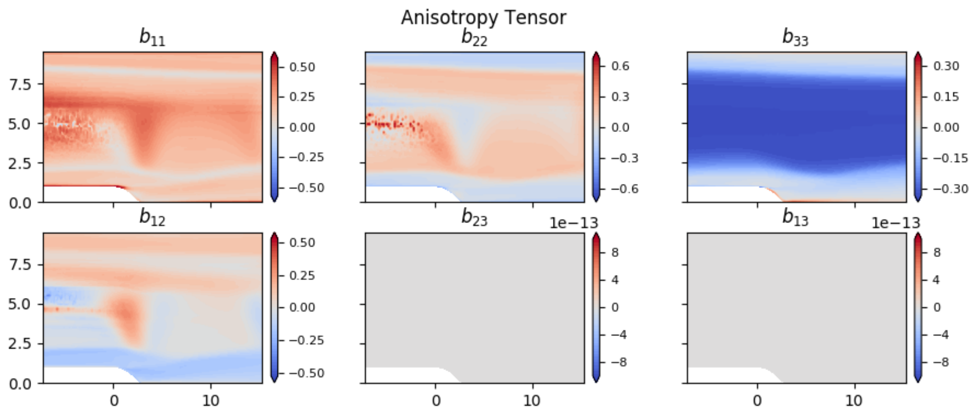


Figure 4.9: Components of b_{ij} for the curved backward facing step (LES)

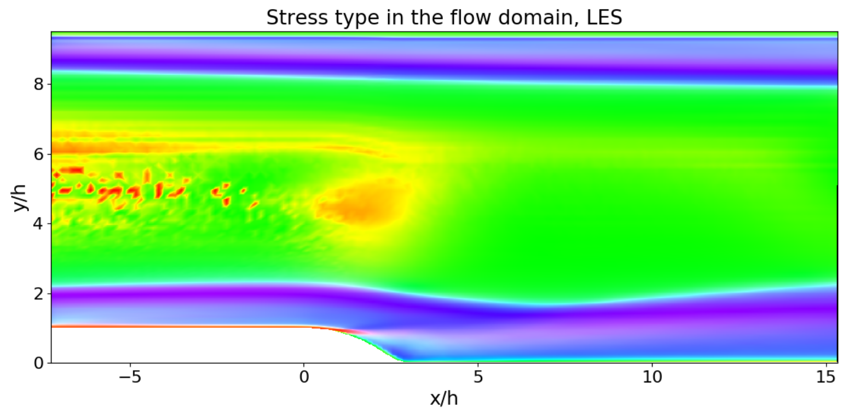


Figure 4.10: Stress type visualized using the RGB-colormap for the curved backward facing step (LES)

4.4 Backward Facing Step

Description The DNS simulation for the backward facing step can be found in Le and Moin (1992) and Le et al. (1997). In contrast with the curved backward facing step the separation location is fixed. The flow case features a circulatory region behind the step rotating in the clockwise direction, and a smaller region in the corner rotating counter-clockwise. The Reynolds number of the flow case is $Re = 5,100$, and is based on the step height and free stream velocity.

The upper boundary of the flow case features a no-stress wall, the lower wall uses a no-slip boundary condition, and the channel inlet boundary condition is given by a developed boundary layer simulation.

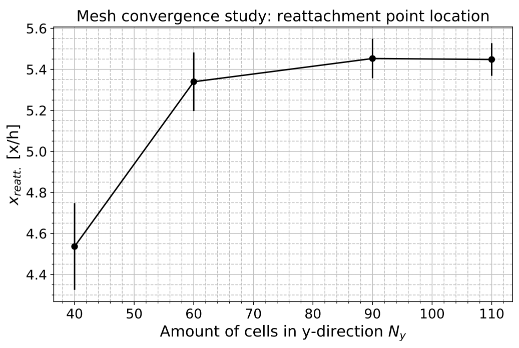
Similarly to the previous flow cases, a boundary layer at equivalent Reynolds number was simulated in OpenFOAM in order to generate the inlet boundary conditions for the velocity and turbulent quantities.

Mesh convergence For the backward facing step flow case, results from the mesh convergence study are presented in Figure 4.11. Both the amount of cells in x-direction (streamwise direction, N_x) and y-direction (N_y) were varied, by multiplying the amount of cells in both directions by a certain factor. For $N_y = 90$, the corresponding value for N_x was set to 150, after which two coarser levels and one finer level were created and run with OpenFOAM. The backward facing step flow case consists of two separate blocks: one before the step (block 1) and one after the step (block 2). Both blocks are connected to each other and are created using the `blockMesh` utility from OpenFOAM. The value N_y refers to the amount of cells in wall-normal direction for block 1. For $N_y = 90$, the amount of cells in wall-normal direction for block 2 corresponds to 140, from which the 90 uppermost cells are adjacent to the 90 cells from block 1. The amount of cells in streamwise direction (N_x) is the same for both block 1 and block 2.

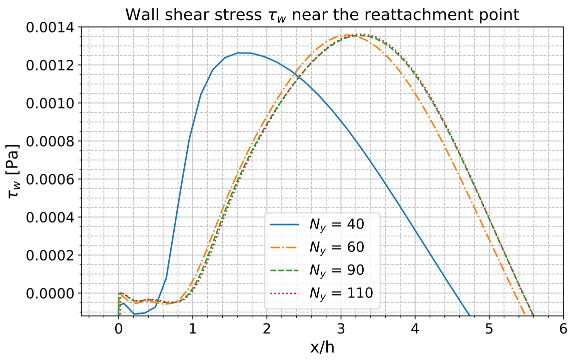
In Figure 4.11a the reattachment point location $x_{reatt.}$ is plotted as a function of N_y. As can be seen the reattachment point location converges to a value of roughly 5.4 x/h when refining the mesh. In Figure 4.11b the wall shear stress along the bottom wall is plotted. For $N_y =$ 40 results are significantly different from the rest. From $N_y = 60$ and onward the changes are very minute, with $N_y = 90$ and $N_y = 110$ virtually overlapping.

A level of $N_y = 90$, for which $N_x = 150$, was selected to be used for further simulations, since further refinement barely has any influence on the reattachment point location.

DNS data-set Unlike the previous flow cases, no data is available for the entire flow field. Therefore it is not possible to check the data using the RGB colouring of the barycentric map. Data is available at only five sections from the simulation presented in Le and Moin (1992), for which locations on the barycentric map can be checked. Results are presented in Figure 4.12 for the five different sections, where the data is coloured according to the location in the



(a) $x_{reatt.}$

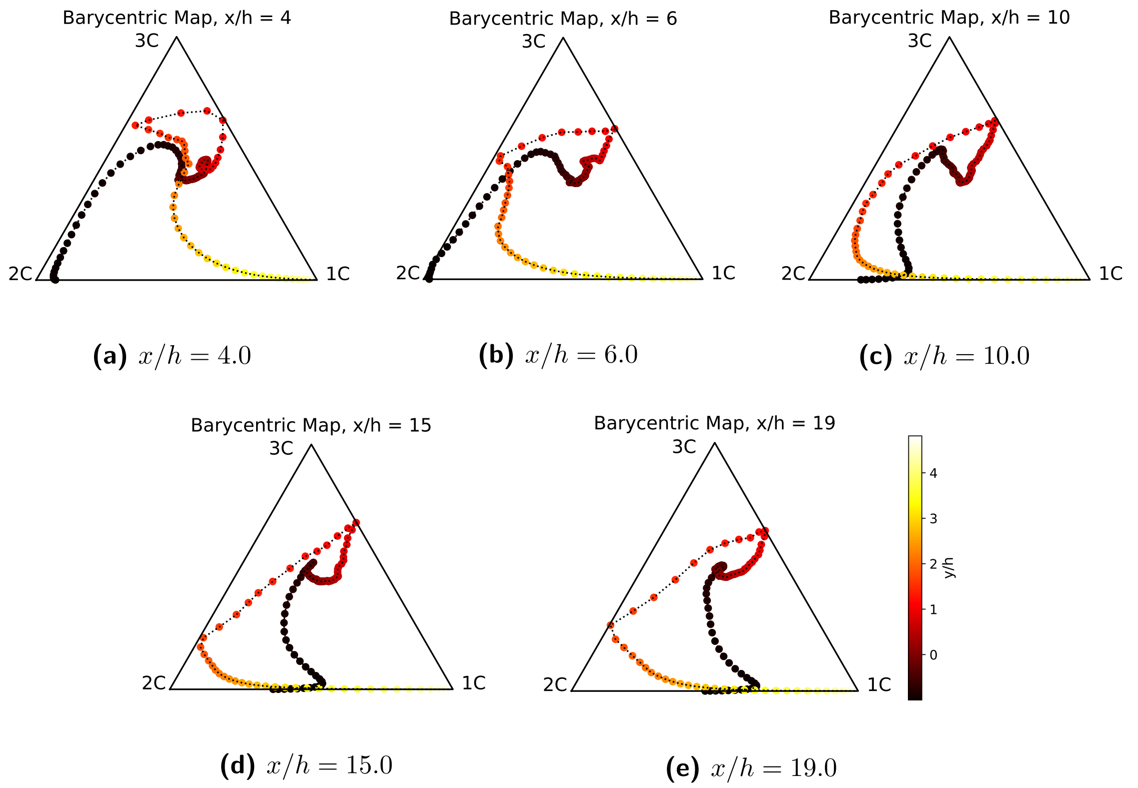


(b) τ_w near the reattachment point

Figure 4.11: Mesh convergence study for the backward facing step

flow field in terms of y/h.

It can be seen that the data shows 2-component turbulence close to the wall at $x/h = 4.0$ and $x/h = 6.0$. When moving further downstream, the turbulence close to the wall shifts slightly more to the 1-component corner. Moving upwards at $x/h = 4.0$, it can be seen that there is a mixture of all three types of turbulence near the location of the shear layer at approximately $y/h = 0.0$. Afterwards the turbulence closer towards the upper wall shows more 1-component behaviour. For the other four x/h locations, the data shows a bend towards the 2-component corner at approximately $y/h = 1.0-2.0$, after which it moves towards the 1-component corner when approaching the upper wall.



(a) $x/h = 4.0$ **(b)** $x/h = 6.0$ **(c)** $x/h = 10.0$

(d) $x/h = 15.0$ **(e)** $x/h = 19.0$

Figure 4.12: Barycentric map locations for the backward facing step (DNS). Colormap of the scattered data ranges from black (lower wall, y/h = -1.0) to yellow (upper wall, y/h = 5.0)

4.5 Square Duct

Description Results from DNS of the square duct flow case at various Reynolds numbers is found in Pinelli et al. (2010). This flow case is symmetric; Figure 4.1e only presents the upper right quadrant of the duct, where the flow in the duct moves out-of-plane. Secondary flow motion is visible in the cross-section of the duct, called Prandtl's secondary motion of the second kind. The goal of Pinelli et al. (2010) was to further investigate this phenomenon at different Reynolds numbers. Reynolds numbers vary from $1,100$ to $3,500$, based on the duct semi-height and the bulk velocity.

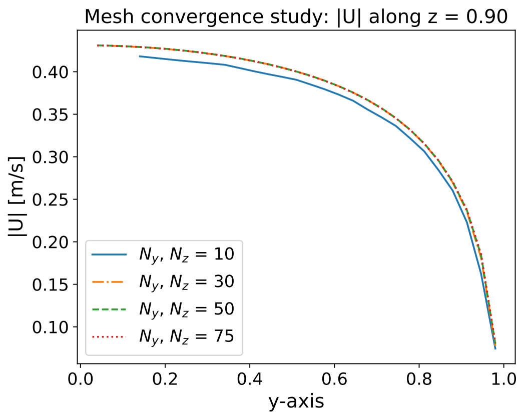
This flow case is interesting for the problem at hand, as secondary flows like this are not captured at all by linear eddy-viscosity turbulence models. Turbulence models using a linear relation between the Reynolds stress and the rates of strain fail to model turbulent fluctuations which cause these secondary flow motions, which is one reason more sophisticated Reynolds stress closure models are necessary (Pecnik and Iaccarino, 2007).

The square duct flow case is fully developed, which means a period boundary condition in streamwise direction was used, along with a momentum source term. All walls feature no-slip boundary conditions.

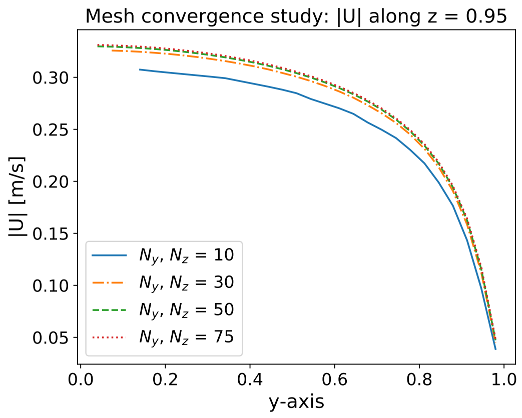
Mesh convergence For the square duct flow case, the influence of the mesh refinement on the velocity profile was investigated. Since the (linear) $k - \omega$ eddy viscosity model does not predict the corner vortices due to Prandtl's secondary motion of the second kind, it is not possible to investigate parameters such as the center of the corner vortices. Therefore only the mean velocity profile out-of-plane (in x-direction) was investigated for different levels of mesh refinement.

Figure 4.13 presents the results for the mesh convergence study of the square duct flow case, conducted at the highest available Reynolds number of 3,500. Two velocity profiles are presented close to the upper wall, one at z = 0.90 (Figure 4.13a), and one at z = 0.95 (Figure 4.13b). The mesh was refined by varying the amount of cells in both the y-direction (N_y) and z-direction (N_z).

In both figures a significant difference can be observed for the coarsest mesh ($N_y, N_z = 10$). For $N_y, N_z = 30$ and the finer meshes not many differences can be observed for z = 0.90. Moving closer to the wall at z = 0.95 some minor differences can be observed for $N_y, N_z = 30$ and the two finest meshes. The mesh at $N_y, N_z = 50$ was selected for further use, as improvements are negligible when going to the finest mesh, and as this refinement level gives a sufficient amount of data to work with in the machine learning framework ($N_y \times N_z = 2500$ data points).



(a) z = 0.90



(b) z = 0.95

Figure 4.13: Mesh convergence study for the square duct: velocity profiles at fixed z-location

DNS data-sets Figure 4.14 presents the 6 different components of the anisotropy tensor for the square duct flow. As can be seen, in this case all six components are non-zero, in contrast to the previous flow cases. A linear eddy-viscosity model would only yield results for the b_{12} and b_{13} components, as these are related to the $\partial \bar{u}/\partial y$ and the $\partial \bar{u}/\partial z$ terms of the velocity gradient tensor. The rest of the terms of the velocity gradient tensor would be zero, since the RANS simulation does not yield any velocities in the y-direction and z-direction, and since the flow is fully developed in the x-direction there is no $\partial \bar{u}/\partial x$ component either.

The stress type in the flow domain of the square duct is presented in Figure 4.15. As can be

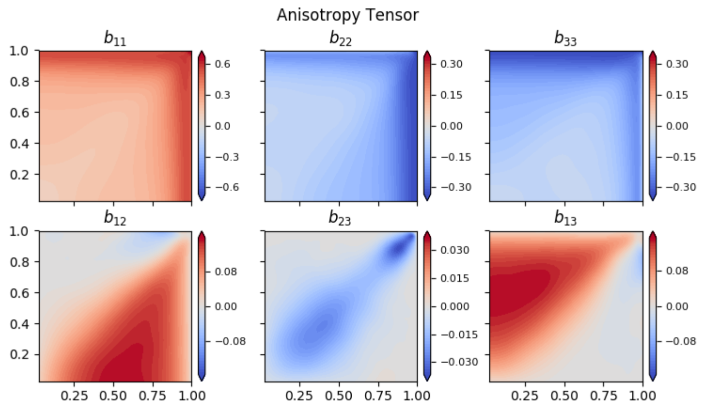


Figure 4.14: Components of b_{ij} for the square duct (DNS)

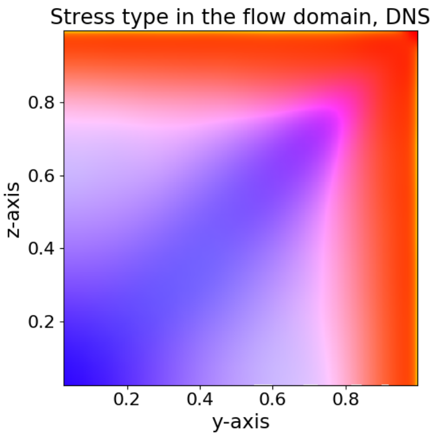


Figure 4.15: Stress type visualized using the RGB-colormap for the square duct (DNS)

seen, there is mainly a mix of 1-component turbulence and 3-component turbulence present in the domain. Close to the walls, the fluctuations in the x-direction dominate, resulting in the 1-component turbulence. More towards the centerline of the duct, the fluctuations in y-direction and z-direction are less damped by the presence of the walls, resulting in turbulence towards the 3-component corner of the barycentric map.

Chapter 5

Results from the preliminary investigations

In this chapter the preliminary results will be presented from which it was to be concluded which method of representing the Reynolds stress is the most suitable (see Section 3.1.2 in the methodology), and which method for propagating the Reynolds stress is the most suitable (see Section 3.1.3 in the methodology). Section 5.1 presents the results for the Reynolds stress representation methods, Section 5.2 presents the results for the propagation methods.

5.1 Representing the Reynolds stress

In order to illustrate the effect of different Reynolds stress representation methods used in literature, its effect was studied using the square duct flow case. Using Equation 2.32, it is possible to decompose the Reynolds stress using:

$$\mathbf{R} = 2k\left(\frac{1}{3}\mathbf{I} + \mathbf{V}\boldsymbol{\Phi}\mathbf{V}^T\right), \tag{5.1}$$

where $\mathbf{V}$ is the matrix containing the eigenvectors, and Φ is the diagonal matrix containing the eigenvalues. As discussed in Section 3.1.2, three different approaches used in literature could be identified. For the first approach only the eigenvalues are predicted (e.g. using the barycentric map) and substituted back in $\boldsymbol{\Phi}$. For the second approach, the anisotropy tensor is predicted (which would be equal to predicting both $\mathbf{V}$ and $\boldsymbol{\Phi}$), after which it can be either substituted back, or used in combination with the k-equation from a turbulence model. The third approach predicts both the anisotropy tensor and k to get a full description of the Reynolds stress. For reasons stated in Section 3.1.2, the third approach where the turbulent kinetic energy is predicted as well will not be investigated further.

The Reynolds stress anisotropy tensor decomposition will be denoted by:

$$\mathbf{b} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T, \tag{5.2}$$

where $\boldsymbol{\Lambda}$ is the diagonal matrix containing the Reynolds stress anisotropy tensor eigenvalues.

Figure 5.1 presents the anisotropy tensor for the square duct flow at a Reynolds number of 3,500. Each row represents a different component of the tensor, each column a different way of representing the tensor. All six unique components of the tensor are presented. The first column presents the anisotropy tensor as given by the DNS data from Pinelli et al. (2010). In the second column, the eigenvectors $\mathbf{V}$ from the RANS simulation are used, in combination with the eigenvalues $\mathbf{\Lambda}$ from DNS. This corresponds to the best case scenario for the first approach, where only the eigenvalues are predicted. Although in this case the barycentric map of column 1 and column 2 would be exactly the same, it can clearly be seen that the anisotropy tensor is not similar at all, due to the fact that the eigenvectors are still different. Slight similarities can be seen for b_{11}, which has a higher magnitude closer to the wall as the DNS data indicates, but for which the overall magnitude is still too small. Some similarities for b_{12} and b_{13} can be observed as well.

In the third column, the exact opposite is done: the eigenvectors are taken from DNS, after which the eigenvalues from RANS are substituted in $\mathbf{\Lambda}$. This is to show that in this case actually more similarities are found with respect to the DNS data, looking for example at b_{12}, b_{13}, and b_{23}. Also, for b_{22} and b_{33} higher magnitudes are predicted towards the right and upper wall of the duct respectively which is in correspondence to the DNS data, although the overall magnitude is still to small, and their magnitudes are zero at the upper and right wall respectively.

In the fourth column, a random forest from the `scikit-learn` library was trained on the anisotropy eigenvalues using the barycentric map. Training was done using data for the square duct at Re = 3,200, after which predictions for the eigenvalues at Re = 3,500 were substituted in Λ. Features based on $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$ were used, in combination with a dummy feature based on the x- and y-coordinates, $x - y$, see Figure 5.2. This extra feature was added in order to make it more easy for the machine learning algorithm to discern between the lower right and the upper left corner (given that most of the other features are symmetric around the diagonal). As can be seen the machine learning algorithm is able to predict the eigenvalues accurately. Substituting the predicted eigenvalues back in the expression for b_{ij} gives almost identical results to the DNS data, except from some minor noise around the middle of the domain.

In the fifth column, more difficulty is added, by letting the machine learning algorithm also predict the eigenvectors in $\mathbf{V}$. This was done using the methodology presented Section 3.1.2, see Equation 3.2 to Equation 3.5. As discussed in Section 3.1.2, one problem with this representation of the Reynolds stress is the fact that it depends on the frame of reference. Going between different square duct cases is not a problem however. As can be seen predictions are still reasonably good. Some extra noise is introduced compared to the previous case.

Finally in the sixth column the anisotropy tensor is predicted by using the TBNN algorithm, meaning it is represented using the tensor basis derived in Pope (1975), see Equation (2.40). Although the predictions are far from perfect, it is visible that the general trends correspond to the DNS data. For example, on average b_{11} has a larger magnitude closer to the wall, b_{22} and b_{33} have larger magnitudes closer to the right and upper wall respectively, and the other three components are predicted quite well. Considering this method is independent on the frame of reference, unlike the prediction in column 5, means that this method of representing

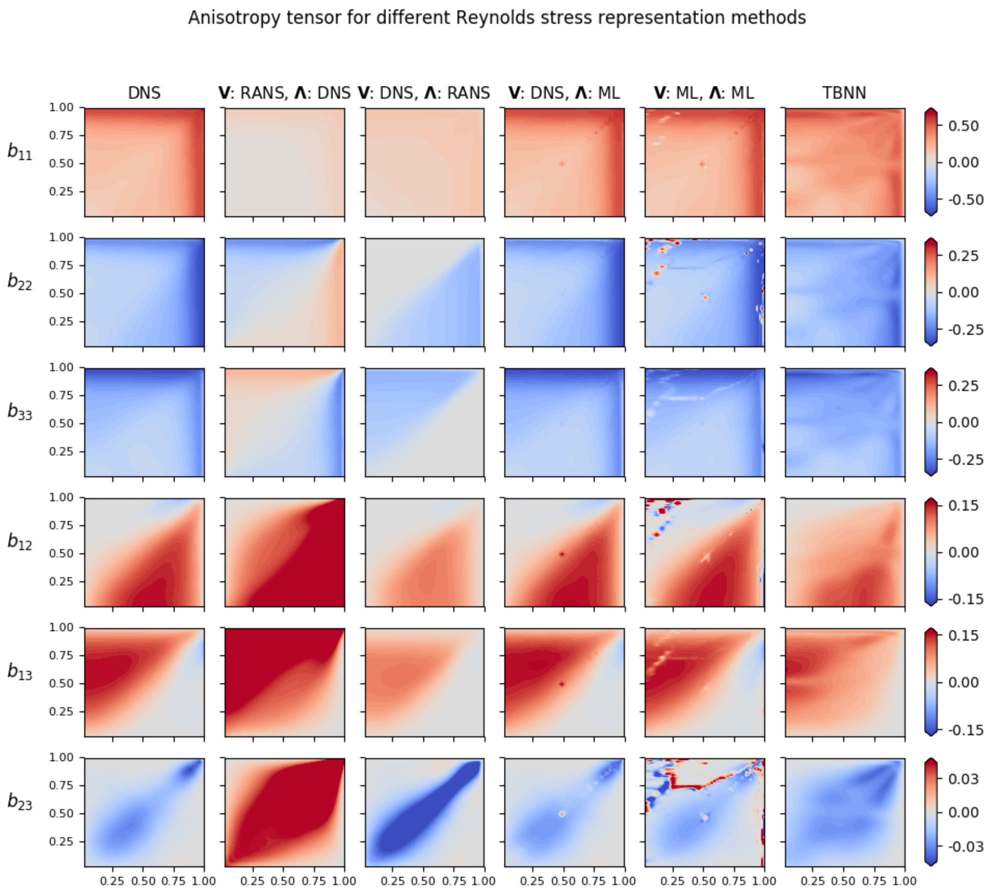


Figure 5.1: Comparison between different methods of representing the Reynolds stress anisotropy tensor. Square duct, Re = 3,500

the anisotropy tensor is a suitable candidate for further exploration.

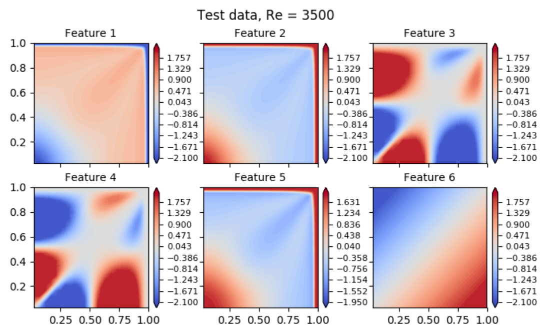


Figure 5.2: Features used for the machine learning algorithms in Figure 5.1

5.2 Propagating the Reynolds stress

In this section preliminary results will be presented with regards to propagating the Reynolds stress. First the appropriate method for propagating the Reynolds stress had to be selected, for which results are presented in Section 5.2.1. Afterwards some propagated flow fields using different methods of representing the Reynolds stress will be presented in Section 5.2.2, in order to investigate whether it would be required to predict the entire tensor instead of just the eigenvalues as discussed in the previous section.

5.2.1 Different approaches for propagation of the Reynolds stress

As explained earlier in Section 3.1.3, the different methods investigated for propagating the Reynolds stress can roughly be divided into three different groups. In the first approach, the Reynolds stress is propagated by inserting it into the momentum equation. This is for example the approach used in Wang et al. (2017a). In the second approach, as used by Ling et al. (2016b), the anisotropy tensor is introduced into the momentum equation, and a modified version of the k-equation is solved. The third approach which was investigated blends the machine learning predictions for the anisotropy tensor with predictions from a standard RANS turbulence model, see Section 3.9 for more explanation.

In order to validate the different approaches, DNS data was propagated for the square duct flow case. This serves as the simplest case possible which the propagation methods should be able to pass, since the DNS data does not show any noise and the test case geometry is the simplest available.

Figure 5.3 presents the flow field as gived by DNS, along with results from the three different approaches for propagating the Reynolds stress. In Figure 5.3a the flow field as given by the DNS data is presented, where the colormap indicates the absolute mean flow velocity, and the quivers indicate the secondary flow motion in the cross-section plane of the duct. As can be seen the secondary flow moves from the bottom left corner to the upper right corner, after which the secondary flow splits and follows the right and upper walls.

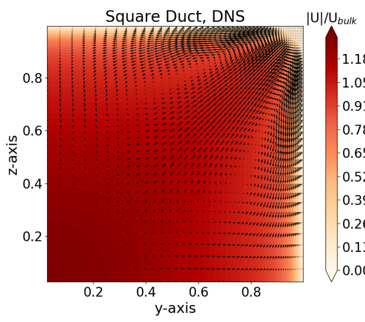
In Figure 5.3b the Reynolds stress field is propagated using the first approach, where the Reynolds stress is injected into the momentum equation. Predictions show a good match to DNS when looking both at mean flow velocity and the secondary flow motion. Results converged to a steady state when looking at the residuals for the velocity. In order to improve convergence of the flow field, results were first propagated using a more dissipative first-order accurate numerical scheme, after which this flow field was used to initialize the second-order accurate numerical scheme which was normally used.

Figure 5.3c presents the propagated velocity field using the second approach. This approach had trouble converging to a steady state, even when changing relaxation factors and using more dissipative first-order accurate numerical schemes. This can be observed when looking at the secondary flow motion, which looks more chaotic and non-symmetric. It can be observed however that close to the corner the secondary flow motion is predicted quite well.

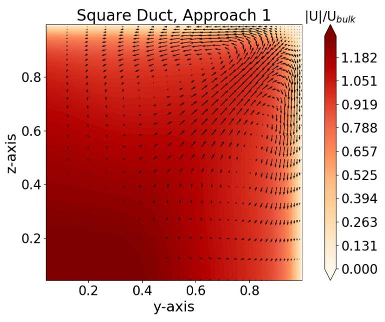
Figure 5.3d presents results for the third approach for propagating the anisotropy tensor. Results show a good match to the DNS data, and the solver converged quite well thanks to the blending parameter introduced in the continuation solver, which was increased up to a value of $\gamma = 0.8$.

In order to test the stability of the different approaches, the next step was to propagate a field predicted by a machine learning algorithm. Since these predictions will contain some noise, this might affect the numerical stability significantly. The anisotropy tensor was predicted for the square duct flow case using the TBNN algorithm, which was trained on the square duct at Re = 2,400; Re = 2,600; Re = 2,900; and Re = 3,200 in order to make the predictions as accurate as possible. Training and testing was done using the features in feature set 1 and feature set 2, see Table 3.1. In order to reduce the noise from the predictions, a Gaussian filter was used. As an example, three of the components from b_{ij} are plotted in Figure 5.4, which include the RANS simulation, the DNS data, the TBNN predictions, and the TBNN predictions after applying the Gaussian filter (left to right, top to bottom). As can be seen the filtered predictions for b_{ij} are quite accurate and smooth. Since it should be possible to use the propagation method for more complex geometries and less accurate/more noisy predictions, it should have no problems propagating this sample. For approach 1, which converged when giving R_{ij} from DNS, the turbulent kinetic energy from the DNS data was used in combination with b_{ij} from the TBNN algorithm ($b_{ij,TBNN}$). It was not possible to get a converging solution this time however, whatever changes were made to the underlying turbulence model for calculating ν_t (which should increase stability of the solver, see Section 3.1.3 for more explanation), the numerical schemes, and the relaxation factors. The third approach did converge however, for which the result is presented in Figure 5.3e. As can be seen there is still great similarity to the DNS data. Most of the discrepancy can be seen closer towards the centerline of the duct, where some minor secondary vortices are present in the propagated flow field.

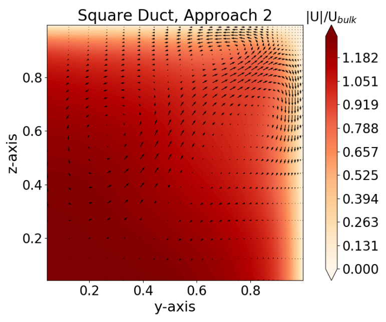
From the preliminary findings presented in this section, it was decided to use the third approach for propagating the anisotropy tensor, using the continuation solver. It outperforms the other two approaches with respect to stability by far, and yields a close match for the propagated velocity field compared to the DNS data.



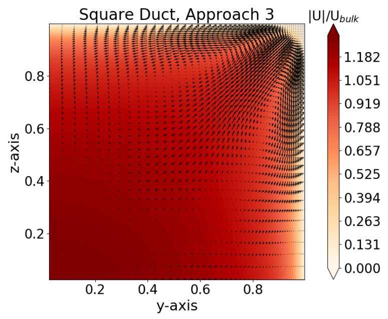
(a) DNS (50×50 mesh)



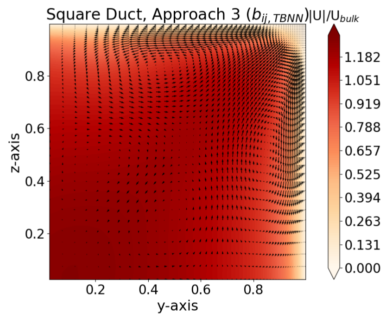
(b) Approach 1, $R_{ij,DNS}$ (30×30 mesh)



(c) Approach 2, $b_{ij,DNS}$ (30×30 mesh)

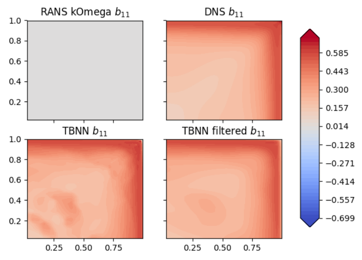


(d) Approach 3, $b_{ij,DNS}$ (50×50 mesh)

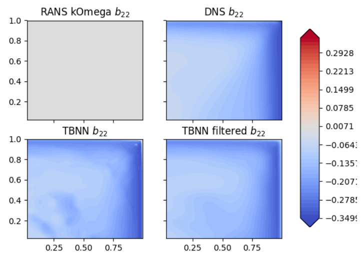


(e) Approach 3, $b_{ij,TBNN}$ (50×50 mesh)

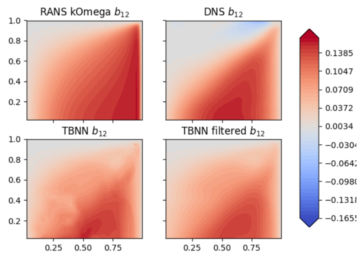
Figure 5.3: Comparison DNS and the propagated flow fields using the different propagation methods



(a) b_{11}



(b) b_{22}



(c) b_{12}

Figure 5.4: Example TBNN predictions for b_{ij} in the square duct, serving as a test case for the propagation methods

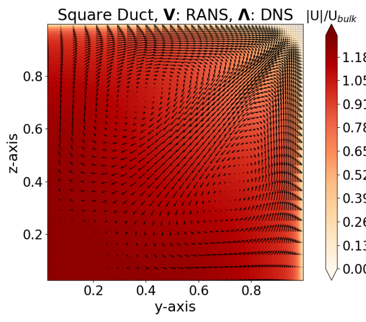
5.2.2 Propagating different representations of the Reynolds stress

In order to investigate whether it is worthwhile to predict the entire anisotropy tensor instead of just its eigenvalues as done in for example Tracey et al. (2013) and Ling et al. (2017b), a study was conducted where DNS data for only the anisotropy tensor eigenvalues or eigenvectors were substituted in Equation 5.2, after which b_{ij} was propagated using the continuation solver. This way an idea can be obtained whether it is really important to predict both the eigenvalues and eigenvectors, or whether only predicting the eigenvalues would be sufficient, even though in this case b_{ij} might look significantly different compared to the DNS data, see Figure 5.1.

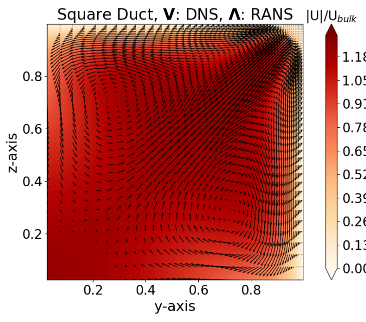
Figure 5.5 presents the results of this study. In Figure 5.5a the eigenvectors from RANS are used, along with the eigenvalues from DNS to reconstruct b_{ij}. This corresponds to the best case scenario of using machine learning algorithms for only predicting the eigenvalues. As can be seen the propagated flow field is completely different from the DNS data which was presented in Figure 5.3a. While at first sight the secondary flow motion seems roughly similar, the rotation is the wrong way around, i.e. flow along the diagonal goes from the upper right corner to the lower left corner.

In Figure 5.5b the anisotropy tensor is reconstructed the other way around: eigenvectors are taken from DNS, and the eigenvalues from the RANS simulation. As can be seen in this case the secondary flow motion is oriented correctly, with the flow along the diagonal moving from the lower left corner to the upper right corner. The magnitude of the secondary flow is overpredicted however, and the magnitude of the mean flow shows discrepancy (it is somewhat heart-shaped, instead of a monotonous decrease from the centreline of the duct towards the walls).

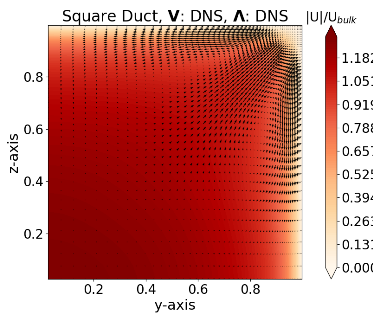
Finally, in order to verify whether the eigenvalue decomposition of the anisotropy tensor was implemented correctly, eigenvectors from DNS were used in combination with eigenvalues from DNS, after which b_{ij} was reconstructed. Results are presented in Figure 5.5c, and as can be seen results are similar to the DNS data, just like the propagated flow field in Figure 5.3d.



(a) $\mathbf{V}$: RANS, $\mathbf{\Lambda}$: DNS



(b) $\mathbf{V}$: DNS, $\mathbf{\Lambda}$: RANS



(c) $\mathbf{V}$: DNS, $\mathbf{\Lambda}$: DNS

Figure 5.5: Comparison of the propagated flow fields using different combinations of eigenvectors and eigenvalues from RANS and DNS data

From the preliminary findings presented in this section it was concluded that it is indeed worthwhile to predict the entire anisotropy tensor, instead of just predicting the eigenvalues. It raises the question whether just predicting the eigenvalues is useful at all, when keeping the eigenvectors from a baseline RANS simulation. This holds for machine learning related results presented in for example Tracey et al. (2013) and Ling et al. (2017b). But it also raises the question whether eigenvalue perturbation techniques such as presented in Emory et al. (2013) are useful for uncertainty quantification of parameters of interest in a flow. When the eigenvectors of the RANS simulation are structurally different from those given by the ground truth (DNS), it seems uncertain that parameters of interest will lie within uncertainty bounds given by this uncertainty quantification method.

Chapter 6

Predicting and Propagating b_{ij} using the TBRF framework

This chapter will discuss the main results from the machine learning framework. Before using the TBRF algorithm, its hyperparameters need to be tuned, for which the process is presented in Section 6.1. Results from the machine learning framework can be divided into two different parts. First of all there are the predictions for the anisotropy tensor itself, as presented in Section 6.2, which includes looking at the anisotropy tensor components, inspecting predictions on the barycentric map, and using the RGB colormap as presented in Section 2.2 to visualize the flow field. Secondly, predictions for the anisotropy tensor will be propagated in order to see whether this leads to an improved flow field in Section 6.3.

6.1 Influence and tuning of the TBRF hyperparameters

Before using the Tensor Basis Random Forest, the hyperparameters need to be tuned for optimal performance. These parameters include the amount of regularization used for the linear least squares regression in the decision tree (λ), the amount of trees in the random forest (N_{trees}), the minimum amount of samples in the TBDT leaf nodes ($N_{s,l}$), the amount of features in the randomly selected subset from which the optimal splitting feature is chosen during the tree building process ($N_{f,s}$), and the median filter threshold ($\theta_{med.}$). Finally, the TBRF also includes the possibility to use an optimization algorithm to select the optimal splitting feature and value instead of using a brute-force search. The effect of the threshold at which optimization is used instead of a brute-force search ($\theta_{optim.}$) could be treated as a hyperparameter. The effect of this threshold could not be studied as thoroughly as the other parameters however, since not using optimization proved to be very computationally expensive. A simple study done using the square duct flow case only is presented in Appendix C.2.

All parameters have been tuned using a training data-set which includes PH10595, PH5600, and CD12600. Ideally, the hyperparameters would be tuned for each different training data-set which is used, using the OOB samples or a unseen flow case as the validation data-set. Due to time constraints the analysis of the hyperparameters was done only once, using the validation data-sets SD3200 and PH2800. These validation data-sets are used for tuning the hyperparameters only. For testing the performance of the TBRF algorithm, separate test data-sets are used.

The different hyperparameters might influence each other, meaning that ideally either a grid search would be done in order to find the optimal combination of parameters, or performing optimization of the parameters, e.g. using Bayesian optimization (Snoek et al., 2012). In the case of Bayesian optimization, performance of the algorithm is modeled as a sample from a Gaussian process, which allows for efficient sampling in the parameter space. In Ling et al. (2016b) Bayesian optimization was utilized in order to optimize the architecture of the TBNN (amount of hidden layers, nodes per layer, and learning rate). Due to time constraints it was assumed that most of the TBRF hyperparameters do not influence each other in a very significant way. Most of the parameters were changed separately in order to observe its influence on the performance of the TBRF, while keeping the other parameters at a constant value.

At the beginning of the hyperparamter tuning process, it was decided to set the minimum amount of samples of the decision trees to 1, leading to fully grown trees which are as diverse as possible. This is also the approach which was taken in Ling et al. (2017b), where in fact the only hyperparameter tuned for the random forest was the amount of decision trees. From preliminary investigations, the initial value for the regularization parameter (λ) was set to $1{\times}10^{-15}$, the median filter threshold ($\theta_{med.}$) to a value of 3, and the amount of features used per split ($N_{f,s}$) to the total amount of features (in this case 17). For $N_{f,s}$ this is the safest choice, since it is not known beforehand whether most of the features are usable. Also starting with the standard setting for the normal random forest (where $N_{f,s}$ is equal to the total amount of features divided by three) might not be appropriate, since the TBRF operates in a significantly different way than the standard random forest. In the parameter tuning process, first the influence of the amount of TBDT's in the TBRF was investigated, since this is the main influence on the computational cost of training. After selecting an appropriate amount of trees, the effect of regularization and filtering was investigated and set to their optimal values. Afterwards the amount of features used for finding the optimal split ($N_{f,s}$) was tuned and set, and finally the minimum amount of samples per leaf node ($N_{s,l}$) was investigated and set to its optimal value.

Total amount of features During the research process, it was found that the random forest performs better when increasing the amount of features, at least for the maximum amount of features tested using the algorithm, which went up to 17 (see Section 3.3). Using only five features based on S_{ij} and Ω_{ij} as done in Ling et al. (2016b) proved to be insufficient. As an example, Appendix D.1 will show results from a preliminary investigation, with a comparison when using only these 5 features, and using 11 features which include those based on $\nabla\mathbf{k}$. Section 6.2.3 will go into more detail and present results for the square duct using

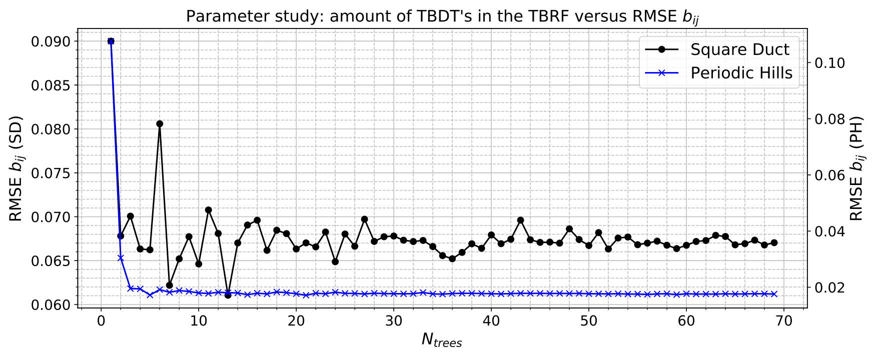


Figure 6.1: Parameter study: influence of the amount of TBDT's in the TBRF

features based on S_{ij} and Ω_{ij} only (feature set 1, see Table 3.1), and using all available features (feature set 1, 2, and 3).

Amount of TBDT's The effect of the amount of trees on the root mean square error of b_{ij} was observed for the validation data-sets containing SD3200 and PH2800. Results are presented in Figure 6.1. As can be seen, on average the RMSE of b_{ij} decreases when increasing the amount of TBDT's in the TBRF. Some noise is visible, due to the fact that some of the trees will perform better or worse on the given test case.

For the square duct flow case, shown by the black line with dots, the first TBDT does a relatively good job at making predictions for b_{ij}, especially considering the outlier filter will not work with only one tree. Introducing more TBDT's reduces the variance of the predictions, eventually leading to a relatively steady behaviour around $N_{trees} = 20$. A more extreme decrease in the RMSE of b_{ij} can be observed for the periodic hills, shown by the blue line with crosses, when using more than one TBDT. After $N_{trees} = 10$ the further reduction in RMSE is relatively small.

For further analysis of the TBRF hyperparameters, a level of $N_{trees} = 30$ was chosen in order to save computational cost, as benefits are marginal when introducing more trees from this point onwards, while still having some margin in case variance of the different trees increases somewhat due to hyperparameter settings. When training the TBRF with the final settings for the hyperparameters, saving training costs is less important, meaning that more than 30 TBDT's will be trained. For the final predictions it was decided to use 100 TBDT's.

Regularization and median filtering In order to obtain smoother predictions, the TBRF has the option to include regularization on the tensor basis coefficients. Different values were analysed for the PH2800 and SD3200 flow cases. Instead of regularization results can also be smoothened using the median outlier filter presented in Section 3.8.1. Both options were analysed to find the optimal combination. Results for the RMSE of b_{ij} are presented in Figure 6.2 for various median filter thresholds, as well as the effect of regularization without using the median filter.

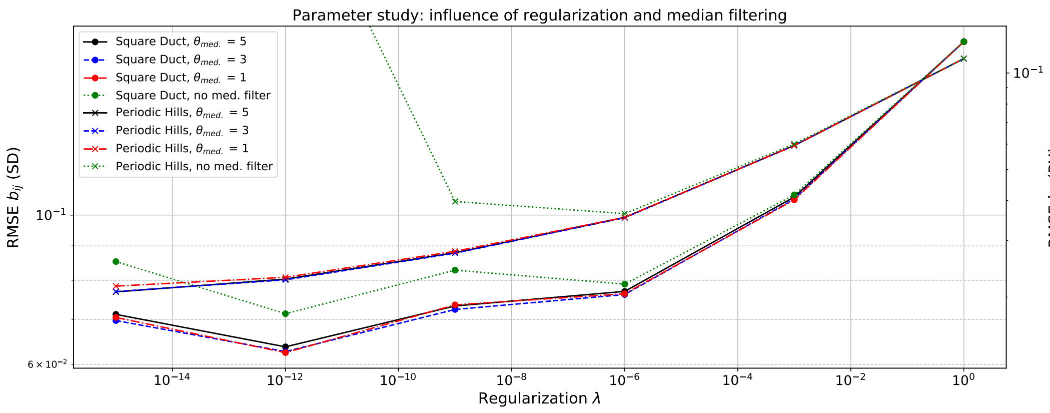


Figure 6.2: Parameter study: influence of the regularization parameter λ, and the median filter threshold $\theta_{med.}$

The result for the square duct is plotted using the different lines marked with dots. Each different line type and color represents a different value for the median filter threshold $\theta_{med.}$. As can be seen using the median filter is always better than not using it (indicated by the green dashed line). It is optimal to set the regularization parameter to a very low value ($\sim 1\times10^{-15}$ - 1×10^{-12}), along with using a median filter threshold of about 1 or 3.

In the same figure results are plotted for the periodic hills, using the lines marked with crosses. Again it can be observed that using the median outlier filter is always advantageous. Around $\lambda = 1\times10^{-6}$ results without filter become similar to the rest of the results. After inspecting a close up of the figure, it was observed that not using a median filter is never better than using one for all values of λ. Overall, a median filter threshold of about 3 or 5 seems to perform the best.

Following the results from the parameter study regarding the regularization factor, it was decided to set it to a very low value of $\lambda = 1\times10^{-12}$, in combination with using a median outlier filter with $\theta_{med.} = 3$. In Appendix D.3 and D.2 a more elaborate discussion is given on the influence of λ on predictions of b_{ij} in the flow field, and the use of the median filter.

Number of features used per split The effect of varying the amount of features which are used to select the optimal splitting feature ($N_{f,s}$) is presented in Figure 6.3. Both results for the square duct test case (left y-axis, black line with dots) and periodic hills test case (right y-axis, blue line with crosses) are plotted. For both test cases it is visible an optimal value is present. For the square duct flow case the least amount of RMSE is given when using 14 features per split. When using a low value for $N_{f,s}$ such as 5, the error is relatively high. It is likely that this is due to the fact that some of the features are not always suitable for creating a split. When only using a small subset of the features for selecting the optimal splitting feature, this reduces the chance a suitable splitting feature is used. The variance of the decision trees will get larger as less features features are available, increasing the left hand side of expression (3.26) for the random forest variance.

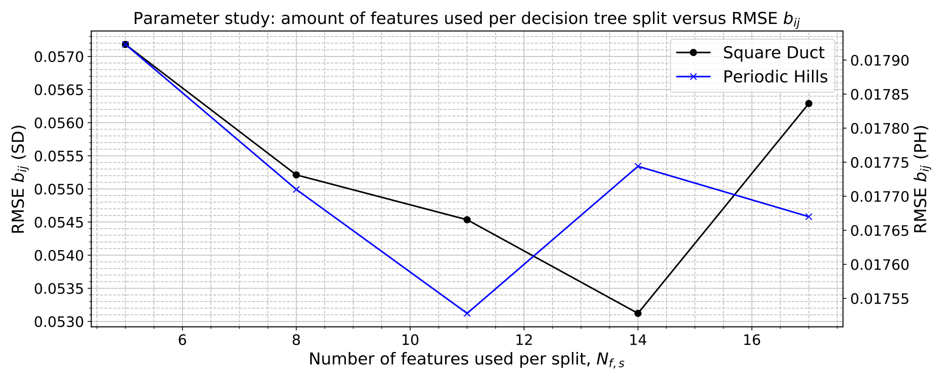


Figure 6.3: Parameter study: influence of the amount of features used for splitting

Using all features for selecting the splitting feature, which in this case corresponds to $N_{f,s} = 17$, is not optimal either. This is due to the fact that the TBDT's in this case are more similar to each other, i.e. the correlation between the trees is higher. As can be seen there is a trade-off between decreasing correlation in the random forest on one hand, and decreasing the variance of the decision trees on the other hand, looking at the expression for the variance of the random forest in (3.26).

For the periodic hills, a similar pattern is found as for the square duct, but in this case an optimum is found around $N_{f,s} = 11$. By looking at the average performance of both test cases it was decided to choose a value of $N_{f,s} = 11$ for the TBRF algorithm. This is different from the value usually recommended for the standard random forest, which is the total amount of features divided by three (Hastie et al., 2008), which in this case would correspond to $N_{f,s} = 6$.

Minimum samples per leaf node The effect of the minimum amount of samples in the TBDT leaf nodes was investigated, for which the results are presented in Figure 6.4. Figure 6.4 presents the results for the square duct flow, indicated by the black line with dots (left y-axis), and for the periodic hills, indicated by a blue line with crosses (right y-axis). While the RMSE for the square duct keeps decreasing up to $N_{s,l} = 13$, the RMSE for the periodic hills increases after $N_{s,l} = 9$. Roughly speaking there seems to be an optimum when looking at the average of both flow cases around $N_{s,l} = 9$, which was therefore chosen as the value to be used.

Increasing the amount of samples per leaf node greatly reduces computational cost of both training and predicting, which allows for the use of bigger data-sets compared to a fully grown tree. As an example, it was observed that the tree file sizes, containing the entire tree structure, approximately reduces by a factor of 5 when comparing the fully grown tree to the tree with nine samples per leaf (at least for the flow cases used in this chapter). The selected value of $N_{s,l} = 9$ is slightly larger than the value recommended for normal random forests used for regression, which is $N_{s,l} = 5$ (Hastie et al., 2008).

The fact that the selected values for $N_{s,l}$ and $N_{f,s}$ are higher for the TBRF compared to

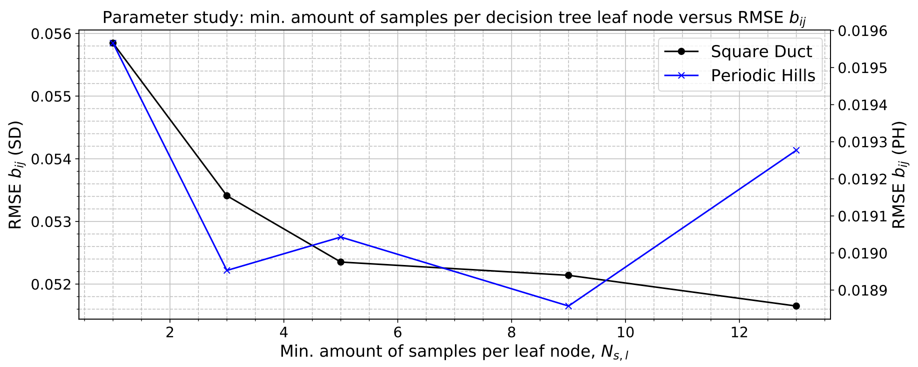


Figure 6.4: Parameter study: influence of the minimum amount of samples per TBDT leaf node

the standard random forest, might be because of the fact that it is more prone to predicting outliers. Since the tensor basis coefficients as predicted by the TBRF multiply the basis tensors, it is possible that the predicted values for b_{ij} will lie far outside of the range of values for b_{ij} present in the training data. This could mean it is important for the TBRF to have a larger set of splitting features to choose an appropriate splitting feature and value from, and to have more samples in the leaf nodes to calculate the least squares fit for $g^{(m)}$ in order to avoid strong outliers.

6.2 Predictions for the anisotropy tensor

In this section predictions for the anisotropy tensor will be presented, which consists of comparing the individual components of the tensor, and comparing results by the use of the barycentric map and in turn the RGB-map derived from the barycentric map.

The data-sets used for training and testing as presented in this section are presented in Table 6.1. The flow cases have been abbreviated, where PH stands for periodic hills, CD for the converging-diverging channel, CBFS for the curved backward facing step, BFS for the backward facing step, and SD for the square duct. The number behind the abbreviation indicates the Reynolds number. The table also presents the amount of samples used for training, $N_{sampl.}$ (randomly sampled from the total data-set), and the amount of usable features present in the training and test data-sets, $N_{feat.}$. The feature sets from which the features are derived are presented as well, where the exact features can be found in Table 3.1. For all cases the $k - \omega$ turbulence model was used for the RANS simulations.

Table 6.1: Data-sets used for training and testing

Case nr.	Training cases	Test cases	$N_{sampl.}$	$N_{feat.}$	Feature sets
C1	PH5600, PH10595, CD12600	CBFS13700	21,000	17	FS1, FS2, FS3
C2	PH5600, PH10595, CD12600	BFS5100	21,000	17	FS1, FS2, FS3
C3	PH5600, PH10595, CD12600	SD3500	21,000	5	FS1
C4	PH5600, PH10595, CD12600	SD3500	21,000	17	FS1, FS2, FS3
C5	PH5600, PH10595	CD12600	21,000	17	FS1, FS2, FS3

First the results for the curved backward facing step will be presented (C1), which is relatively similar to the training cases for which reliable data was available (periodic hills and the coverging-diverging channel). Afterwards results for the backward facing step will be presented (C2), which features stronger separation compared to the training cases and it therefore likely to feature more extrapolation. Afterwards results for the square duct will be presented. A comparison will be made for the case using only features based on $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$ as was done in Ling et al. (2016b) (C3), and a case where all available features are used (C4). Finally predictions for the converging-diverging channel will be presented, using only the periodic hills as training data (C5).

6.2.1 Curved Backward Facing Step

For the curved backward facing step, case C1 in Table 6.1, highly resolved LES data for the Reynolds stress was available for the entire flow field. This means that results can be analyzed by looking at the individual tensor components of b_{ij} in the entire flow field, as well as analyzing results using the barycentric map and the RGB-map corresponding to the barycentric map.

Figure 6.5 presents the four non-zero unique components of b_{ij} for the curved backward facing step, as given by the LES data, the RANS ($k-\omega$) simulation, and the TBRF and TBNN algorithms. What can be seen first of all, is that the predictions from the TBRF and TBNN are significantly different from the LES in the middle of the duct. The LES data in the middle of the duct is quite noisy, and when inspecting the type of stress mainly 2-component turbulence is seen, whereas 3-component turbulence is expected, see the discussion in Section 4.3. For further comparison results closer to the step will be used, where the LES data is smooth and compares well with the predictions from the machine learning algorithms.

Both machine learning algorithms give accurate predictions closer to the step, and predictions are relatively smooth. The RANS simulation only gives reasonable predictions for the b_{12} component. In the linear eddy viscosity hypothesis it is assumed that the anisotropy tensor is aligned with the mean rate of strain tensor. This is clearly not a good approximation for the anisotropy of the normal stresses, and only produces a viable result for the b_{12} component which is based on $\partial\bar{u}/\partial y$. On top of the step the b_{11} component is captured well by the TBRF and TBNN algorithms, as well as the b_{33} component after the step. What furthermore can be seen is that the TBRF and TBNN algorithms give predictions similar to the RANS

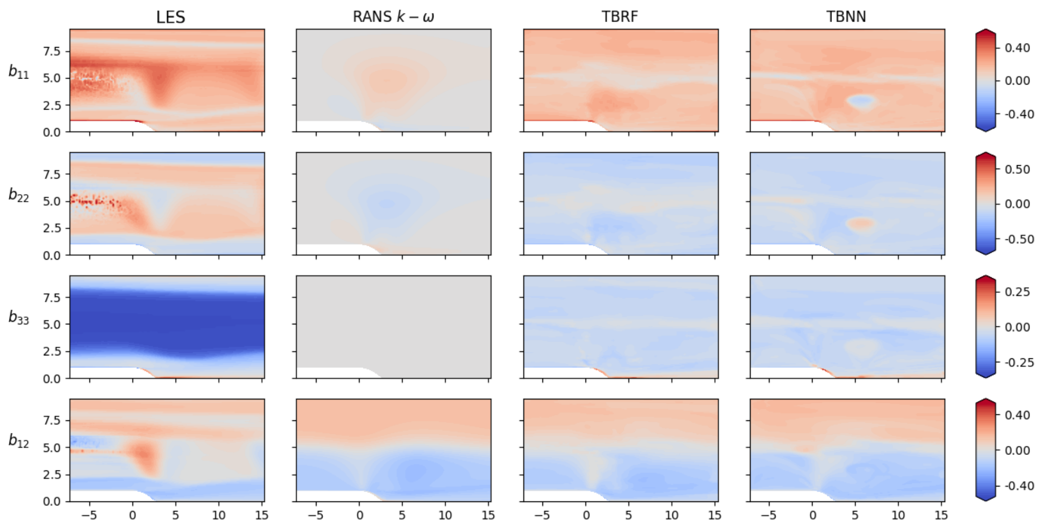


Figure 6.5: Curved Backward Facing Step, b_{ij} components from DNS, RANS, TBRF, and TBNN

turbulence model for the b_{12} component.

Results closer to the step will be analysed further, for which the domain is presented in Figure 6.6. Three sections which are analyzed more closely on the barycentric map are indicated with the dashed lines, located at $x/h = 2$, $x/h = 3$, and $x/h = 4$, which are located in the front, middle, and aft part of the separated region.

The different stress types in the domain are plotted in Figure 6.7 using the RGB map. As can be seen, the LES data shows 1-component turbulence on top of the step, which bulges out at the location where the shear layer separates. In Bentaleb et al. (2012) it is noted that production of the streamwise fluctuation is strongly increased at the shear layer separation location due to shear-generation, leading to the 1-component state of turbulence. When this shear layer diverges from the wall, the turbulence evolves more towards the 3-component state due to the redistribution process. On the curved part of the step and the bottom wall after the step 2-component turbulence can be observed, and 3-component turbulence can be observed more towards the center of the duct. This 3-component turbulence, or the isotropic state of turbulence, is also present in the recirculation region away from the walls where the flow moves more slowly.

The RANS simulation mainly predicts turbulence in the 3-component region. The TBRF algorithm accurately captures the turbulent state as given by the LES data: 1-component turbulence can be seen on top of the hill and at the separation location, it accurately predicts the 2-component state on the curved part of the walls and on the bottom wall after the step, and 3-component turbulence can be observed in the recirculation region. Some noise is visible however, most notably around $x/h = 0.0$ to $x/h = 1.0$ away from the wall. The TBNN algorithm captures the different types of turbulence accurately as well. Close to the wall on top of the step it captures the 1-component turbulence a bit less accurately compared to the TBRF algorithm, and some spurious patterns are visible above the step and close to

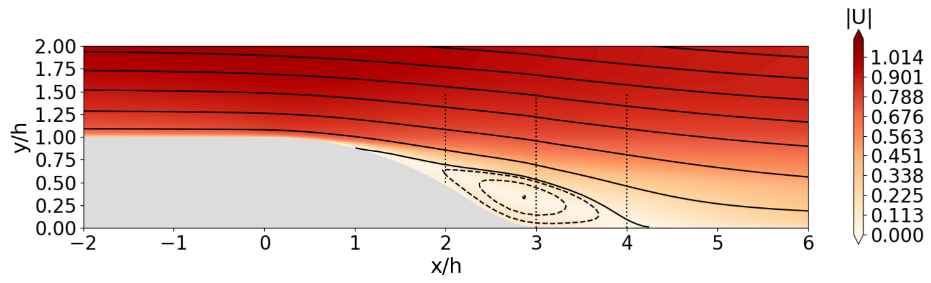
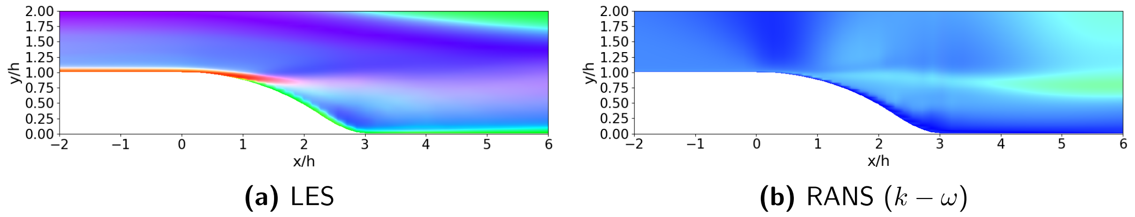


Figure 6.6: Domain used in Figure 6.7, along with the section locations used in Figure 6.8 (dashed lines)



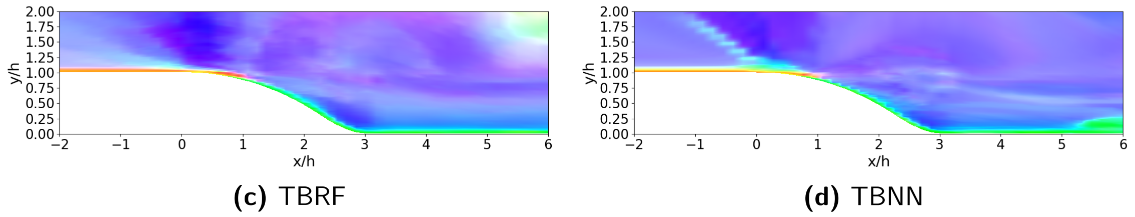
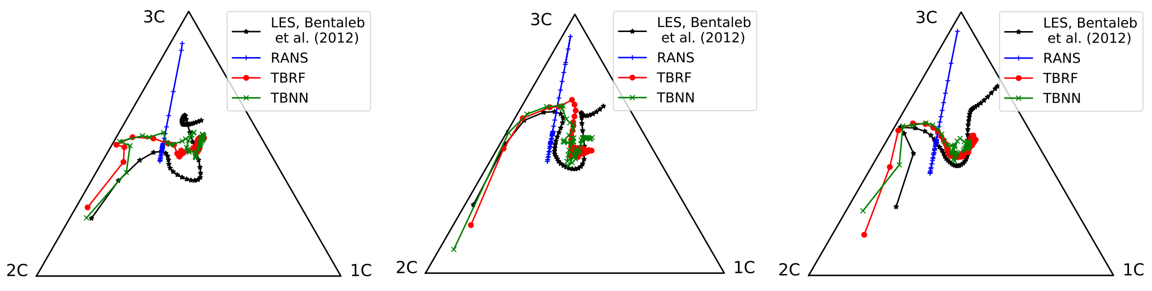


Figure 6.7: RGB map of the stress type in the flow domain based on the barycentric map, curved backward facing step

the lower wall around $x/h = 6.0$.

Barycentric map predictions for the three sections are presented in Figure 6.8. Results are presented for the LES data (black asterisks), the RANS simulation (blue plusses), the TBRF predictions (red dots), and the TBNN predictions (green crosses). As can be seen both machine learning algorithms are quite close to the LES reference data. They accurately capture the 2-component turbulence close to the wall, and move towards the 3-component corner when moving away from the wall. Slightly more discrepancy can be seen when moving upwards away from the wake, where the LES data indicates more 3-component turbulence compared to the machine learning algorithms. Predictions are close nonetheless, especially compared to the RANS simulation.



(a) x/h = 2.0 **(b)** x/h = 3.0 **(c)** x/h = 4.0

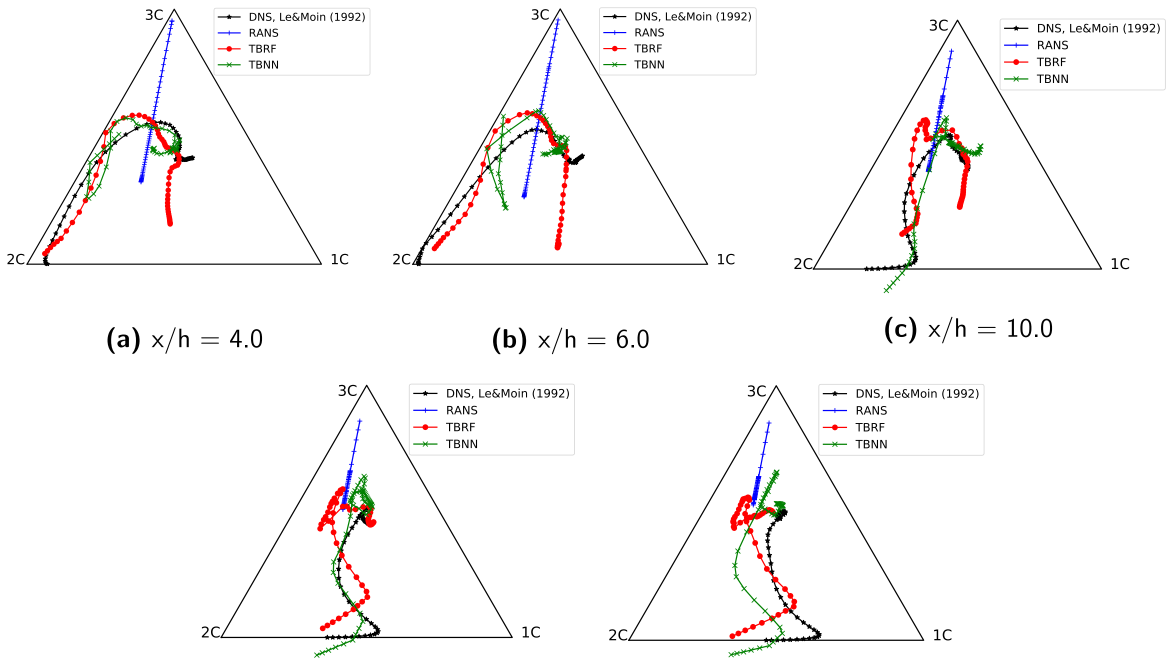
Figure 6.8: Barycentric map locations along three sections for the curved backward facing step flow

6.2.2 Backward Facing Step

No reliable DNS/LES data could be found for the full Reynolds stress field of the backward facing step flow case (C2, Table 6.1). From Le and Moin (1992) DNS data was available for five different sections at specified x/h locations for the Reynolds stresses and velocities however, and from Jovic and Driver (1994) experimental results were available for some of the Reynolds stresses, the velocities, and the skin friction coefficient (at a very slightly different Reynolds number of $Re = 5000$). In this section, the barycentric map will be analysed for the five available sections, after which predictions of the TBRF and TBNN algorithms will be plotted using the RGB map together with the RANS simulation.

Locations on the barycentric map of the five sections for which reliable DNS data was available are plotted in Figure 6.9. The Figures show results for DNS (black asterisks), RANS (blue plusses), the TBRF algorithm (red dots), and the TBNN algorithm (green crosses). The sections range from y/h = -1 (bottom wall) to y/h = 0 (location of the step). Results indicate that the machine learning algorithms are able to give an accurate description of the anisotropy tensor. For x/h = 4 and x/h = 6 predictions close to the wall are more accurate for the TBRF algorithm compared to the TBNN algorithm. Moving away from the wall into the shear layer the TBRF algorithm predicts a mixture of 1-component and 2-component stress, whereas a mixture of all three components is given by the DNS data. The TBNN algorithm is more accurate around the shear layer in predicting this mixture of all three turbulent states. For x/h = 10, x/h = 15, and x/h = 19 it can be seen that close to the wall predictions match the DNS data closely for both the TBRF and TBNN algorithm. Predictions show a similar pattern to the DNS data when moving away from the wall: at similar locations in the flow field, similar movements on the barycentric map can be observed for the machine learning algorithms compared to the DNS data. It is as if the profiles from the machine learning algorithms have been stretched with respect to the DNS data, especially for x/h = 15, and x/h = 19.

Stress types in the flow domain are presented in Figure 6.10. No DNS data is plotted since no reliable Reynolds stresses were available for the entire flow field. As can be seen both the



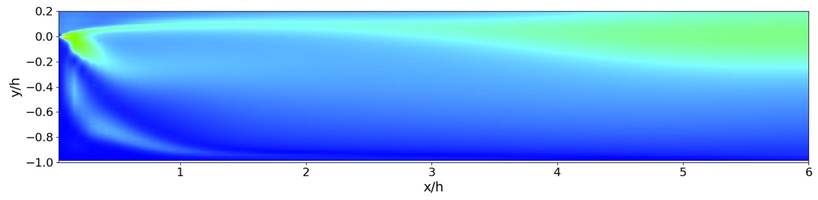
(a) x/h = 4.0 **(b)** x/h = 6.0 **(c)** x/h = 10.0

(d) x/h = 15.0 **(e)** x/h = 19.0

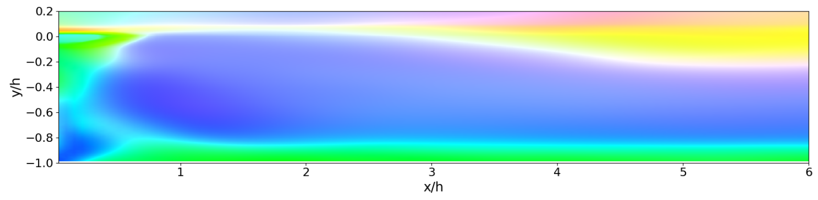
Figure 6.9: Barycentric map locations along five sections for the backward facing step flow

TBRF and TBNN algorithms predict 2-component stresses close to the lower wall and next to the wall of the step. Just after the step the TBNN algorithm predicts some non-realizable stresses going over the 1-component limit. The TBRF does a good job at filtering out some of the extreme non-realizable outliers, resulting in more plausible stresses just after the step. In the shear layer the TBRF predicts a mixture of 1-component and 2-component stresses, and the TBNN algorithm predicts more of a mixture of all three turbulent limiting states.

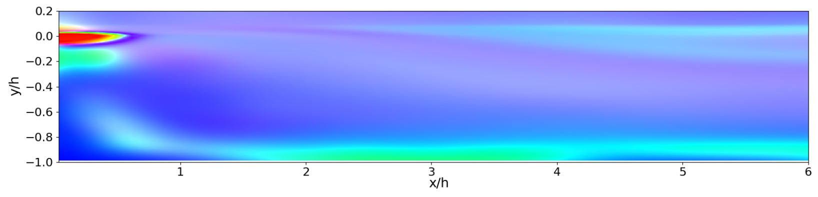
Previously in Figure 6.9 it was observed that around y/h = 0 at x/h = 4.0 and x/h = 6.0, the TBRF algorithm overpredicts the amount of 1-component and 2-component turbulence, whereas the TBNN algorithm correctly predicts a mixture of the three limiting states of turbulence. It is therefore likely that the TBNN algorithm is giving a more accurate description in the shear layer in this case. It was also observed however that close to the wall the TBRF algorithm more accurately predicts 2-component turbulence, which makes it likely that the layer of 2-component turbulence in Figure 6.10b next to the wall is more accurate than that in Figure 6.10c.



(a) RANS $(k - \omega)$



(b) TBRF



(c) TBNN

Figure 6.10: RGB map of the stress type in the flow domain based on the barycentric map, backward facing step

6.2.3 Square Duct

Figure 6.11 presents the six different components of the anisotropy tensor for the square duct case, for DNS, RANS ($k-\omega$), predictions from the TBRF, and predictions from the TBNN. Both the predictions by the machine learning algorithms for case C3 and C4 (see Table 6.1) are included in the figure.

As can be seen, the $k-\omega$ turbulence model only yields non-zero predictions for the b_{12} and b_{13} components. This can be explained by looking at the Boussinesq approximation, as these components are based on the $\partial \bar{u}/\partial y$ and $\partial \bar{u}/\partial z$ velocity gradient components, which are the only non-zero components in the RANS simulation.

When looking at the predictions by the machine learning algorithms, it can clearly be seen that the introduction of the extra features leads to a more accurate prediction of the Reynolds stress anisotropy in the flow domain. For case C3, the anisotropy of the Reynolds stress is not captured very well close to the walls when looking for example at the b_{11} component. It also shows more underprediction of the magnitude of the b_{12} and b_{13} components for both the TBNN and TBRF algorithms. In all cases the magnitude of b_{23} is overpredicted by the machine learning algorithms, which can be explained due to the fact that this component is much smaller than the other five. When looking at case C4, it can be observed that the TBRF algorithm gives slightly better results for the anisotropic normal stress components b_{11}, b_{22}, and b_{33}. It is a bit more consistent compared to the TBNN algorithm, which shows some slightly spurious patterns when looking for example at b_{22} and b_{33}. For the anisotropic shear stresses the TBNN algorithm is more accurate in capturing the right magnitude however. The root mean square error (RMSE) of the anisotropy tensor with respect to the DNS data is given in Table 6.2. The RMSE's are lower when introducing more features (C4) for both algorithms. It can also be seen that the TBNN performs better compared to the TBRF when using only the five features based on $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$, and that the TBRF performs better compared to the TBNN when expanding the amound of features. This difference can be explained due to the fact that the TBNN hyperparameters were optimized using the five features based on $\hat{S}_{ij}$ and $\hat{\Omega}_{ij}$ in Ling et al. (2016b). It is possible that the TBNN might perform better when redoing the hyperparameter optimization using all features present in case C4.

Figure 6.12 presents the stress type using the RGB-map for the square duct flow case. It presents the DNS data, results from the $k-\omega$ RANS simulation, and predictions by both the TBRF and TBNN algorithms for case C3 and C4. As can be seen the DNS data shows 1-component stress near the walls, which goes towards 3-component stress near the centreline of the duct. The RANS simulation only predicts stresses near the 3-component limit. When looking at case C3, it can be seen in Figure 6.12c that the TBRF algorithm mainly predicts

Table 6.2: Comparison TBRF and TBNN with respect to the RMSE of b_{ij}, square duct flow case

	TBRF	TBNN
RMSE, C3:	0.0995	0.0871
RMSE, C4:	0.0521	0.0681

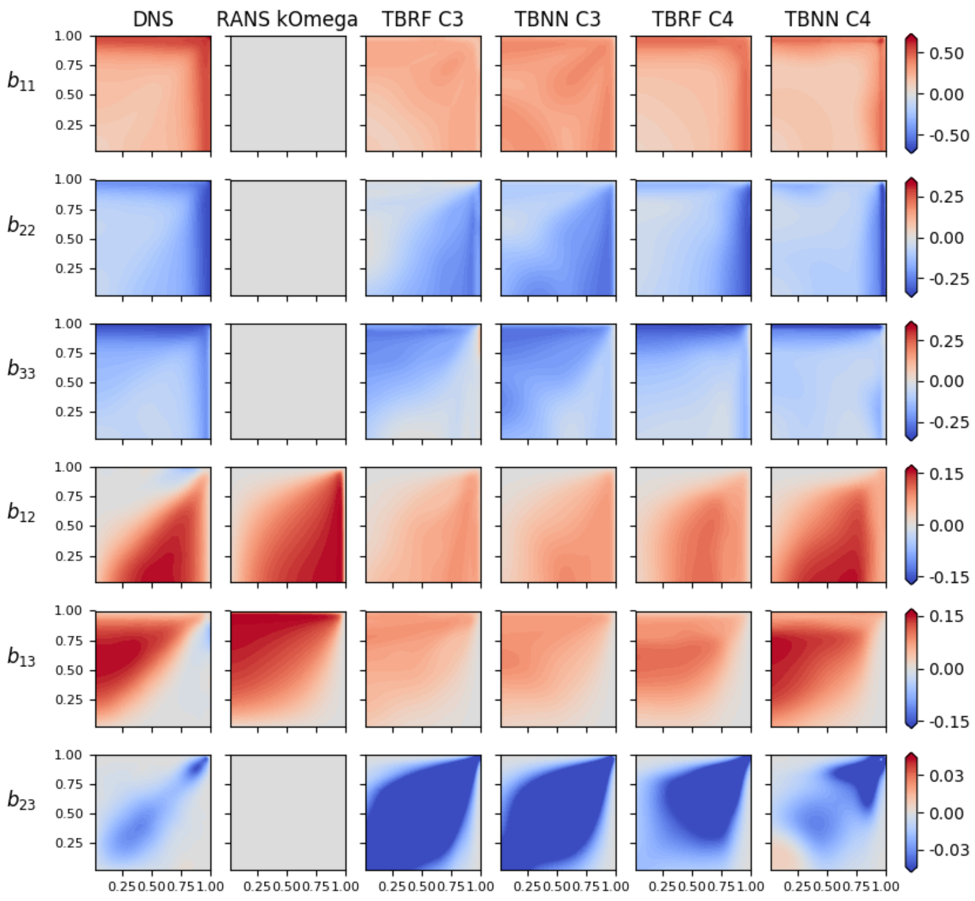
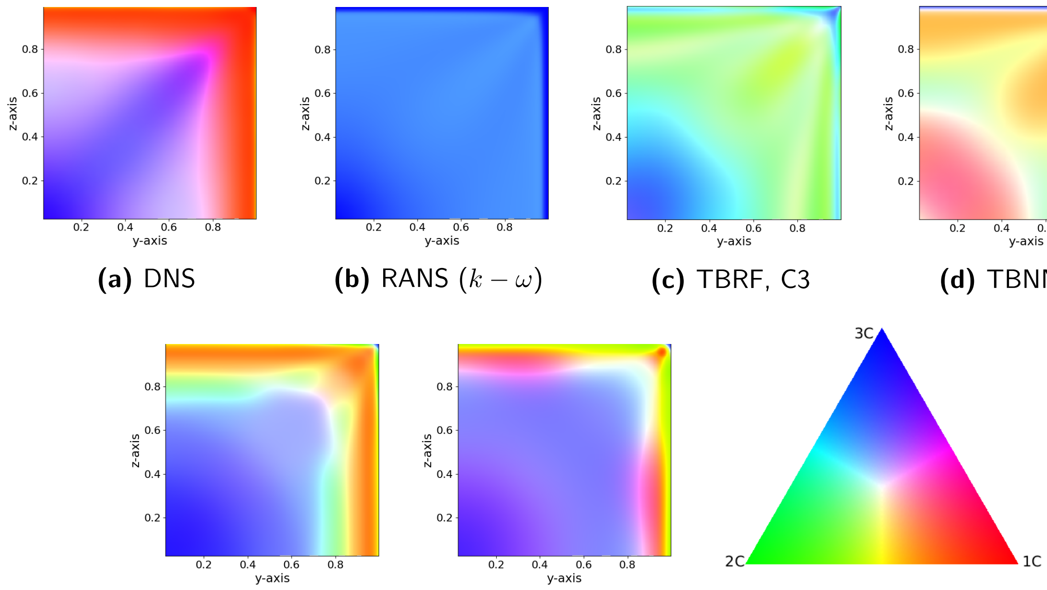


Figure 6.11: Square duct, b_{ij} components from DNS, RANS, TBRF, and TBNN



(a) DNS **(b)** RANS $(k-\omega)$ **(c)** TBRF, C3 **(d)** TBNN, C3

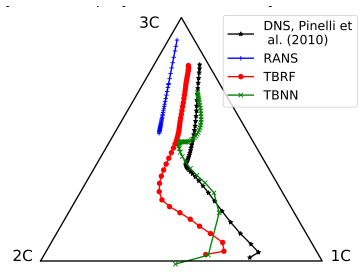
(e) TBRF, C4 **(f)** TBNN, C4 **(g)** Legend for the RGB colormap

Figure 6.12: RGB map of the stress type in the flow domain based on the barycentric map, square duct

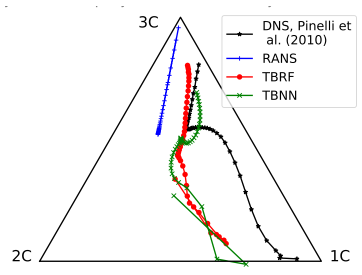
2-component stresses near the wall, and 3-component stresses near the centreline and at the corner of the duct. The TBNN algorithm, for which results are presented in Figure 6.12d, predicts more 1-component stresses at the centreline, a mixture with more 1-component and 2-component stresses closer to the wall, and 3-component stresses next to the wall, which is the opposite the DNS data indicates.

It can be observed in Figure 6.12e and Figure 6.12f that results improve significantly when introducing the extra features from case C4. The TBRF algorithm predicts a mixture of 1-component and 2-component stresses close to the wall, which is distributed evenly along the entire length of the wall. Compared to the DNS data the TBRF algorithm predicts too much 2-component stresses when moving towards the centreline of the duct, where the DNS data moves from the 1-component corner to the 3-component corner directly. The TBNN algorithm predicts the 1-component stresses close to the wall and at a small spot in the corner more accurately, but it is distributed less evenly along the length of the wall compared to the TBRF algorithm, and more 2-component stresses are visible next to the wall.

Figure 6.13 presents the barycentric map along two sections in the square duct for data-set C4. The first section in Figure 6.13a is given along the horizontal line ranging from the centerline of the duct at (y,z) = (0.0,0.0), to (y,z) = (1.0,0.0). The second section in Figure 6.13b is given along the diagonal, from (y,z) = (0.0,0.0), to the upper right corner of the duct at (y,z) = (1.0,1.0). As can be seen predictions from the machine learning algorithms are quite accurate when looking at the horizontal section, with the turbulence going from close to the 3-component limit, towards the 1-component limit. Both algorithms somewhat underpredict the amount of 1-component turbulence with respect to the 2-component turbulence. Predictions



(a) (y,z) = (0.0,0.0) to (y,z) = (1.0,0.0)



(b) (y,z) = (0.0,0.0) to (y,z) = (1.0,1.0)

Figure 6.13: Barycentric map locations along two sections for the square duct flow

by the machine learning algorithms show more discrepancy on the barycentric map when going towards the corner, as can be seen in Figure 6.13b. The algorithms have difficulty predicting turbulence close to the 1-component limit. When looking at the results closely, it can be seen that at the very corner there is a jump back towards the middle of the barycentric map, which is represented by tiny blueish spot in the corner in Figure 6.12f and Figure 6.12e.

6.2.4 Converging-Diverging Channel

Figure 6.14 presents the individual components of the anisotropy tensor for the converging-diverging channel as given by the DNS data, RANS $k-\omega$ simulation, and the TBRF and TBNN algorithms corresponding to case number C5 in Table 6.1. Right at first sight it can be observed that the machine learning predictions show considerable more noise compared to the previous flow cases. Most of this noise in concentrated around the middle of the converging part of the channel. In the rest of the domain the machine learning algorithms are able to give reasonable predictions for the anisotropy tensor components.

The noise can also clearly be observed when using the RGB-colormap based on the barycentric map, for which results are presented in Figure 6.15. A lot of noise is present on top of the hills when looking at the TBRF and TBNN predictions. What also can be observed, is that the machine learning algorithms underpredict the amount of 1-component turbulence in the front part of the channel, and instead predict 2-component turbulence. Both algorithms predict some 1-component turbulence on top of the hill near the separation region. Furthermore predictions in the aft part of the channel give a reasonable match to the DNS data, with 2-component turbulence close to the wall, and 3-component turbulence in the middle of the channel.

Discrepancies can largely be explained due to a high amount of extrapolation, causing decreased performance of the machine learning algorithms. This can be substantiated by looking at the a priori confidence parameters for this flow case as introduced in Section 3.4. The normalized Mahalanobis distance ($M_{norm.}$) and the KDE-distance (D_{KDE}) are presented for the flow domain in Figure 6.16. The areas of high extrapolation, i.e. the areas for which the training data is less representative, are indicated in dark red. As can be seen these areas

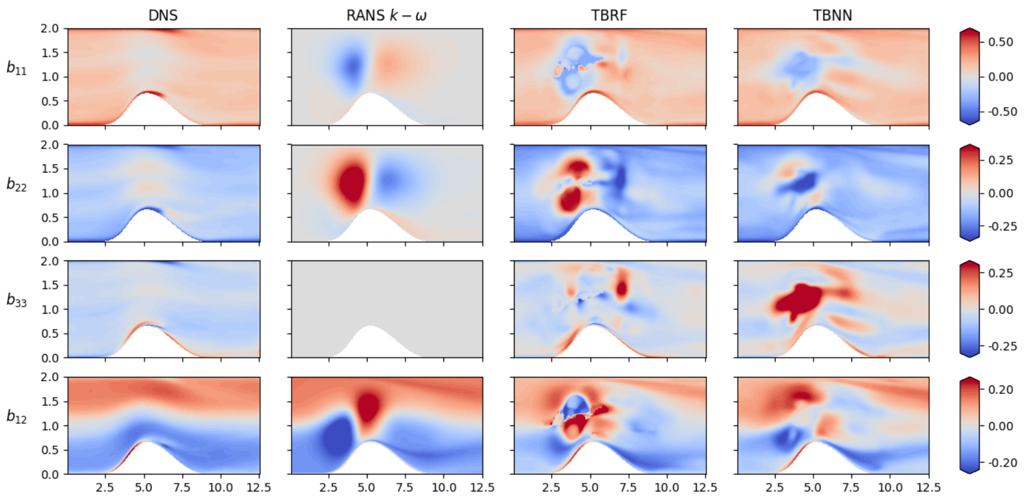
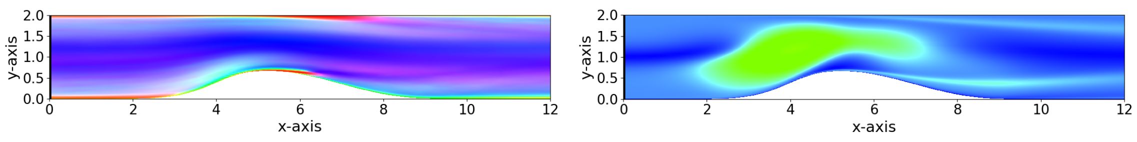
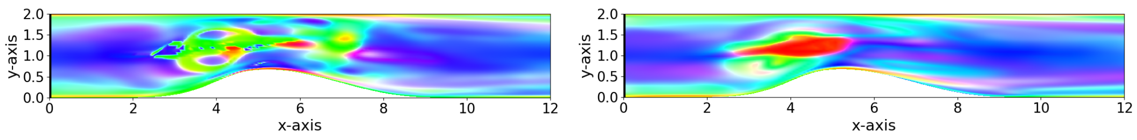


Figure 6.14: Converging-diverging channel, b_{ij} components from DNS, RANS, TBRF, and TBNN



(a) DNS

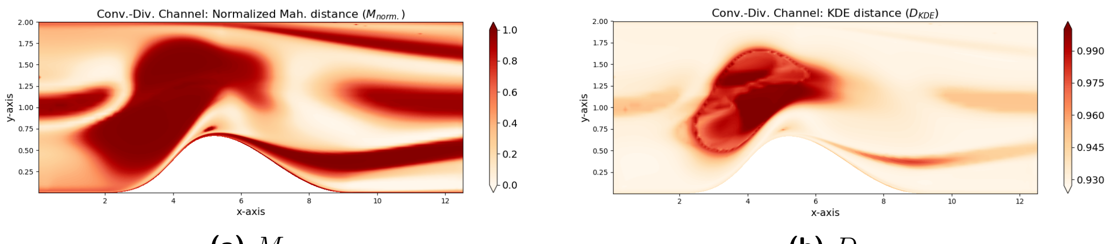
(b) RANS $(k-\omega)$



(c) TBRF

(d) TBNN

Figure 6.15: RGB map of the stress type in the flow domain based on the barycentric map, converging-diverging channel



(a) $M_{norm.}$ **(b)** D_{KDE}

Figure 6.16: A priori confidence parameters plotted for the converging-diverging channel

show a match to the areas with high error in the converging part of the channel. This is especially true for the KDE-distance, which should give a better representation of the training data distribution compared to the normalized Mahalanobis distance.

By looking at the features of the converging-diverging channel, one can further investigate why the extrapolation is so severe in front of the hill. In Appendix F, Figure F.3, the features for the converging-diverging channel are plotted. As can be seen, there is a very close match between D_{KDE} and feature FS1,1 in Figure F.3. The value of this feature is a lot larger compared to the training data, since first of all the Reynolds number of the flow is larger ($Re = 12,600$ versus $Re = 10,595$ and $Re = 5,600$), along with the fact the flow accelerates in the converging part of the duct. This will increase the velocity gradients and the strain rate tensor in the flow, thus increasing the value of the first feature, trace(S^2).

As will be shown later in Section 8.1, feature FS1,1 is one of the most important features for the TBRF algorithm. This means that when the algorithm is extrapolating with respect to feature FS1,1, it is likely this will cause errors in the predictions.

6.2.5 Comparison with the standard RF

It was verified whether the predictions of the TBRF algorithm adhere to Galilean invariance by checking the condition

$$\mathbf{Q}\, b_{ij}(\hat{\mathbf{S}}, \hat{\mathbf{\Omega}}, \nabla \mathbf{k}, \nabla \mathbf{p}, ...)\, \mathbf{Q}^T = b_{ij}(\mathbf{Q}\hat{\mathbf{S}}\mathbf{Q}^T, \mathbf{Q}\hat{\mathbf{\Omega}}\mathbf{Q}^T, \mathbf{Q}\nabla \mathbf{k}, \mathbf{Q}\nabla \mathbf{p}, ...), \tag{6.1}$$

for all vectors and tensors serving as input features. As a comparison results from a normal random forest algorithm are presented, which does not adhere to Galilean invariance. Results are presented in Figure 6.17, where the coordinate system was rotated around the z-axis with 30°. As can be seen, predictions from the TBRF algorithm adhere to (6.1): all four non-zero components are the same, independently of either rotating predictions from the original frame of reference, or rotating the input vectors/tensors.

For the standard random forest algorithm predictions differ, except from the b_{33} component.

This is due to the fact that this is the component acting in the z-axis, which served as the rotation axis. Since the predictions of b_{ij} are not properly aligned with the coordinate system, as for example done by introducing the tensor basis, the normal random forest is not useful for predicting tensor quantities when training and testing on different flow cases. Only when one would find a suitable method for rotating the eigenvectors which is Galilean invariant this might be possible.

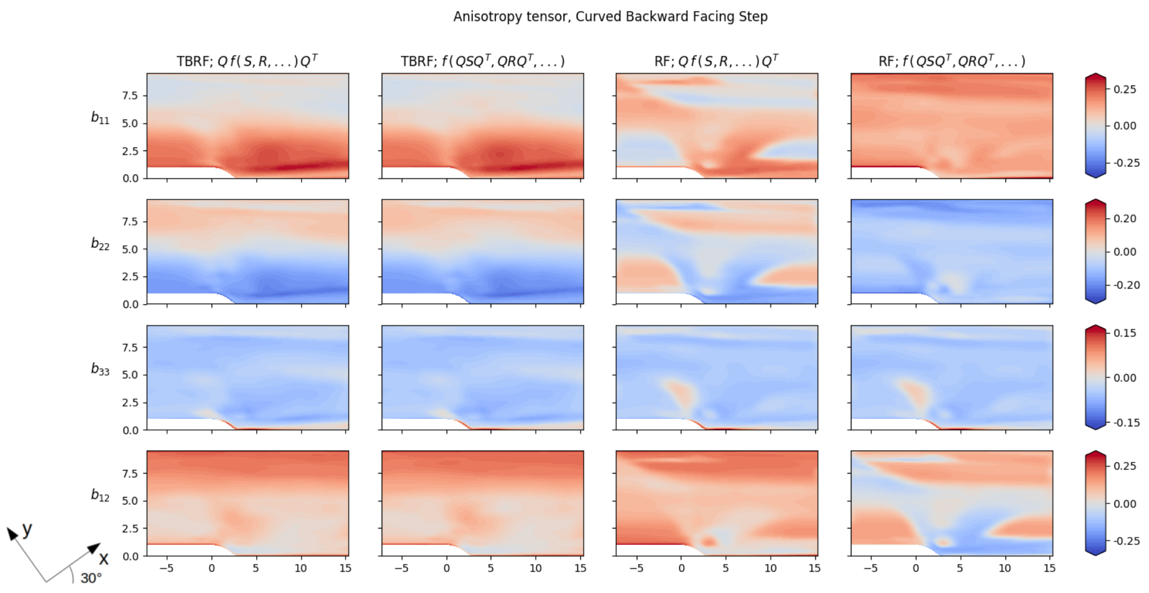


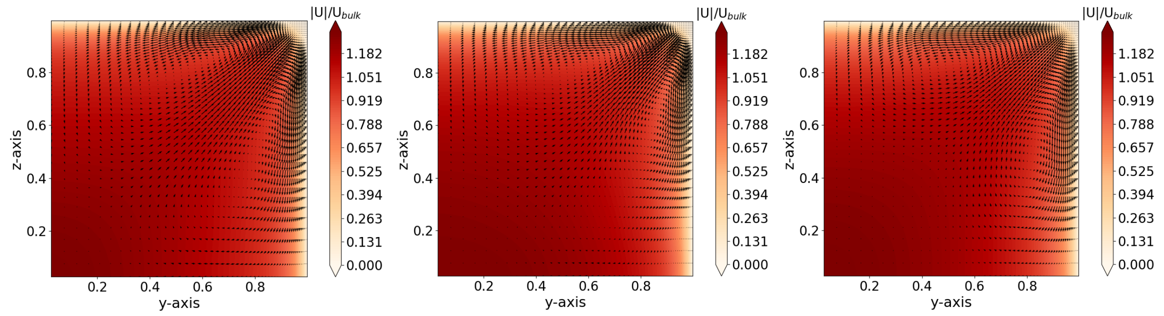
Figure 6.17: Comparison invariance of the TBRF and RF when predicting b_{ij} for the Curved Backward Facing Step

6.3 Propagating the anisotropy tensor

This section presents the results of the propagated anisotropy tensor from the square duct flow case and the backward facing step flow case. Results were propagated using the continuation solver presented in Section 3.9.

6.3.1 Square Duct

Figure 6.18 compares the flow field as given by the DNS data to propagations using the continuation solver, at a Reynolds number of 3,500. Figure 6.18a presents the square duct flow case as given by the DNS data, where the velocity magnitude is indicated by the colormap, and the quivers show the secondary flow motion. In Figure 6.18b the anisotropy tensor from the DNS data was propagated to verify the continuation solver. As can be seen results are virtually the same. There is a slight difference as the propagated flow field is not perfectly symmetric, due to the fact that the anisotropy tensor from the DNS data-set is not perfectly symmetric. The propagated flow field for the anisotropy tensor predicted by the TBRF algorithm ($b_{ij,TBRF}$), as shown in Figure 6.11 in the fifth column, is presented in Figure 6.18c. Results are close to the given DNS data, with an accurate representation of the secondary flow and velocity magnitude. Since predictions for the anisotropy tensor are symmetric, the propagated flow field is as well. The biggest discrepancies are seen near the centerline of the duct, where the magnitude of the secondary flow motions is underpredicted.

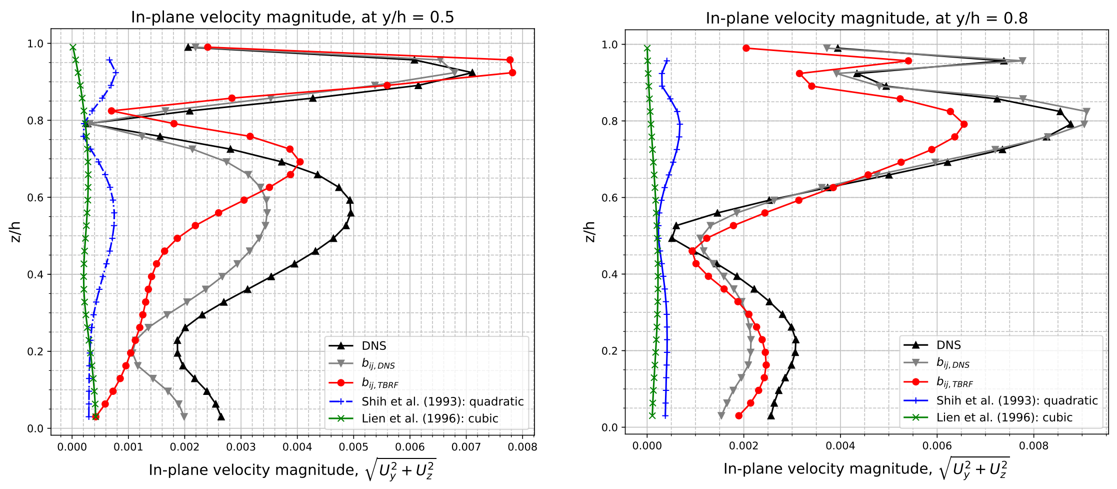


(a) DNS **(b)** $b_{ij,DNS}$ **(c)** $b_{ij,TBRF}$

Figure 6.18: Comparison DNS and the propagated flow fields using the continuation solver

In order to be better able to quantify the results, the in-plane velocity magnitude along two sections in the square duct is plotted in Figure 6.19. The vertical section located at $y/h = 0.5$, ranging from the centreline of the duct ($z/h = 0$) to the top wall ($z/h = 1.0$) is presented in Figure 6.19a. Results are presented for the DNS data, the propagated flow field using $b_{ij,DNS}$, and the propagated flow field using $b_{ij,TBRF}$, all corresponding to the results presented in Figure 6.18. Furthermore, results from the quadratic eddy viscosity model by Shih et al. (1993) are presented, as well as results from a cubic eddy viscosity model from Lien et al. (1996), both of which are turbulence models available in OpenFOAM. As can be seen, the propagated velocity fields using $b_{ij,TBRF}$ (the red dots) and $b_{ij,DNS}$ (the grey triangles) both show the most discrepancy around $z/h = 0.5$, which was also visible in Figure 6.18. For $b_{ij,TBRF}$ this discrepancy is larger, with the peak of the maximum in-plane velocity magnitude shifted upwards, and underprediction of the velocity magnitude at the centreline of the duct, at $z/h = 0.0$. What is also clear however, is that predictions are still far more accurate than both non-linear eddy-viscosity models. The model from Shih et al. (1993) is able to predict the peaks of the in-plane flow magnitude quite accurately, but still significantly underpredicts the overall magnitude. Predictions by the cubic eddy-viscosity model from Lien et al. (1996) are far off overall.

Figure 6.19b presents the same analysis for the section located at $y/h = 0.8$. It can be seen that the propagated velocity field using $b_{ij,TBRF}$ is accurate in predicting the in-plane velocity magnitude. It slightly underpredicts the magnitude with respect to the DNS data and to the field using $b_{ij,DNS}$, but only slightly so, and the locations of the peaks match quite well. Again the non-linear eddy-viscosity models show significant discrepancy. The model from Shih et al. (1993) is able to predict the location of the largest peak in the in-plane velocity magnitude relatively well, but again underpredicts the magnitude along the entire section.



(a) y/h = 0.5 **(b)** y/h = 0.8

Figure 6.19: Comparison DNS and the propagated flow fields using the continuation solver

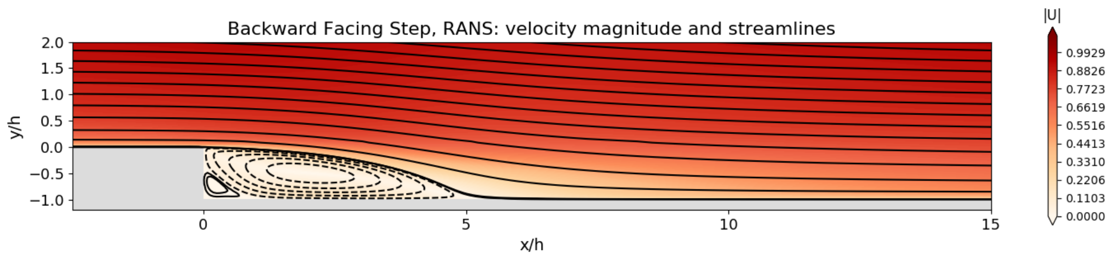
6.3.2 Backward Facing Step

Figure 6.20 presents streamlines in the flow field for the backward facing step, with results from the $k-\omega$ RANS simulation, the propagated velocity field using the predicted anisotropy tensor from the TBRF algorithm ($b_{ij,TBRF}$), and DNS data from Le et al. (1997). Since no DNS data was available for the velocity in the entire flow field, results for the DNS simulation are reproduced from Le et al. (1997). As can be seen, the size of the recirculation region is much more similar for the propagated velocity field using $b_{ij,TBRF}$ compared to the baseline RANS simulation. This also counts for the small counter-clockwise rotating recirculation region in the corner of the step. Further away from the wall the solver using $b_{ij,TBRF}$ does not seem to introduce any spurious effects and results are similar to the baseline RANS simulation. The reattachment point locations ($x_{reatt.}$) for all three cases are presented in Table 6.3. A significant improvement is shown for the propagated velocity field compared to the baseline RANS simulation.

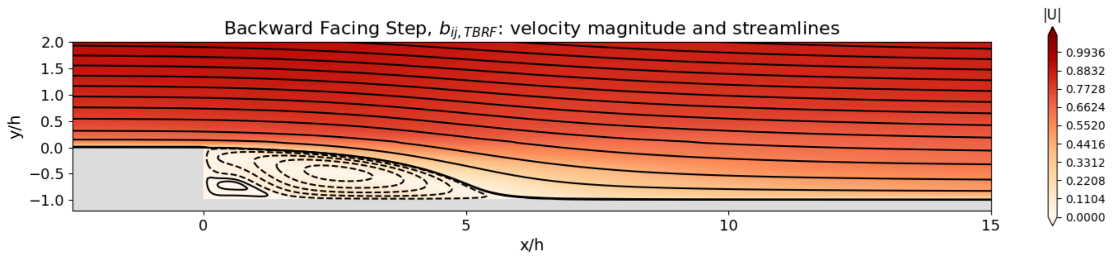
Table 6.3: Backward facing step, reattachment point location comparison

Model:	RANS	$b_{ij,TBRF}$	DNS (Le et al., 1997)
$x_{reatt.}$ [x/h]:	5.45	6.32	6.28

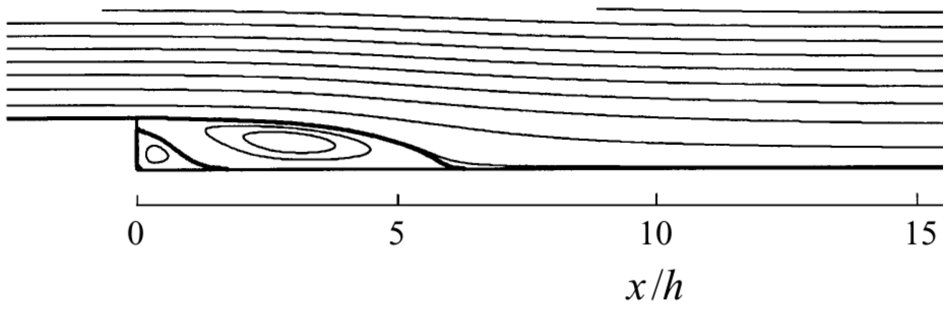
The skin friction coefficients from the RANS simulation and the propagated flow field using $b_{ij,TBRF}$ are compared to experimental data from Jovic and Driver (1994) in Figure 6.21. The propagated flow field shows a very close match to the experimental data, and the majority of results fall within the error bounds given by the experiment ($\pm 0.0005 c_f$). The reattachment point of the propagated flow field (6.32) compares well to the experimentally found reattachment point (6 ± 0.15).



(a) RANS ($k-\omega$)



(b) $b_{ij,TBRF}$



(c) DNS (reproduced from Le et al. (1997))

Figure 6.20: Comparison of the streamlines for the backward facing step: RANS, the propagated flow field using $b_{ij,TBRF}$, and DNS results from Le et al. (1997).

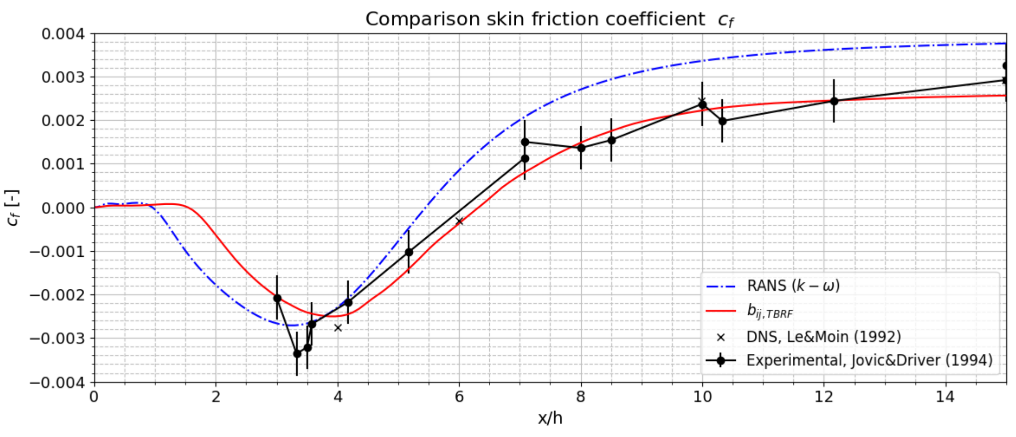


Figure 6.21: Skin friction coefficient for the backward facing step flow case, experimental data from Jovic and Driver (1994), DNS data from Le and Moin (1992)

Velocity profiles from DNS data (Le and Moin, 1992) are compared to the baseline $k-\omega$ RANS simulation and the velocity field obtained by propagating $b_{ij,TBRF}$ as presented in Figure 6.22. As can be seen the velocity profiles obtained by propagating $b_{ij,TBRF}$ are closer to the reference DNS data. Especially closer to the wall the propagated velocity field matches a lot better. The total RMSE for all five velocity profiles with respect to the DNS data amounts to 0.049 for the field obtained by propagating $b_{ij,TBRF}$, and to 0.094 for the baseline $k-\omega$ RANS simulation.

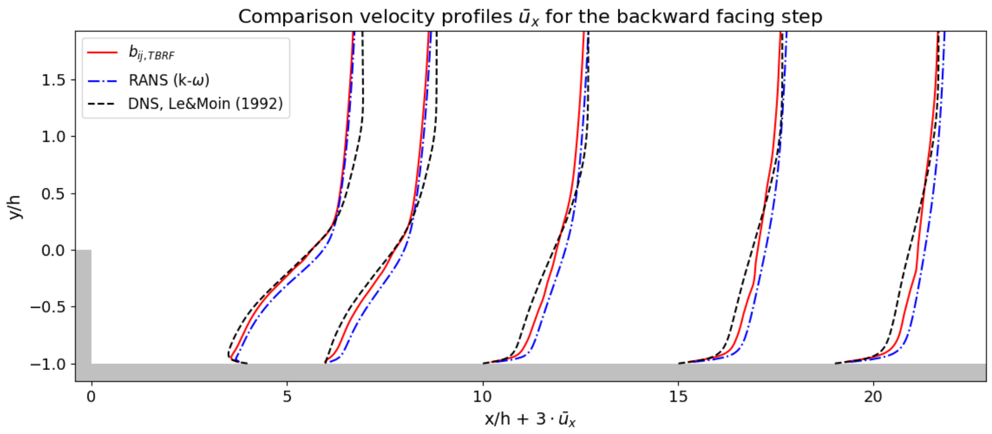


Figure 6.22: Velocity profiles for the backward facing step flow case, DNS data from Le and Moin (1992)

Chapter 7

Uncertainty Quantification

In this chapter the possibility of using the TBRF algorithm for uncertainty quantification will be explored. First, the distribution on the barycentric map will be analysed for the different TBDT's in the TBRF in Section 7.1. In Section 7.2 a Monte Carlo simulation will be done for various parameters of interest for the backward facing step, using different TBDT combinations. An analysis of whether it would be possible to use the variance of the TBDT's in the TBRF for creating an a posteriori confidence parameter will be presented in Section 7.3.

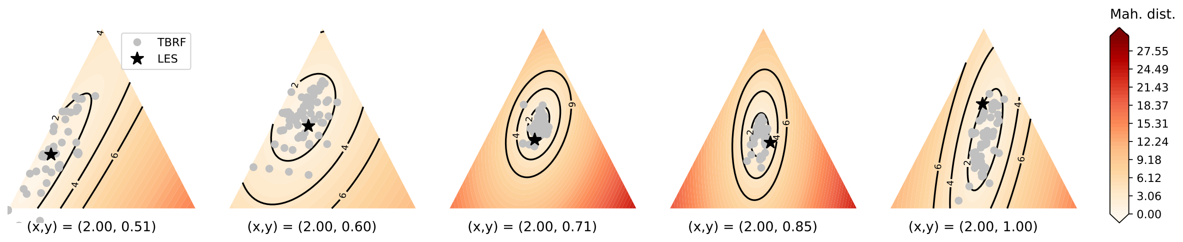
7.1 Decision tree samples

Since the TBRF consists of a large number of TBDT's, it is possible to say something about the certainty of a prediction at a given location. Some examples will be presented for the curved backward facing step and the backward facing step using the barycentric map. For every location in the flow domain the trees in the random forest can be analyzed for their variance on the barycentric map. This would make it possible to use an anisotropy eigenvalue perturbation methodology to quantify uncertainty such as proposed in Emory et al. (2013), using the spread of the confidence bounds from the TBDT's as the perturbation magnitudes.

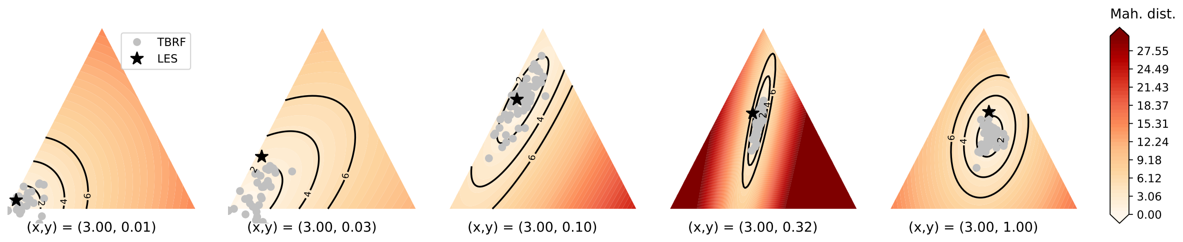
7.1.1 Curved Backward Facing Step

Along three sections (corresponding to Figure 6.8) in the curved backward facing step, 5 different points were taken to observe the distribution of the TBDT samples in the TBRF on the barycentric map. The samples were taken using a log-scale, with more points close to the wall. Outliers were filtered using the median filter introduced in Section 3.8.1 with a threshold of 3. Samples from the different TBDT's are shown in Figure 7.1 using grey dots. In order to visualize the distribution of the samples, it was assumed that the points are distributed according to a bivariate Gaussian distribution. This is not necessarily true, but is used to give an indication on the spread of the samples. The Mahalanobis distance M was calculated on all locations of the barycentric map, which gives an indication of how close these locations are to the distribution of the TBDT's. The triangles are coloured according to M, and furthermore isolines are plotted at $M = 2$, $M = 4$, and $M = 6$. As a reference, for 2 dimensions the 68%, 95% and 99.7% confidence intervals are given by $M \approx 1.52$, $M \approx 2.49$, and $M \approx 3.44$ respectively (Gallego and Cuevas, 2013). Samples from the LES data are plotted with a black asterisk as well.

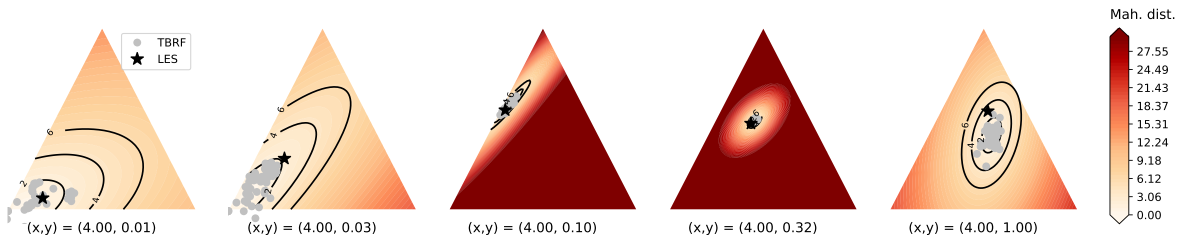
As can be seen, for almost all cases the LES results fall close to or within the $M = 2$ isolines. More variance is visible on locations where the mean of the different decision trees has more discrepancy with respect to the LES data points. Especially when looking at $(x, y) = (4.00, 0.32)$, predictions are very accurate in both directions of the barycentric map coordinates, and the predictions shows a close match to the LES data.



(a) x/h = 2



(b) x/h = 3



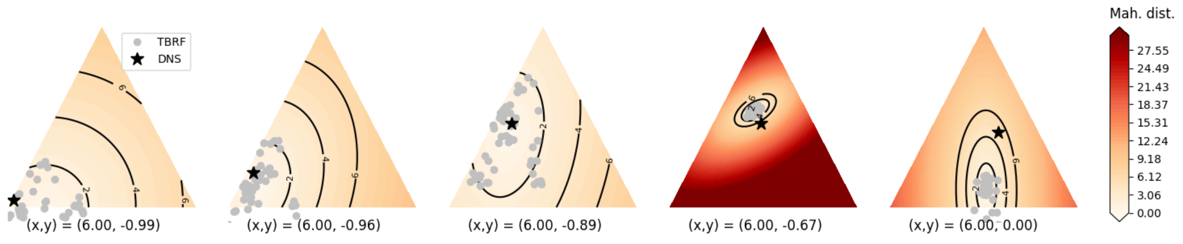
(c) x/h = 4

Figure 7.1: Uncertainty quantification on the barycentric map, curved backward facing step

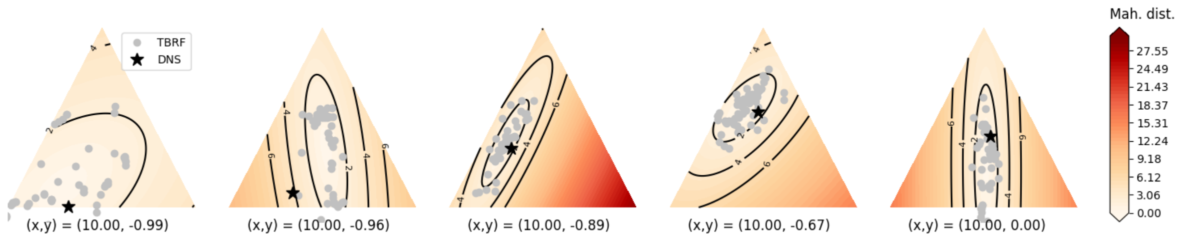
7.1.2 Backward Facing Step

Along three sections of the backward facing step, predictions of decisions trees in the random forest are plotted on the barycentric map using grey dots, see Figure 7.2. Locations according to the DNS data are plotted as well using a black asterisk. Again the Mahalanobis distance is used to give an indication on the spread of the different decision trees. The sections correspond to three of the sections shown earlier in Figure 6.9.

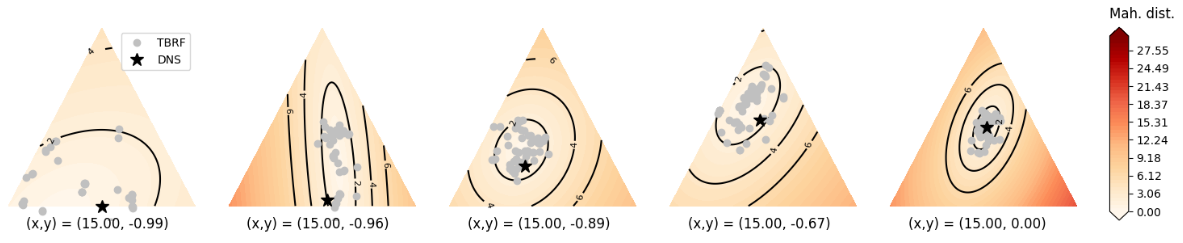
Just like for the curved backward facing step, it can be observed that in many cases the barycentric coordinates from DNS fall close to or within the $M = 2$ isolines. The two major discrepancies are seen for $(x, y) = (6.00, -0.67)$ and $(x, y) = (6.00, 0.00)$. Especially close to the wall higher uncertainties can be observed, and uncertainties are in general higher compared to the curved backward facing step, which can be explained due to the fact that the curved backward facing step is relatively similar to the periodic hills flow case which is included in the training data.



(a) x/h = 6



(b) x/h = 10



(c) x/h = 15

Figure 7.2: Uncertainty quantification on the barycentric map, backward facing step

7.2 Monte Carlo simulation

A Monte Carlo simulation was done to quantify uncertainty of the reattachment point. A random selection of decision trees was used to construct 80 different TBRF's, for which the predictions of b_{ij} were propagated using the continuation solver.

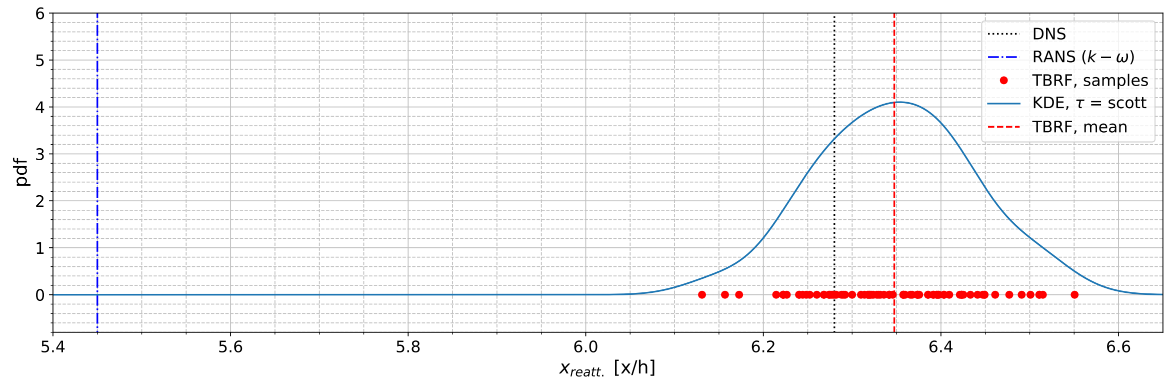
A choice has to be made on the amount of decision trees included in the random forest. Multiple decision trees need to be used, since a single decision tree shows too much variation and non-realizable values of the anisotropy tensor to propagate. Two simulations were done, one with 10 decision trees in each forest, and one with 30 decision trees. The amount of decision trees will affect the variance of the reattachment point, which will be lower when averaging over more trees. After propagating, the reattachment point was calculated from the wall shear stress. A kernel density estimate was then used to give an indication of the possible distribution of the reattachment point.

Figure 7.3 presents the Monte Carlo simulation for the reattachment point of the backward facing step flow case. Figure 7.3a presents the case for 10 TBDT's per TBRF, Figure 7.3b presents the case for 30 TBDT's per TBRF. The 80 samples from the different TBRF's are plotted using the red dots, the mean of all the TBRF's is indicated by the red dashed line, and the reference values from DNS and the $k-\omega$ RANS simulation are plotted with the black dotted line and blue dash-dot line respectively. Furthermore the kernel density estimate is plotted, where the bandwidth τ was estimated using Scott's rule.

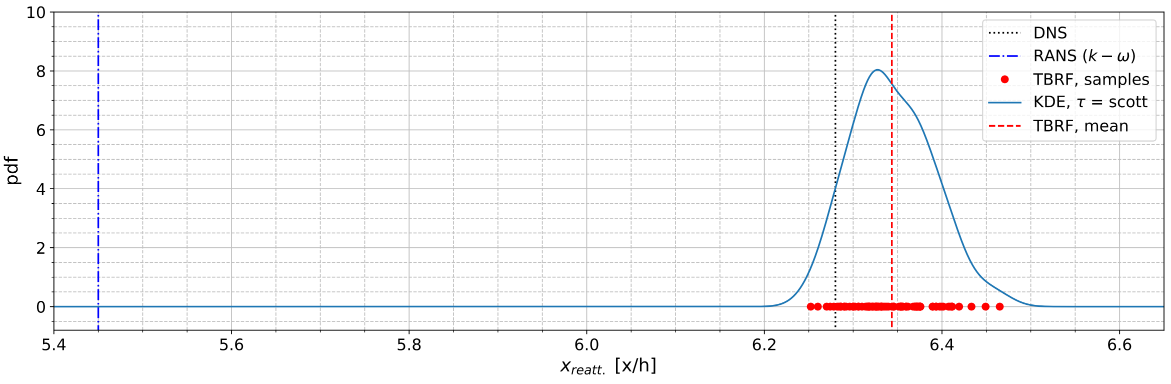
What can be seen, is that the variance decreases when averaging over more TBDT's, which is expected. Using the cumulative probability distribution from the KDE using Scott's rule, the DNS data point falls within the 87% confidence bound for 30 TBDT's, and within the 64% confidence bound for 10 TBDT's. All results from the different TBRF's result in a significantly improved reattachment point location compared to the RANS simulation.

The same Monte Carlo simulation can be used to quantify uncertainty of the propagated velocity field and skin friction coefficient. Results are presented for the case using 10 TBDT's per TBRF in Figure 7.4. Figure 7.4a presents the velocity profiles as given by the DNS data and the RANS simulation, along with predictions from the TBDT's. Since 80 different TBRF's were propagated, the two lowest and two highest samples for each location in the flow field for which the velocity profiles are plotted were discarded in order to obtain the 95% confidence bounds. As can be seen the variance of the different random forests can not explain a lot of the discrepancy present in the propagated velocity fields with respect to the DNS data. It seems that the velocity field is not overly sensitive to the variations in b_{ij} given by the random forests. Low variance of the propagated velocity profiles can be seen around x/h = 4 and x/h = 6 moving further away from the wall, while the discrepancy in velocity is relatively high at these locations. Higher variance of the propagated velocity profiles can be seen around x/h = 15 and x/h = 20 closer to the wall, which in this case somewhat matches the higher discrepancies with respect to the DNS data.

Figure 7.4b presents the Monte Carlo simulation for the skin friction coefficient. Again the 95% confidence bounds were obtained for each x/h location by discarding the two lowest and



(a) 10 trees per TBRF

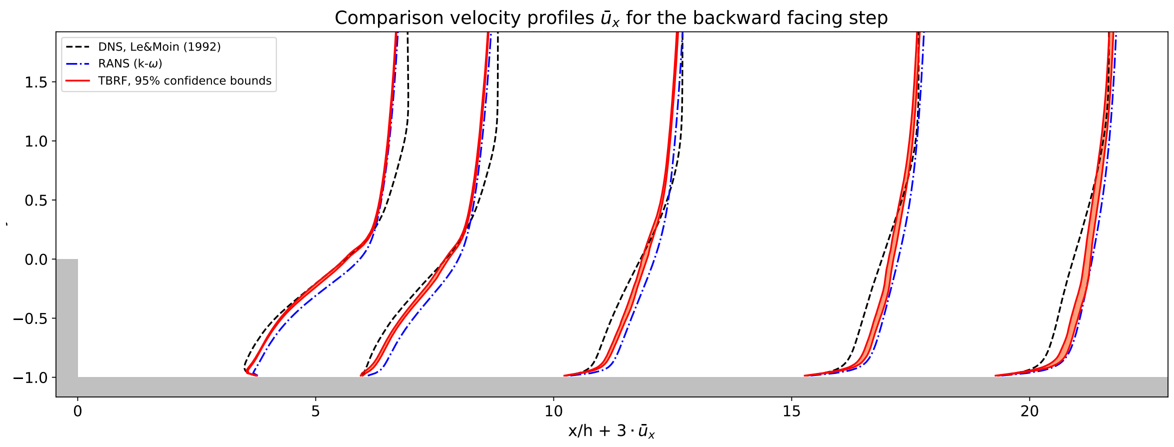


(b) 30 trees per TBRF

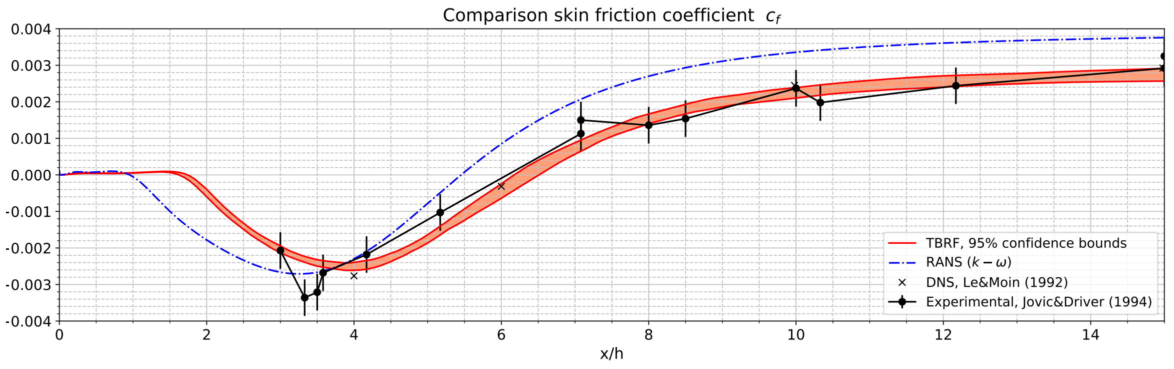
Figure 7.3: Monte Carlo simulation for the reattachment point of the backward facing step flow case

the two highest samples. In this case the confidence bounds capture the four samples from DNS at x/h = 4.0, x/h = 6.0, x/h = 10.0, and x/h = 15.0 quite well. Only the discrepancy with respect to the experimental data round x/h = 3 is not explained by the variance of the propagated velocity fields.

A possible explanation for the fact that the confidence bounds of the velocity profiles in Figure 7.4a show more discrepancy with respect to the DNS data compared to the skin friction coefficient in Figure 7.4b, would be that predictions close to the wall by the TBRF are more accurate. This would also explain the fact that the velocity profiles close to the wall are actually quite accurate when comparing to the DNS data, and are an improvement over the baseline RANS simulation. As was shown in Figure 6.9, predictions on the barycentric map by the TBRF became less accurate when moving away from the wall.



(a) Velocity profiles



(b) Skin friction coefficient

Figure 7.4: Monte Carlo simulation for the velocity profiles and skin friction coefficient

7.3 Confidence parameters

As shown in Section 7.1, it seems that the amount of uncertainty of a prediction from the TBRF correlates with the amount of error one can expect. Generally, the DNS/LES samples seem to fall within the cloud of points from the individual TBDT's. Given this observation, an analysis was done to see whether an increase in the variance of predictions from different TBDT's in the TBRF leads to a larger discrepancy in the predictions for b_{ij} on average. When this is true, the variance between the decision trees might be usable as an a posteriori confidence parameter.

The barycentric map was used to calculate the variance of the TBDT's. The average of the variance in the ξ-direction and the η-direction of the barycentric map was used as the independent variable. For the dependent variable the RMSE of b_{ij} as predicted by the TBRF was calculated compared to the DNS/LES data at the same mesh locations. This led to a cloud of points, which were binned using boxplots in order to give a clearer overview of the general trend.

Two different TBRF's were used. One was trained on PH10595, PH5600, and CD12600. This TBRF was tested on SD3500, and CBFS13700, see case number C4 and C1 respectively in Table 6.1. For the square duct reliable DNS data was available in the entire flow field, making it a suitable choice for this analysis. For the curved backward facing step only the region close to the wall ($y/h < 1.2$) was used, see Subsection 4.3. In order to also utilize the flow field for CD12600 for testing, which has reliable data in the entire flow domain, a TBRF was trained on PH10595 and PH5600, see case number C5 in Table 6.1.

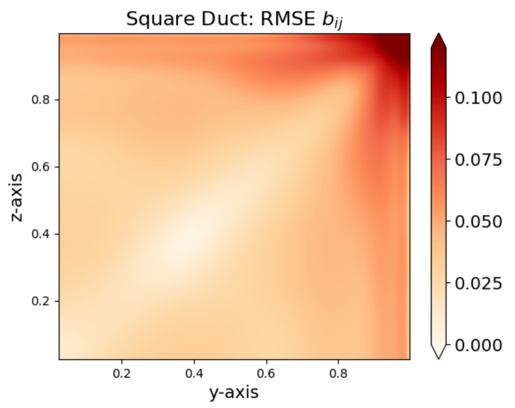
7.3.1 Square Duct

Figure 7.5 presents the RMSE of b_{ij} in the flow domain of the square duct, along with possible measures for prediction confidence. The TBRF has been trained on PH10595, PH5600, and CD12600, and 100 TBDT's are present in the TBRF.

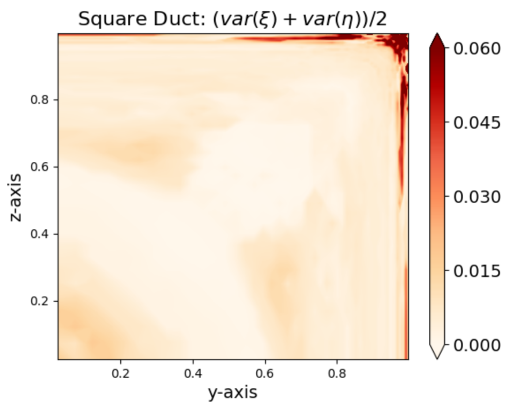
In Figure 7.5a a contour plot of the RMSE of b_{ij} is presented. As can be seen, most of the error is present in the corner of the duct. Relatively high error values are also observed close to the walls. Moving towards the center of the duct the RMSE decreases. In Figure 7.5b a similar pattern can be observed when plotting the variance on the barycentric map of the TBDT's in the TBRF. Most of the variance is present in the corner of the duct, and close to the walls. Variance along the diagonal closer to the centreline of the duct is much smaller, which is also observed for the RMSE of b_{ij}.

The a priori confidence parameters as proposed in Wu et al. (2016) are presented in the flow domain in Figure 7.5c (KDE distance), and in Figure 7.5d (Mahalanobis distance). Different bandwidth sizes τ for the kernels of the KDE were used, from which $\tau = 4$ yielded the clearest results and was therefore chosen. Both parameters predict the highest amount of extrapolation at the centreline of the duct, where predictions by the TBRF are actually very accurate. Closer to the walls the a priori confidence parameters seem to be more accurate: more extrapolation is predicted here, which matches with the higher RMSE of b_{ij} in Figure 7.5a.

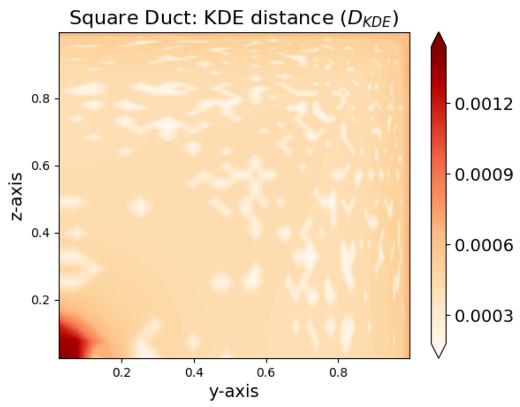
Boxplots of the variance of the TBDT's in the TBRF versus the RMSE of b_{ij} are presented in Figure 7.6. The data has been plotted on log-log scale, since the data has a skewness towards lower values of the average variance on the barycentric map. The boxplots represent the median (orange line in the middle), the upper and lower quartiles of the data (the upper and lower limits of the box), the 99% confidence intervals of the data (the whiskers), and outliers are plotted by means of the black circles. A clear trend can be observed, where on average the RMSE of b_{ij} increases when the variance of the TBDT's increases. As a comparison, the a priori confidence parameters $M_{norm.}$ and D_{KDE} are plotted against the RMSE of b_{ij} as well in Figure 7.7. It can be seen that both a priori confidence parameters show two separate upward trends. For the normalized Mahalanobis distance $M_{norm.}$ a split in the upward trend can be observed after $M_{norm.} \approx 0.63$ in Figure 7.7a. This split can be related to $M_{norm.}$ in the flow field presented in Figure 7.5d. In the centerline of the duct high extrapolation is



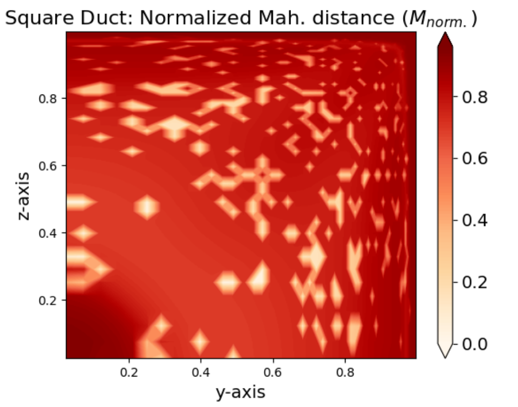
(a) RMSE b_{ij}



(b) $\frac{1}{2}\left(\mathrm{var}(\xi) + \mathrm{var}(\eta)\right)$



(c) D_{KDE}, $\tau = 4$



(d) $M_{norm.}$

Figure 7.5: Correlation study, square duct: RMSE b_{ij} along with variance of the TBDT's and a priori confidence parameters

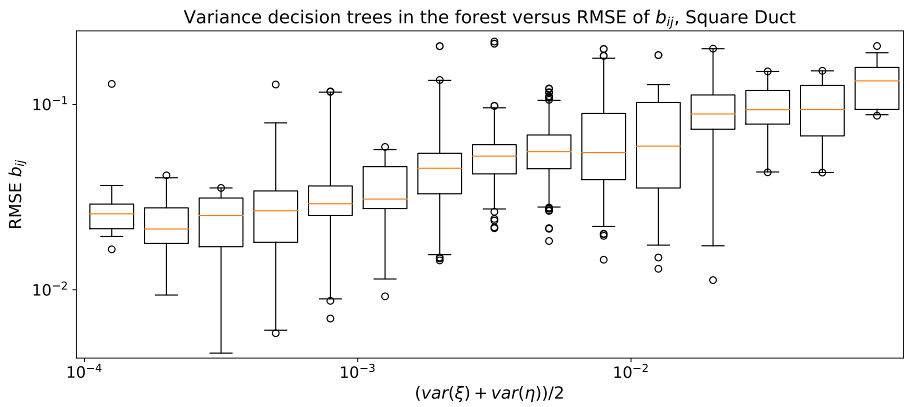
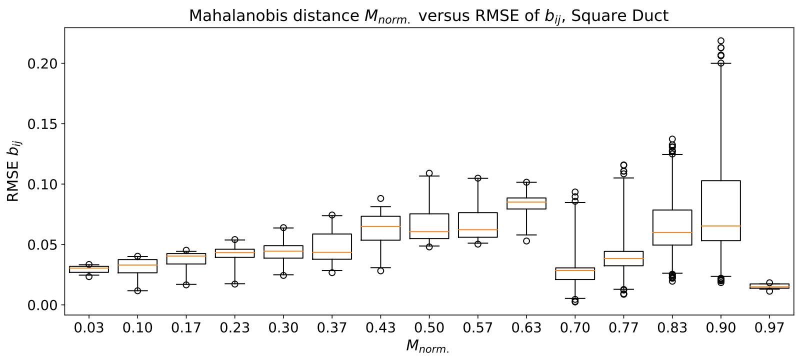
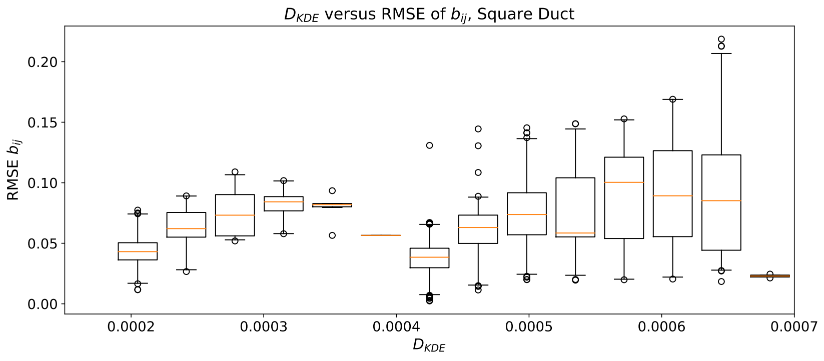


Figure 7.6: Correlation study, square duct: $\frac{1}{2}(\mathrm{var}(\xi) + \mathrm{var}(\eta))$ versus the RMSE of b_{ij}

given by the a priori confidence parameters, while the RMSE of b_{ij} is low at this location. This corresponds to the data in the boxplots ranging from $M_{norm.} \approx 0.63$ to $M_{norm.} = 1$. A similar split can be observed for D_{KDE} in Figure 7.7b.



(a) $M_{norm.}$



(b) D_{KDE}

Figure 7.7: Correlation study, square duct: boxplots of the a priori confidence parameters versus the RMSE of b_{ij}

7.3.2 Curved Backward Facing Step

Figure 7.5 presents the RMSE of b_{ij} in the flow domain of the curved backward facing step, along with possible measures for prediction confidence. Just like for the square duct, the TBRF has been trained on PH10595, PH5600, and CD12600, and 100 TBDT's are present in the TBRF.

The RMSE of b_{ij} in the flow domain, and the possible metrics for prediction confidence are presented in Figure 7.8. Most of the error is present close to the wall of the hill. Especially at the start of the curved part of the hill higher errors can be observed. This is the area where the shear layer starts to separate, and where 1-component turbulence greatly increases due to shear-generation, see Section 4.3. Two areas of relatively high error can be observed as well above the hill, between approximately x/h = -2 and x/h = 4.

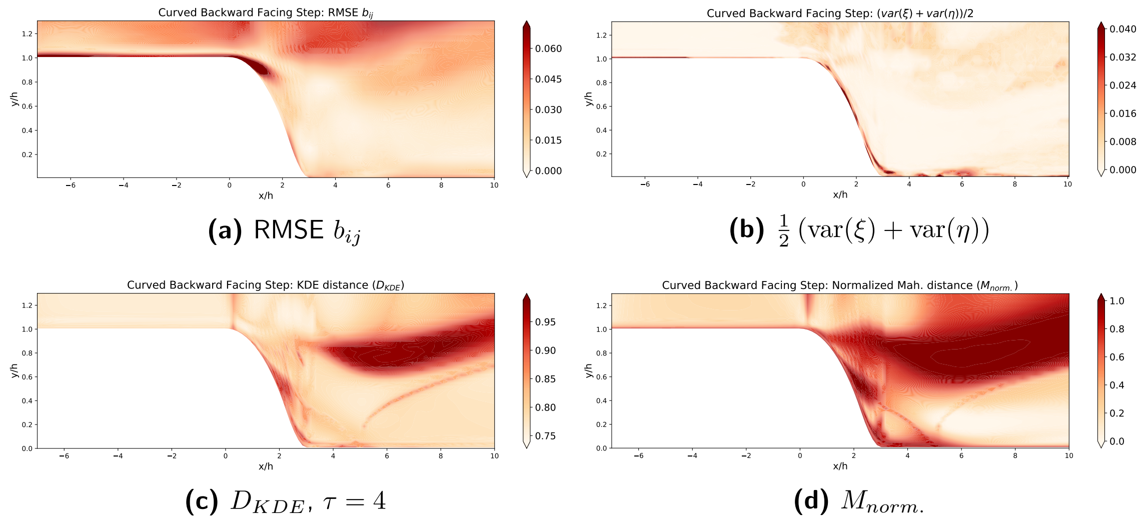
The variance on the barycentric map of the TBDT's in the TBRF is presented in Figure 7.8b. Most of the variance is present close to the walls, which matches the higher errors as given in Figure 7.8a. No extra variance is predicted at the shear layer separation location however. Some more variance is present above the hill, between approximately x/h = 0 and x/h = 4, which roughly matches the area with higher values for the RMSE of b_{ij}. Low levels of variance are present inside of the slowly moving recirculation region, and the region around $y/h = 0.4$ ranging from $x/h \approx 4$ to $x/h \approx 10$, which is in accordance to lower RMSE values of b_{ij} as well.

The a priori confidence parameters from Wu et al. (2016) show less correspondence to the RMSE of b_{ij}, as can be seen when looking at Figure 7.8c and Figure 7.8d. Both parameters give increased values for extrapolation close to the walls, but give most of the extrapolation in a large region around $y/h \approx 1.0$ after the curved part of the hill. This does not correspond with the RMSE of b_{ij} as presented in Figure 7.8a. Furthermore, higher extrapolation can be observed in the outer shear layer of the recirculation region, for which no increased error is present when looking at the RMSE.

Boxplots of the variance of the TBDT's versus the RMSE of b_{ij} are presented in Figure 7.9. In general, again an increase in RMSE of b_{ij} can be observed when predictions from the TBDT's have higher variance. Only for very low values of the variance this effect is less present. The a priori confidence parameters $M_{norm.}$ and D_{KDE} perform much worse in correlating with the RMSE of b_{ij}. Boxplots for these confidence parameters are presented in Appendix E. No upward trend can be identified for either of the parameters.

7.3.3 Converging-Diverging Channel

The RMSE of b_{ij} in the flow domain of the converging-diverging channel, as well as the possible measures for confidence, are plotted in Figure 7.10. In this case the TBRF algorithm was trained on PH10595 and PH5600. The RMSE of b_{ij} is given in Figure 7.10a. As can be seen, most of the error is present above the hill crest. Furthermore, higher levels of error can



(a) RMSE b_{ij}

(b) $\frac{1}{2}\left(\mathrm{var}(\xi) + \mathrm{var}(\eta)\right)$

(c) D_{KDE}, $\tau = 4$

(d) $M_{norm.}$

Figure 7.8: Correlation study, curved backward facing step: RMSE b_{ij} along with variance of the TBDT's and a priori confidence parameters

be observed at the lower walls, and on the wall of the hill ranging from $x \approx 4$ to $x \approx 6$. Just like for the curved backward facing step, some additional error can be observed at the shear layer separation location, where 1-component turbulence is produced by the increased shear stresses ($x \approx 6$).

Variance of the TBDT's in terms of the coordinates on the barycentric map is presented in Figure 7.10b. A lot of the variance is present on top of the hill, which broadly speaking matches the high RMSE of b_{ij} around this area. Higher areas of variance are also observed close to the wall, especially just before the hill crest, and at $x \approx 8$ where the channel geometry becomes flat again. A streak of higher variance is present in wake of the separation region, which seems to match a similar streak of high RMSE of b_{ij}. High variance is also present in the middle of the duct near the outlet, which is not present in the contour plot for the RMSE of b_{ij}. The large values for the RMSE of b_{ij} on top of the hill can largely be explained when looking at the extrapolation metrics in Figure 7.10c and Figure 7.10d. Apparently the features in this region are not similar to those present in the training data, and the large amount of extrapolation likely makes the TBRF algorithm less accurate. A streak of high extrapolation can be seen after the hill crest, similar to the streak present for the variance. Furthermore more extrapolation is predicted in the middle of the duct near the outlet, also similar to the high variance region in Figure 7.10b.

Boxplots of the variance on the barycentric map versus the RMSE of b_{ij} are presented in Figure 7.11 for the converging-diverging channel. A general trend can be observed just like for the previous flow cases, where on average the RMSE of b_{ij} increases when having more variance of the TBDT's. More spread can be observed however compared to the previous flow cases. The relation between the a priori confidence parameters ($M_{norm.}$, D_{KDE}) and the RMSE of b_{ij} is presented in Appendix E. No general trend can be identified for $M_{norm.}$. For the KDE distance there seems to be a trend, with higher values of D_{KDE} leading to more error in b_{ij} on average.

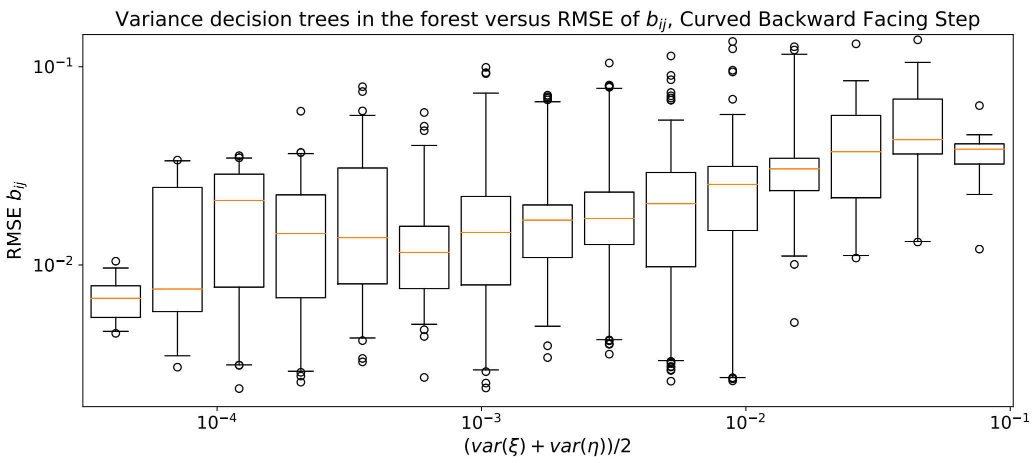
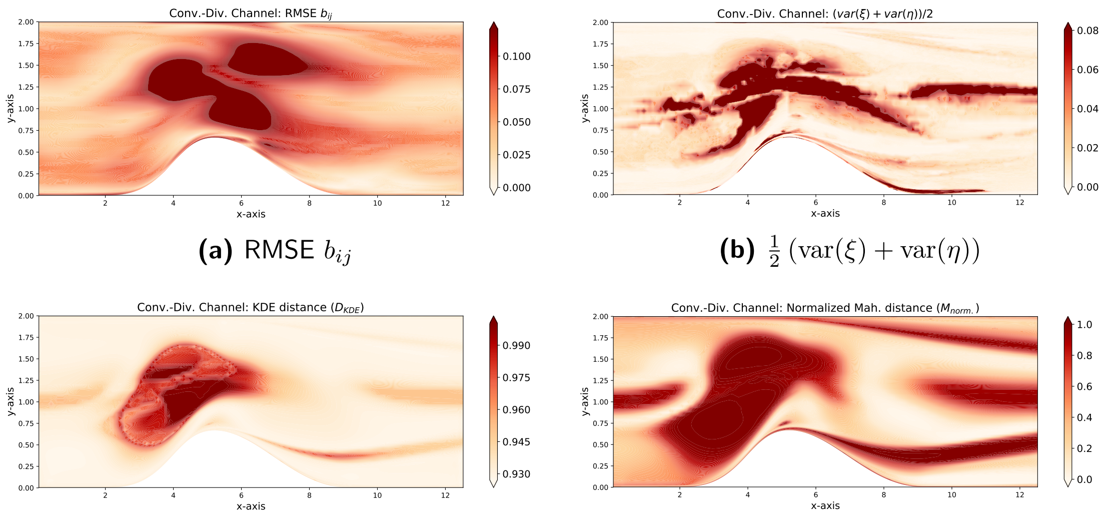


Figure 7.9: Correlation study, curved backward facing step: $\frac{1}{2}(\mathrm{var}(\xi) + \mathrm{var}(\eta))$ versus the RMSE of b_{ij}



(a) RMSE b_{ij}

(b) $\frac{1}{2}(\mathrm{var}(\xi) + \mathrm{var}(\eta))$

(c) D_{KDE}, $\tau = 8$

(d) $M_{norm.}$

Figure 7.10: Correlation study, converging-diverging channel: RMSE b_{ij} along with variance of the TBDT's and a priori confidence parameters

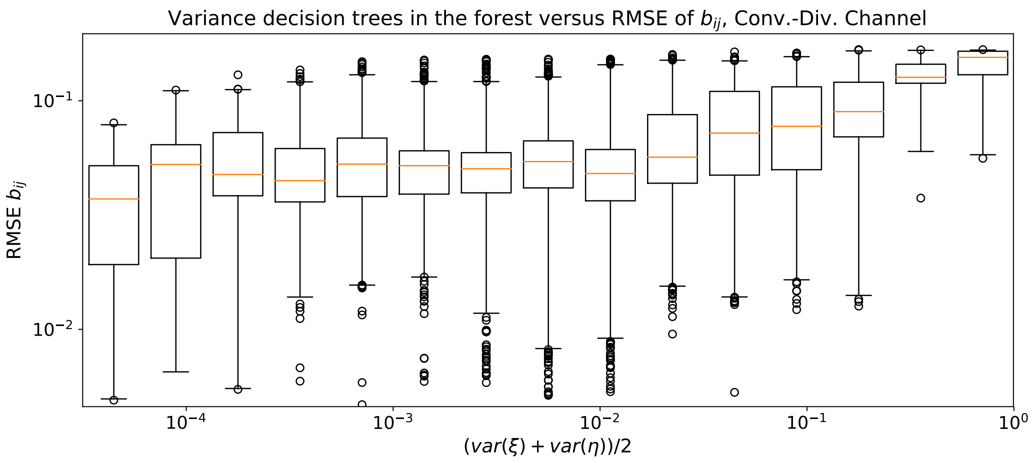


Figure 7.11: Correlation study, converging-diverging channel: $\frac{1}{2}\left(\mathrm{var}(\xi) + \mathrm{var}(\eta)\right)$ versus the RMSE of b_{ij}

Chapter 8

Interpretation of the TBRF results

In this chapter, a closer look at the results from the Tensor Basis Random Forest will be conducted. One of the claimed benefits by the authors of the random forest algorithm, is that it is relatively interpretable (Hastie et al., 2008). The random forest algorithm is able to give an indication of how important different features are in fitting the data. First of all, this information can potentially be used to identify features which are for example useful to create new turbulence models. Secondly, this information can be used for feature selection: for some machine learning algorithms it is important that only relevant features are used in order to obtain good performance, see for example the support machines in Ling and Templeton (2015). Results for the feature importance analysis will be presented in Section 8.1. Next to studying the feature importance, the TBRF algorithm might give some insight in the tensor basis as introduced by Pope (1975), by looking for example at locations where certain tensor basis coefficients play a large role. Results will be presented in Section 8.1.

8.1 Feature Importance

An idea of importance of the features used for training the TBRF algorithm can be obtained. Each time a certain feature is used for creating a split in a TBDT, the MSE between the response and the fit of the data reduces. The total contribution of all features to the reduction in MSE of b_{ij} can be tracked for all the decision trees. Finally the mean of the total contribution for each feature can be calculated over all the trees to get an idea of the importance of different features in the TBRF. This results in the squared relative importance of a feature, see Hastie et al. (2008).

The expression for the squared relative importance $\mathcal{I}^2$ of the j-th feature is given by

$$\mathcal{I}_j^2 = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{T} \left[\mathrm{MSE}_{R_L+R_R} - \frac{1}{n_{s,R_L} + n_{s,R_R}} \left(\mathrm{MSE}_{R_L} \cdot n_{s,R_L} + \mathrm{MSE}_{R_R} \cdot n_{s,R_R} \right) \right],$$

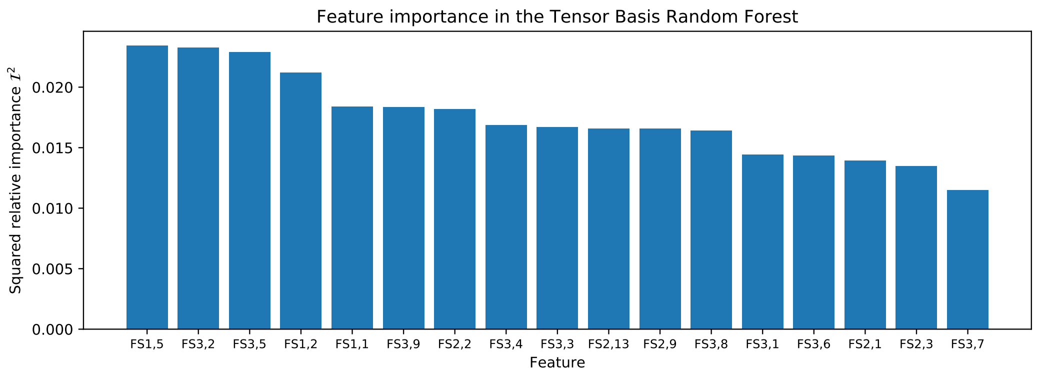


Figure 8.1: Feature importance: TBRF trained on PH10595, PH5600, and CD12600

(8.1)

where M is the total amount of trees, T are all the internal nodes where feature j was selected as the splitting feature, R_R and R_L are the right and left bin in which the data is split at the given node, and n_s are the amount of samples present in a given bin.

The result for the squared relative importance of different features when training on PH10595, PH5600, and CD12600 (corresponding to case numbers C1, C2, and C4 in Table 6.1) is presented in Figure 8.1. The feature set, and the feature number from the set is indicated on the x-axis, see Table 3.1 for the exact definitions. The values of the features in the flow domain can be found in Appendix F. As can be seen, the features from feature set 1 are relatively useful in reducing the MSE and fitting the response. The feature with the highest squared relative importance is FS1,5 (trace($\hat{\mathbf{\Omega}}^2\hat{\mathbf{S}}^2$)), the fourth highest feature is FS1,2 (trace($\hat{\mathbf{S}}^3$)), and the fifth highest feature is FS1,1 (trace($\hat{\mathbf{S}}^2$)). Two features from feature set 3 can be seen in the top five: FS3,2 (turbulence intensity) and FS3,5 (ratio of turbulent time scale to mean strain time scale).

A comparison can be made to the case when training the TBRF on the periodic hills only. Figure 8.2 presents the squared relative importance for a TBRF trained on PH10595 and PH5600 corresponding to case number C5 in Table 6.1. While being similar to the results in 8.1, some small differences can be observed. First of all, feature FS3,9 (non-orthogonality between velocity and its gradient) plays relatively speaking a larger role. The rest of the top five most important features are largely speaking the same, with again feature FS1,5 being the most important. The least important features are also quite similar.

More differences can be expected when training on a entirely different flow case. Figure 8.3 presents the squared relative importance when training a TBRF on the square duct flow case, with four different Reynolds numbers ranging from $Re = 2,400$ to $Re = 3,200$. As can be seen, the squared relative importance of the different features is quite different from the previous cases. Features which are still important are those from feature set 1, and FS3,9. Furthermore, FS3,3 is seen as important, which is the wall-distance based Reynolds number. Looking at the features in the flow domain, see Figure F.6 in Appendix F, this makes sense, since many of the other features show similar values close to the wall and more towards the

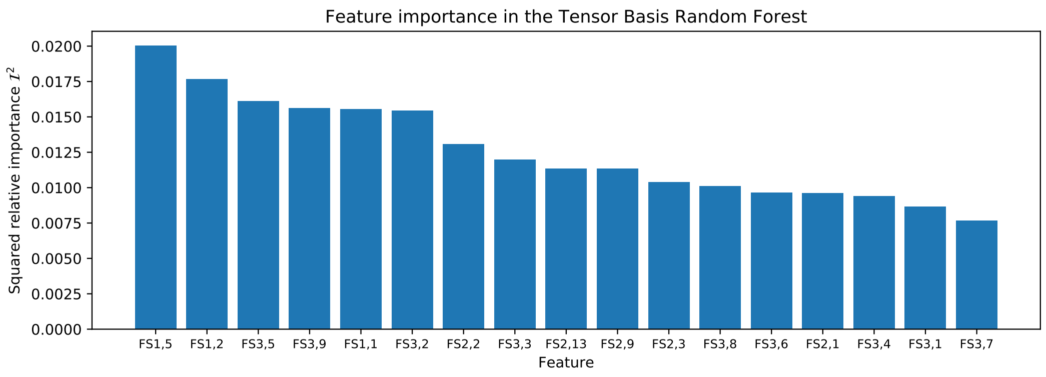


Figure 8.2: Feature importance: TBRF trained on PH10595 and PH5600

centreline of the duct, making the wall-distance based Reynolds number a good discriminator. For the other flow cases the boundary layer is thinner than for the square duct, making this feature less useful for distinguishing between parts closer to the wall and further away from the wall. The cap of the wall-distance based Reynolds number was set to a value of 2, as specified in Ling and Templeton (2015). For further research it might be advantageous to increase this value, in order to make this feature more useful for the higher Reynolds number flow cases as well.

What can be seen when looking at Figure 8.3, is that six features are barely useful at all. These include FS3,4; FS3,6; FS3,7; FS2,2; FS3,8; and FS3,1. One cause of this, is the fact that the pressure is ill-defined in the square duct. This means that all features based on the pressure gradient can not be used, which are features FS3,4 and FS3,6. For the rest of these features it can be seen that the variance is very minimal, making them unsuitable to use for training.

The fact that the importance when using a training set with the periodic hills and converging diverging channel is so different when using a training set with only the square duct is interesting. In Section 6.2.3 it was shown that predictions for case number C4 (see Table 6.1) are actually quite good. This means that the TBRF algorithm is able to deal well with outliers, since in this case part of the features used for making predictions in the square duct are ill defined.

The ability of the TBRF algorithm to evaluate the importance of different features has some interesting use cases. This information can first of all be used for feature selection. In for example Ling and Templeton (2015), information about the importance of different features was used to select the best ones for training a support vector machine algorithm (which is much more sensitive to 'bad' features compared to the random forest). In this case the AdaBoost decision tree algorithm was used to produce information about the feature importance, which can produce the importance metric in a similar fashion to the random forest.

The information about which features are the most important is also valuable for the user of the algorithm. As an example, it was shown in Section 6.2.4 that the high errors in the flow domain corresponded to a high amount of extrapolation. The high amount of extrapolation

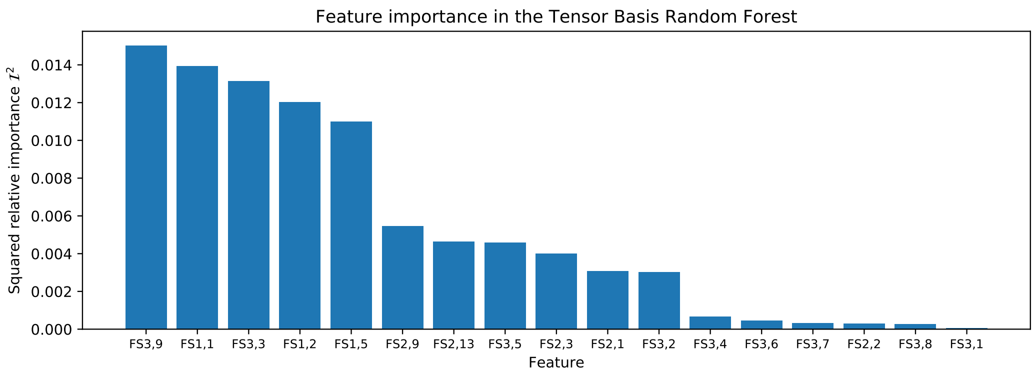


Figure 8.3: Feature importance: TBRF trained on SD3200, SD2900, SD2600, and SD2400

corresponded to an area in the flow where the value of feature FS1,1 was much higher than encountered in the training data. From the feature importance output of the algorithm (Figure 8.2), one could have seen that this is one of the most important features, which means that it is more likely the algorithm will produce poor results in this case. When it would have been one of the less important features causing a high amount of extrapolation, its influence would likely have been less severe.

One idea from these findings would be adapting the a priori confidence parameters by scaling the feature space with the corresponding importance values. This way, the amount of extrapolation would be increased when extrapolating with respect to an important feature, and the amount of extrapolation would be decreased when the the given feature would not be so influential. This way a better confidence parameter could be obtained for the predictions.

Other methods exist for defining the importance of features. One such method would be randomly perturbing a given variable (Hastie et al., 2008), and letting the algorithm make predictions. The decrease in accuracy with respect to the unperturbed input feature set can then be used as a measure of the given feature importance. One possibility which might be interesting to investigate, would be looking at the flow domain and analysing which features are important in different areas of the domain. As an example, perhaps it might be possible to relate a high level of anisotropy at a specific area in the flow to a flow phenomena described by a certain input feature. This way it might be possible to use the machine learning algorithm to learn more about turbulence in a flow case.

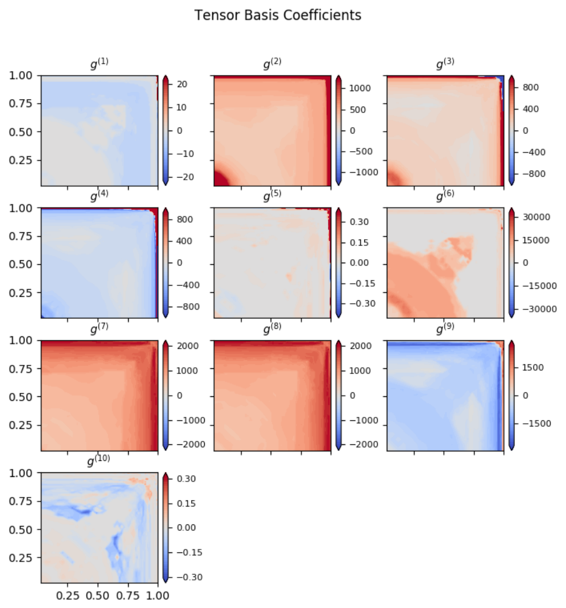
8.2 Tensor Basis Coefficients

It could be useful looking at the tensor basis coefficients and the basis tensors themselves, in order to obtain an idea why certain terms of the tensor basis are important at certain locations in the flow. For example, this can give an idea which of the tensors are really essential to represent the Reynolds stress anisotropy tensor when creating a new turbulence model based on the idea from Pope (1975). Results for the square duct and the curved backward facing step will be analysed further in this section.

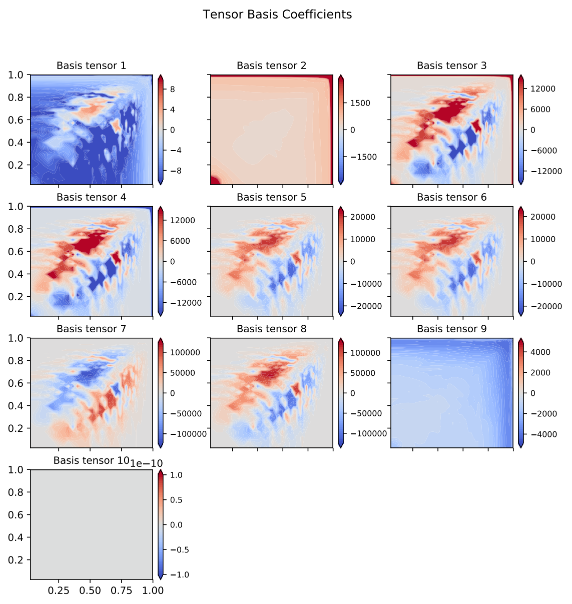
8.2.1 Square Duct

Figure 8.4 presents the tensor basis coefficients $g^{(m)}$ in the flow domain of the square duct, for $Re = 3,500$. Figure 8.4a corresponds to data-set C4 in Table 6.1. Figure 8.4b was trained using four square duct flow cases, ranging from $Re = 2,400$ to $Re = 3,200$, which will be denoted by case number C6. What is interesting to see, is that completely different values for $g^{(m)}$ can arrive at relatively similar expressions for b_{ij} (for b_{ij} corresponding to the coefficients in Figure 8.4b, see Figure G.1). Only the second tensor basis coefficient shows good correspondence between both cases. For the case using case C6, a lot of noise can be observed, which is approximately symmetric with opposing sign around the diagonal. A likely explanation for the fact that so much noise is present for this case, is that the training data was too similar, resulting in overfitting of the data, creating a function which will not generalize well when going to other flow cases. The coefficients using case C4 look a lot smoother and regular.

What can be seen when observing the results for case C4, is that that the tensor basis coefficients $g^{(m)}$ reach high values close to the wall. This is likely due to the scaling of the mean strain rate tensor and mean rotation rate tensor with k/ϵ, see (2.23). Since the $k - \omega$ turbulence model has been used, the tensors have been scaled with the equivalent time scale $1/(C_\mu \omega)$. As ω increases close to the wall, this means the magnitude of the basis tensors decreases, which in turn means the tensor basis coefficients must increase to have the same effect closer to the wall. What can also be observed, is that $g^{(5)}$ and $g^{(10)}$ have very low values overall.



(a) Case C4



(b) Case C6

Figure 8.4: Tensor basis coefficients $g^{(m)}$ for the square duct flow case

Since the different basis tensors have different magnitudes, it was decided to analyse the combination of the tensor basis coefficients together with the norm of the tensor (see (3.20)), to get a better impression of which tensors play an important role, and where. Results are presented in Figure 8.5 for case C4.

It can be observed that the fifth and tenth terms in the tensor basis do not play a role. By looking at the basis tensors $\mathbf{T}^{(m)}$, which are presented in Appendix G (Figure G.2 and G.3), some observations can be made why given tensors are important at certain locations. Looking at Figure 8.5, the sixth term in the tensor basis seems to play an important role, as its magnitude is the largest overall. This can be explained due to the fact that the only basis tensors which are able to model the shear stress components (using the $T_{12}^{(m)}$ and $T_{13}^{(m)}$ entries) are $\mathbf{T}^{(6)}$ in combination with $\mathbf{T}^{(1)}$. In fact the $T_{12}^{(m)}$ and $T_{13}^{(m)}$ entries are already quite similar to the shape of b_{12} and b_{13} (Compare Figure 4.14 to Figure G.2a and G.2f).

Looking at $\mathbf{T}^{(1)}$ and $\mathbf{T}^{(6)}$ in Figure 8.5, it can be seen they have an inverted shape with respect to each other, with opposite signs. It might be possible that the algorithm prefers to use $\mathbf{T}^{(6)}$ in the lower left corner, since $\mathbf{T}^{(1)}$ has a larger magnitude at this location for the $T_{22}^{(1)}$ and $T_{33}^{(1)}$ components, which is better avoided to not introduce any effects on b_{22} and b_{33}. In the upper right corner of the domain $T_{22}^{(1)}$ and $T_{33}^{(1)}$ are almost zero, making $\mathbf{T}^{(1)}$ preferable in this region since it can purely be used to model the shear components of b_{ij}, while not introducing any additional effects on the normal stresses which would need to be cancelled out.

What can be observed is that $\mathbf{T}^{(2)}$, $\mathbf{T}^{(3)}$, $\mathbf{T}^{(7)}$, and $\mathbf{T}^{(8)}$ all play a similar role, mainly being responsible for modelling b_{11}. The $\mathbf{T}^{(3)}$ basis tensor also contributes in modelling b_{22} and b_{33}, together with the $\mathbf{T}^{(4)}$ and $\mathbf{T}^{(9)}$ basis tensors. Overall, looking at the role of the different tensors in modelling b_{ij} and their magnitude in Figure 8.5, it seems like basis tensor $\mathbf{T}^{(2)}$, $\mathbf{T}^{(3)}$, $\mathbf{T}^{(4)}$, and $\mathbf{T}^{(6)}$ are the most essential to include in the tensor basis.

8.2.2 Curved Backward Facing Step

In Figure 8.6 the norm of the tensor times the corresponding coefficient is plotted for the curved backward facing step (case C1). For the sake of brevity the coefficients $g^{(m)}$ themselves are left out, but can be found in Appendix G instead (Figure G.6).

Similarly to the square duct, it can be seen that the sixth basis tensor, $\mathbf{T}^{(6)}$ plays quite a large role. This can be explained when looking at the basis tensors, which are presented in Appendix G, due to the fact that it is largely responsible for modelling the shear stresses, together with $\mathbf{T}^{(1)}$. This matches with the result for the square duct, where $\mathbf{T}^{(6)}$ was responsible for modelling b_{12} and b_{13} together with $\mathbf{T}^{(1)}$. The basis tensors $\mathbf{T}^{(2)}$, $\mathbf{T}^{(3)}$, $\mathbf{T}^{(7)}$, and $\mathbf{T}^{(8)}$ all play a role in modelling the b_{11} component. Furthermore $\mathbf{T}^{(3)}$ is one of the few basis tensors which is able to represent the b_{22} component, leading to a negative value closer to the wall for $g^{(3)}$. The $\mathbf{T}^{(4)}$ and $\mathbf{T}^{(9)}$ basis tensors are the only ones which are able to model the b_{33} component, resulting in the coefficient field corresponding to these basis tensors looking similar to b_{33} (see

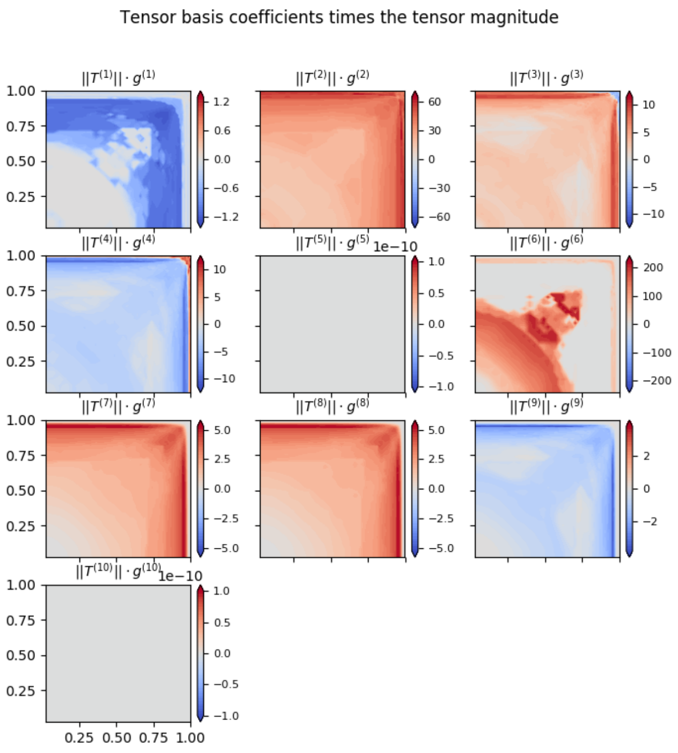


Figure 8.5: Tensor basis coefficients $g^{(m)}$ multiplied with the tensor norm $\left|\left|T_{ij}^{(m)}\right|\right|$ (case C4)

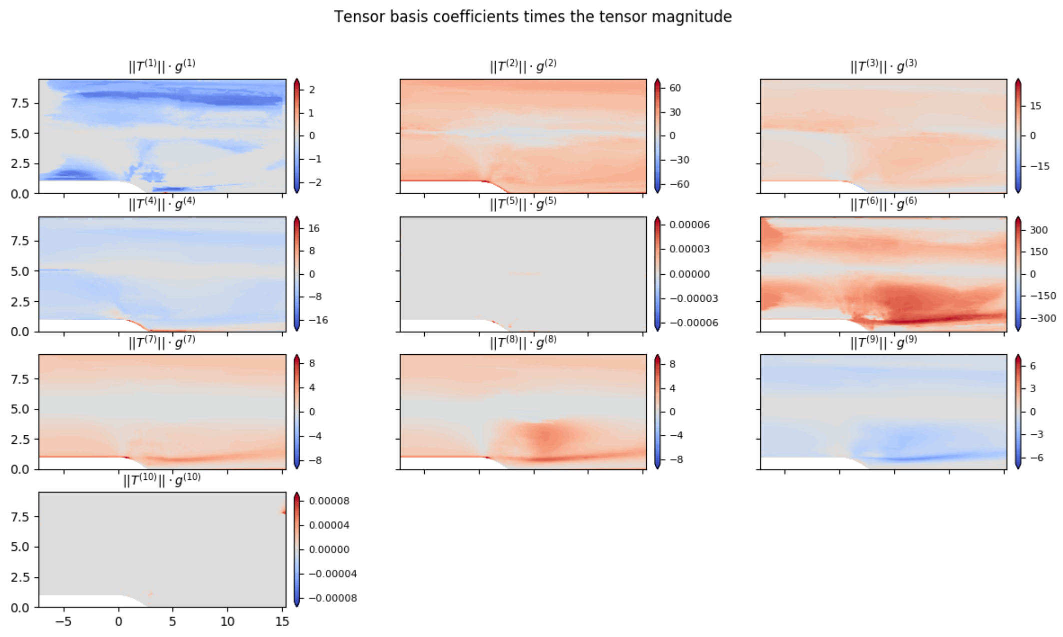


Figure 8.6: Tensor basis coefficients $g^{(m)}$ multiplied with the tensor norm $\left|\left|T_{ij}^{(m)}\right|\right|$ (case C1)

Figure 4.9). Just like for the square duct, basis tensors $\mathbf{T}^{(2)}$, $\mathbf{T}^{(3)}$, $\mathbf{T}^{(4)}$, and $\mathbf{T}^{(6)}$ seem to play an essential role in modelling b_{ij}, taking their role and magnitude in Figure 8.6 in account.

It is questionable however whether b_{ij} could not just be modelled with the first four tensors instead, since $\mathbf{T}^{(1)}$ and $\mathbf{T}^{(6)}$ are quite similar. It is possible that the algorithm prefers to use $\mathbf{T}^{(6)}$ instead of $\mathbf{T}^{(1)}$, since for the higher order tensor the normal component magnitudes are smaller with respect to the shear component magnitudes, compare Figure G.4a to Figure G.4f ($T_{11}^{(1)}$ and $T_{12}^{(1)}$ are of the same order of magnitude, $T_{11}^{(6)}$ is smaller than $T_{12}^{(6)}$). This means it becomes easier to use $\mathbf{T}^{(6)}$ purely for modelling the shear components, since less action is required to counteract the influence of the tensor on the normal components, for which $\mathbf{T}^{(1)}$ and $\mathbf{T}^{(6)}$ are no so suitable.

Chapter 9

Conclusions and Recommendations

A novel machine learning algorithm based on the random forest was presented in this thesis, called the Tensor Basis Random Forest (TBRF), which is able to predict the normalized anisotropy tensor. The algorithm was trained on several data-sets from the periodic hills, converging-diverging channel, and the square duct. Validation of the algorithm was done by testing it on test data-sets not present in the training data. In Section 9.1 conclusions will be given regarding the work presented in this thesis. In Section 9.2 recommendations are given for possible improvements and further extensions regarding the TBRF algorithm, and machine learning applied to turbulence modelling in general.

9.1 Conclusions

The main conclusions from this thesis will be given by making use of the three main research questions.

Q1: What limitations do the machine learning based approaches for data-driven RANS modelling such as presented in Ling et al. (2016b) and Ling et al. (2017b) have?

Results from Ling et al. (2016b) were reproduced, by using the Tensor Basis Neural Network (TBNN) for predicting the anisotropy tensor b_{ij} in different flow cases. The implementation of the algorithm using `python` and the `lasagna` machine learning library was obtained from Ling et al. (2017a). The algorithm was trained on a given data-set which contained multiple flow fields from different geometries, after which it was tested on a set of unseen flow fields. Results showed a good improvement over the baseline RANS simulation when validating results with DNS. One limitation of data-driven machine learning algorithms such as these, are their relatively inaccurate predictions in the case of extrapolation. This was for example observed when training machine learning algorithms on the periodic hills only, and making predictions on the converging-diverging channel, see Section 6.2.4. Some a priori confidence parameters have been introduced to measure this amount of extrapolation, based on the Mahalanobis distance (Ling et al., 2016b), or a distance metric based on the Kernel Density Estimate (Wu et al., 2016), which were observed to correlate with the error of the predictions in Wu et al. (2016). Another limitation of the artificial neural network are its large amount of hyperparameters which need to be tuned, and its relatively complicated training process.

Q2: Can the machine learning based approach presented in Ling et al. (2016b) be extended by making use of random forests?

The tensor basis random forest (TBRF) algorithm was introduced in this thesis, as a possible improvement on the tensor basis machine learning method from Ling et al. (2016b). Benefits of the random forest are its efficient way of dividing the data into a training and validation data-set, having few hyperparameters to tune, being relatively straightforward to implement, and the possibility to be easily trained in parallel. The tensor basis from Pope (1975) which serves as a basis for b_{ij}, was implemented by including linear least-squares regression in the decision tree nodes. Instead of each decision tree node having a constant approximation of the response (b_{ij}), this led to each node of the decision tree having a constant approximation of the tensor basis coefficients. Results showed great potential for the implemented algorithm. Results of the TBRF algorithm were in general on-par or more accurate in comparison with the TBNN algorithm. Predictions by the TBRF are Galilean invariant just like for the TBNN algorithm, which means training and testing can be done using different flow cases with different frames of reference. Predictions by the standard random forest do not have this property, making it not suitable for predicting tensor quantities such as b_{ij}, see Section 6.2.5.

Q3: Can these machine learning based approaches be improved by including the amount of uncertainty of the prediction?

Since the TBRF consists of a large number of decision trees, this makes it possible to quantify uncertainty of the output in a natural way. A Monte Carlo simulation on the reattachment point and skin friction coefficient of the backward facing step showed a good match with the DNS data. Furthermore it was demonstrated that the variance of the decision trees could function as an a posteriori confidence parameter of the predictions, as it showed more clearly a correlation with the RMSE of b_{ij} compared to the Mahalanobis distance and KDE-based distance.

9.2 Recommendations for future work

Recommendations for future work will be divided into three categories. First of all, recommendations will be given which might increase accuracy of the results presented in this thesis. Afterwards, recommendations for further extensions and explorations of the TBRF algorithm will be given for related topics. Finally, thoughts on machine learning applied to turbulence modelling in general will be given.

9.2.1 Possible improvements

Some recommendations can be made with respect to possible improvements of the implemented TBRF framework.

The most important step is to gather more reliable data on which the machine learning algorithms can be trained. As could be seen, the TBRF algorithm performs well when giving it enough data. When training on two different Reynolds numbers for the periodic hills, and one for the converging-diverging channel, the algorithm was able to make good predictions for the curved backward facing step, backward facing step, and square duct. However, when training it on the periodic hills only, it was observed that predictions on the converging-diverging channel were quite inaccurate. More reliable data, preferably also for higher Reynolds numbers, means that attempts can be made for predicting b_{ij} for more complex flow cases, where it can also be further investigated how reliable the continuation solver is. Developements for solutions regarding propagation of the anisotropy tensor should be kept track of as this is still a topic of current research, see e.g. Wu et al. (2017b).

It should be investigated how much the choice of turbulence model affects the accuracy of the predictions. It was decided to choose the $k-\omega$ turbulence model for the implementation, since it showed better convergence for the different flow cases, and as the propagation method (the continuation solver) used this turbulence model as its baseline. It was discovered however that the tensor basis coefficients $g^{(m)}$ become very large close to the wall, due to the scaling of the basis tensors with the time scale $1/(C_\mu \omega)$. Perhaps training the algorithm on the flow field given by the low Reynolds number Launder-Sharma $k-\epsilon$ model could provide improved predictions. This low Reynolds number turbulence model uses a modified turbulence dissipation rate ϵ, which goes to zero towards the wall. Setting both ϵ and k to a very low value at the wall (e.g. $1{\times}10^{-15}$) would mean the time scale k/ϵ would go towards unity at the wall.

It is recommended to further investigate the possibility of using the TBRF for uncertainty quantification. The use of random forests for uncertainty quantification is relatively new. Some important papers regarding this subject are Meinshausen (2006) and Mentch and Hooker (2016), which are recommended for further investigations for creating approximately unbiased random forest regressors with meaningful confidence intervals. It is clear that different hyperparameters for the TBRF algorithm will result in different variances between the individual trees, and thus in a different interval in which the predictions from the individual trees will lie. One possibility of addressing this issue and creating meaningful confidence intervals, would be an hyperparameter calibration similar to that presented in Tracey et al. (2013). Here, the kernel width of the stochastic kernel regression algorithm was varied, and optimized using k-fold cross validation to maximize the log-likelihood of the algorithm output with respect to the reference data.

Since the TBRF algorithm consists of a lot of TBDT's, a smart way of combining these predictions in the spatial domain to obtain a smooth field for b_{ij} can be thought of. It might be possible to use something related to a Gaussian process regression algorithm to interpolate the predictions for b_{ij} in the flow domain. This algorithm, see Section 2.5.4, is able to take in account noise of the predictions (e.g. the range of outputs from the different TBDT's), and could therefore be used to yield a smooth field for b_{ij}. It was attempted to implement the reconstruction of b_{ij} using the `GaussianProcessRegression` function in the `scikit-learn` library. It proved to be prohibitively expensive however to do this for larger meshes, as in this case the very large covariance matrices are not able to be stored in the memory of the computer. For the square duct, which has a relatively small mesh of 50×50 cells it was still possible. Results for the raw anisotropy tensor without any filtering ($b_{ij,raw}$) are presented in the upper row of Figure 9.1, the reconstructed fields using the Gaussian Process regression algorithm ($b_{ij,GP}$) are presented in the lower row. These results correspond to case number C4 in Table 6.1. Further investigations into smart filtering of the predicted b_{ij} using the method described above, or by somehow taking in account spatial correlations in the flow domain might be worthwhile.

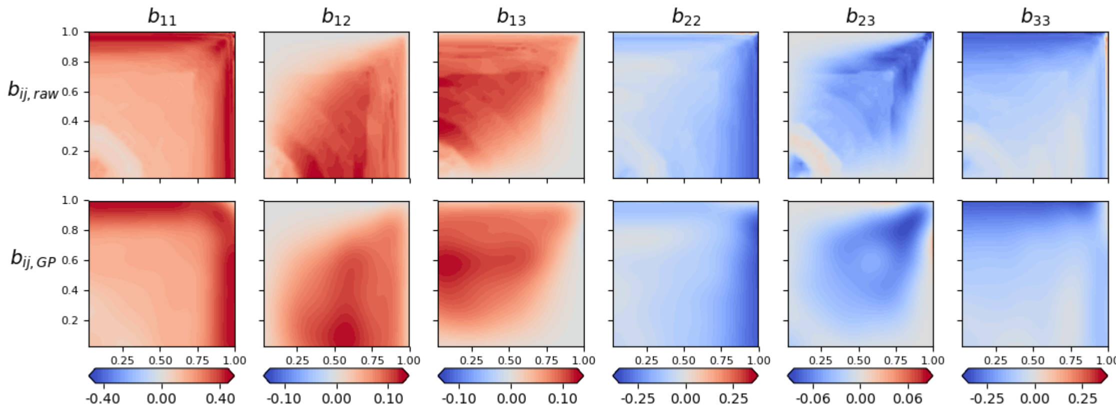


Figure 9.1: Raw values for b_{ij}, along with the reconstructed field using Gaussian process regression, square duct flow

9.2.2 Further extensions and explorations

The use of a tensor basis in machine learning opens up new interesting possibilities in the field of turbulence modelling. First of all, it would be interesting to see whether the TBRF algorithm can be used in an iterative sense, see the discussion in the end of Section 3.1.1, where it would relate the mean strain rate tensor and mean rotation rate tensor to the anisotropy tensor, thus replacing the original turbulence model. In order to be successful, predictions from the algorithm need to be smooth enough in order to attain convergence of the flow solver, for which the regularization term can be used in combination with the median filter.

Secondly, it would be interesting to see whether the TBRF algorithm can be used within a turbulence model. One example would be using it for modelling the pressure-rate-of-strain tensor $\mathcal{R}_{ij}$, see (2.26). For this term the Cayley-Hamilton theorem has been used to write it in terms of basis tensors created from b_{ij}, S_{ij}, and Ω_{ij}, see e.g. Speziale et al. (1991). The coefficients for the linear terms as used in Speziale et al. (1991) could be learned by the machine learning algorithm, or an extension could be made including more non-linear tensors in the expression for $\mathcal{R}_{ij}$.

Another possibility would be using these tensor basis machine learning approaches in a Large Eddy Simulation. The subgrid-scale stresses are often modelled similarly to the Reynolds stress. In for example the Smagorinsky model, a linear relation is assumed between the strain rate tensor and the subgrid-scale stress. Including the non-linear basis tensors from Pope (1975) for modelling the subgrid-scale stress is for example proposed in Lund and Novikov (1992), which could be a good starting point for further research regarding this possible application of the TBRF algorithm. Using machine learning algorithms for a hybrid approach combining RANS and LES could be interesting as well, as discussed in for example Durbin (2018). In both LES and RANS there is an unclosed stress τ_{ij}. Depending on how well the grid is refined at a certain location, the unclosed stress can be blended using

$$\tau_{ij} = (1-\gamma)R_{ij} + \gamma\,\tau_{ij}^{SGS}, \tag{9.1}$$

where τ_{ij}^{SGS} is the subgrid-scale stress from the LES model. The blending parameter γ depends on the ratio of the grid dimension to the turbulence scale, ranging from 0 (at locations with a very coarse mesh relative to the turbulence scales) to 1 (at locations where the mesh is fine enough to resolve all turbulence). At least looking at results for the curved backward facing step and backward facing step, the TBRF algorithm seems to be able to accurately model the Reynolds stress close to the walls, where the turbulence scales will be the smallest. This might make the use of the TBRF algorithm in such a hybrid approach useful.

The TBRF algorithm has to possibility to give a measure of the importance of given features, based on their contribution in performance increase, see (8.1). It would be interesting to identify parts in a certain flow for which it is known what kind of turbulent phenomena induce anisotropy of the Reynolds stress tensor, and to see to which input features the TBRF algorithm relates these processes. For example, does the machine learning algorithm relate the strongly 1-component behaviour of turbulence on top of the curved backward facing step (around $x/h = 1.0$, $y/h = 1.0$ in Figure 6.7a) to features related to shear-generation, corresponding to the explanation in Bentaleb et al. (2012)? And what features does the algorithm see as relevant for the strongly 1-component turbulence in the converging part of the periodic hills (around $x/h = 8.0$ in Figure 4.3), which is explained to be caused by the vortex splatting effect in Fröhlich et al. (2005)? Using for example the physically interpretable features from feature set 3 in Table 3.1 could make this analysis more meaningful.

Lastly it should be mentioned that the application of machine learning algorithms using a tensor basis might be useful in other fields than turbulence modelling as well. One example might be the modelling of non-linear relations between stresses and strains in solid mechanics. See for example Huston (1965), where the stresses are modelled using higher order tensors based on the strain.

9.2.3 Machine learning and turbulence modelling in general

So far, the machine learning algorithms used to predict the anisotropy tensor only use local flow quantities at a certain location on the mesh. Including non-local effects in the machine learning algorithm could be a possible topic for future research. It is proposed to use the convolutional neural network (CNN), which is often used in image processing. By using a type of locally connected neural network, this algorithm makes use of filters, which 'scan' the area around a given point. This way, the algorithm is able to extract higher-level features in images. For example, in the case of classifying pictures of cars, the consequent layers might activate in the presence of curved lines, wheels, and eventually the entire body of a car. Perhaps this same idea would enable the neural network to identify vortex structures in the case of a turbulence modelling scenario, or convecting turbulent quantities. In Appendix A preliminary investigations regarding the application of a convolutional neural network will be presented.

It should be further investigated how much an increase in the Reynolds number will affect predictions by the machine learning algorithms. In the best case scenario, one would like to use algorithms as presented in this thesis to make predictions for higher Reynolds numbers, while training them on lower Reynolds numbers where DNS/LES is still viable. Another option would be looking at experimental data instead of numerical data for training, where it might be more viable to attain data for higher Reynolds numbers, think for example about using data obtained by particle image velocimetry (PIV).

Bibliography

S. Banerjee, R. Krahl, F. Durst, and C. Zenger. Presentation of anisotropy properties of turbulence, invariants versus eigenvalue approaches. *Journal of Turbulence*, 8(32), 2007. ISSN 1468-5248. doi: 10.1080/14685240701506896.

Y. Bentaleb, S. Lardeau, and M. Leschziner. Geometric properties of particle trajectories in turbulent flows. *Journal of Turbulence*, 13:1–28, 2012. ISSN 1468-5248. doi: 10.1080/14685248.2011.637923. URL http://arxiv.org/abs/0901.3521.

P. C. Bhat and H. B. Prosper. Bayesian Neural Networks. In *Statistical Problems in Particle Physics, Astrophysics and Cosmology*, pages 151–154, 2005.

R. P. Brent. An algorithm with guaranteed convergence for finding a zero of a function. In *Algorithms for Minimization without Derivatives*. Prentice-Hall, 1973. ISBN 0-13-022335-2.

M. Breuer, N. Peller, C. Rapp, and M. Manhart. Flow over periodic hills - Numerical and experimental study in a wide range of Reynolds numbers. *Computers and Fluids*, 38(2): 433–457, 2009. ISSN 00457930. doi: 10.1016/j.compfluid.2008.05.002. URL http://dx.doi.org/10.1016/j.compfluid.2008.05.002.

T. J. Craft, B. E. Launder, and K. Suga. Development and application of a cubic eddy-viscosity model of turbulence. *International Journal of Heat and Fluid Flow*, 17(2):108–115, 1996. ISSN 0142727X. doi: 10.1016/0142-727X(95)00079-6.

K. Duraisamy, Z. J. Zhang, and P. S. Anand. New Approaches in Turbulence and Transition Modeling Using Data-driven Techniques. *53rd AIAA Aerospace Sciences Meeting*, (January):1–14, 2015. ISSN 978-1-62410-343-8. doi: doi:10.2514/6.2015-1284.

P. A. Durbin. Some Recent Developments in Turbulence Closure Modeling. *Annual Review of Fluid Mechanics*, 50:77–103, 2018. ISSN 1947-5454. doi: 10.1146/annurev-fluid-122316-045020. URL http://arxiv.org/abs/1507.05135.

M. Emory and G. Iaccarino. Componentality-based wall-blocking for RANS models. In *Stanford University, Center for Turbulence Research, Annual Research Briefs*, pages 193–208, 2014.

M. Emory, J. Larsson, and G. Iaccarino. Modeling of structural uncertainties in Reynolds-averaged Navier-Stokes closures. *Physics of Fluids*, 25(11), 2013. ISSN 10706631. doi: 10.1063/1.4824659.

J. Fröhlich, C. P. Mellen, W. Rodi, L. Temmerman, and M. A. Leschziner. Highly resolved large-eddy simulation of separated flow in a channel with streamwise periodic constrictions. *Journal of Fluid Mechanics*, 526:19–66, 2005. ISSN 00221120. doi: 10.1017/S0022112004002812.

G. Gallego and C. Cuevas. On the Mahalanobis Distance Classification Criterion for Multidimensional Normal Distributions. *IEEE Transactions on Signal Processing*, 61(17):4387–4396, 2013. doi: 10.1109/TSP.2013.2269047.

T. B. Gatski and C. G. Speziale. On Explicit Algebraic Stress Models for Complex Turbulent Flows. *Journal of Fluid Mechanics*, 254:59–78, 1993. doi: 10.1017/S0022112093002034.

C. J. Greenshields. *OpenFOAM Programmer's Guide*. 2015. URL http://foam.sourceforge.net/docs/Guides-a4/ProgrammersGuide.pdf.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2008.

R. L. Huston. Nonlinear stress-strain equations. *International Journal of Solids and Structures*, 1(4):463–470, 1965. ISSN 00207683. doi: 10.1016/0020-7683(65)90009-0.

S. Jovic and D. Driver. Backward-Facing Step Measurements at Low Reynolds Number. *NASA Technical Memorandum 108807*, 1994.

A. Karpathy. Course notes CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, 2017. URL http://cs231n.github.io/convolutional-networks/.

D. P. Kingma and J. L. Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/pdf/1412.6980.pdf.

D. A. Knoll and D. E. Keyes. Jacobian-free NewtonKrylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193:357–397, 2004. doi: 10.1016/j.jcp.2003.08.010.

M. Kubat. *An Introduction to Machine Learning*. Springer, 2015. ISBN 978-3-319-20009-5. doi: 10.1007/978-3-319-20010-1. URL http://link.springer.com/10.1007/978-3-319-20010-1.

B. Launder and B. Sharma. Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc. *Letters in Heat and Mass Transfer*, 1(2):131–137, nov 1974. ISSN 0094-4548. doi: 10.1016/0094-4548(74)90150-7. URL https://www.sciencedirect.com/science/article/pii/0094454874901507.

B. Launder and D. Spalding. The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 3(2):269–289, 1974. ISSN 00457825. doi: 10.1016/0045-7825(74)90029-2.

B. Launder, G. Reece, and W. Rodi. Progress in the Development of a Reynolds Stress Turbulence Closure. *Journal of Fluid Mechanics*, 68(October):537–566, 1975. doi: 10.1017/S0022112075001814.

J.-P. Laval and M. Marquillie. Direct Numerical Simulations of Converging-Diverging Channel Flow. In *Progress in Wall Turbulence: Understanding and Modeling*, pages 203–209, 2011. ISBN 9789048196029. doi: 10.1007/978-90-481-9603-6.

H. Le and P. Moin. Direct numerical simulation of turbulent flow over a backward-facing step. In *Stanford University, Center for Turbulence Research, Annual Research Briefs*, pages 161–173, 1992.

H. Le, P. Moin, and J. Kim. Direct numerical simulation of turbulent flow over a backward-facing step. *Journal of Fluid Mechanics*, 330:349–374, 1997.

F. Lien, W. Chen, and M. A. Leschziner. Low-Reynolds-Number Eddy-Viscosity Modelling Based on Non-Linear Stress-Strain/Vorticity Relations. In *Engineering Turbulence Modelling and Experiments 3*, pages 91–100. 1996.

J. Ling and J. Templeton. Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty. *Physics of Fluids*, 27(8), 2015. ISSN 10897666. doi: 10.1063/1.4927765.

J. Ling, R. Jones, and J. Templeton. Machine learning strategies for systems with invariance properties. *Journal of Computational Physics*, 318:22–35, 2016a. ISSN 0021-9991. doi: 10.1016/j.jcp.2016.05.003.

J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807(2016): 155–166, 2016b. ISSN 0022-1120. doi: 10.1017/jfm.2016.615.

J. Ling, A. Kurzawski, and J. Templeton. Tensor Basis Neural Network, 2017a. URL https://github.com/tbnn/tbnn.

J. Ling, A. Ruiz, G. Lacaze, and J. Oefelein. Uncertainty Analysis and Data-Driven Model Advances for a Jet-in-Crossflow. *Journal of Turbomachinery*, 139(February):1–9, 2017b. doi: 10.1115/1.4034556.

J. L. Lumley. Computational modeling of turbulent flows. *Advances in applied mechanics*, 18:123–176, 1979. ISSN 00652156. doi: 10.1016/S0065-2156(08)70266-7.

J. L. Lumley and G. R. Newman. The return to isotropy of homogeneous turbulence. *Journal of Fluid Mechanics*, 82(1):161–178, 1977. doi: 10.1017/S0022112077000585.

B. T. S. Lund and E. A. Novikov. Parameterization of subgrid-scale stress by the velocity gradient tensor. *Center for Turbulence Research Annual Research Briefs*, pages 27–43, 1992.

N. Meinshausen. Quantile Regression Forests. *Journal of Machine Learning Research*, 7: 983–999, 2006.

C. P. Mellen, J. Frohlic, and W. Rodi. Large Eddy Simulation of the flow over periodic hills. In *16th IMACS World Congress*, pages 3–8, 2000.

L. Mentch and G. Hooker. Quantifying Uncertainty in Random Forests via Confidence Intervals and Hypothesis Tests. *Journal of Machine Learning Research*, 17:1–41, 2016.

F. R. Menter. Zonal Two Equation k-w Turbulence Models for Aerodynamic Flows. In *AIAA 24th Fluid Dynamics Conference*, 1993. doi: 10.2514/6.1993-2906.

M. Milano and P. Koumoutsakos. Neural Network Modeling for Near Wall Turbulent Flow. *Journal of Computational Physics*, 182:1–26, 2002. doi: 10.1006/jcph.2002.7146.

E. J. Parish and K. Duraisamy. A paradigm for data-driven predictive modeling using field inversion and machine learning. *Journal of Computational Physics*, 305:758–774, 2016. ISSN 0021-9991. doi: 10.1016/j.jcp.2015.11.012. URL http://dx.doi.org/10.1016/j.jcp.2015.11.012.

R. Pecnik and G. Iaccarino. Predictions of turbulent secondary flows using the v2 - f model. In *Stanford University, Center for Turbulence Research, Annual Research Briefs*, pages 333–343, 2007.

A. Pinelli, M. Uhlmann, A. Sekimoto, and G. Kawahara. Reynolds number dependence of mean flow structure in square duct turbulence. *Journal of Fluid Mechanics*, 644:107, 2010. ISSN 0022-1120. doi: 10.1017/S0022112009992242. URL http://www.journals.cambridge.org/abstract{_}S0022112009992242.

S. B. Pope. A more general effective-viscosity hypothesis. *Journal of Fluid Mechanics*, 72(2): 331–340, 1975. ISSN 0022-1120. doi: 10.1017/S0022112075003382.

S. B. Pope. *Turbulent Flows*. Cambridge University Press, 2000. ISBN 9780521591256.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. ISBN 026218253X.

J. Rotta. Statistische Theorie nichthomogener Turbulenz. *Zeitschrift fur Physik*, 129(6): 547–572, 1951.

C. Rumsey. NASA Langley Research Center, Turbulence Modeling Resource, 2017. URL https://turbmodels.larc.nasa.gov/.

S. Russell and P. Norvig. *Artificial Inteligence: A Modern Approach*. Prentice Hall, 3rd editio edition, 2010. ISBN 9780136042594.

Scikit-learn. User Guide, 2017. URL http://scikit-learn.org/stable/user{_}guide.html.

D. Scott. *Multivariate Density Estimation. Theory, Practice and Visualization*. Wiley, New York, 1992.

T.-h. Shih, J. Zhu, and J. L. Lumley. A Realizable Reynolds Stress Algebraic Equation Model. *NASA Technical Memorandum 105993*, 1993.

M. A. Shipp, K. N. Ross, P. Tamayo, A. P. Weng, J. L. Kutok, R. C. T. Aguiar, M. Gaasenbeek, M. Angelo, M. Reich, G. S. Pinkus, T. S. Ray, M. A. Koval, K. W. Last, A. Norton, T. A. Lister, J. Mesirov, D. S. Neuberg, E. S. Lander, J. C. Aster, and T. R. Golub. Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature Medicine*, 8(1):68–74, 2002. ISSN 1078-8956. doi: 10.1038/nm0102-68.

K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

A. P. Singh, S. Medida, and K. Duraisamy. Machine Learning-augmented Predictive Modeling of Turbulent Separated Flows over Airfoils. URL http://arxiv.org/abs/1608.03990. 2016.

A. P. Singh, K. Duraisamy, and S. Pan. Characterizing and Improving Predictive Accuracy in Shock-Turbulent Boundary Layer Interactions Using Data-driven Models. *55th AIAA Aerospace Sciences Meeting*, (January):1–14, 2017. doi: 10.2514/6.2017-0314. URL http://arc.aiaa.org/doi/10.2514/6.2017-0314.

S. Sivaraman and M. M. Trivedi. Active learning for on-road vehicle detection: a comparative study. *Machine Vision and Applications*, pages 1–13, 2011. ISSN 09328092. doi: 10.1007/s00138-011-0388-y.

G. G. Slabaugh. Computing Euler angles from a rotation matrix, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.371.6578{&}rep=rep1{&}type=pdf.

J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis. CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences. *NASA Technical Report NASA/CR-2014-21878*, 2014. ISSN 1098-6596. doi: 10.1017/CBO9781107415324.004. URL https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140003093.pdf.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems*, pages 2951–2959, 2012.

P. Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. In *AIAA 30th Aerospace Sciences Meeting and Exhibit*, 1992. doi: 10.2514/6.1992-439.

C. G. Speziale, S. Sarkar, and T. B. Gatski. Modelling the pressure-strain correlation of turbulence: an invariant dynamical systems approach. *Journal of Fluid Mechanics*, 227 (December):245–272, 1991. ISSN 1469-7645. doi: 10.1017/S0022112091000101.

B. Tracey, K. Duraisamy, and J. J. Alonso. Application of Supervised Learning to Quantify Uncertainties in Turbulence and Combustion Modeling. *51st AIAA ASM Including the New Horizons Forum and Aerospace Exposition*, (January), 2013. doi: 10.2514/6.2013-259.

J.-X. Wang, J. Wu, J. Ling, G. Iaccarino, and H. Xiao. A Comprehensive Physics-Informed Machine Learning Framework for Predictive Turbulence Modeling. URL http://arxiv.org/abs/1701.07102. 2017a.

J.-X. Wang, J.-L. Wu, and H. Xiao. Physics-Informed Machine Learning for Reconstructing Reynolds Stress Modeling Discrepancies Based on DNS Data. *Physical Review Fluids*, 2: 1–41, 2017b. doi: 10.1103/PhysRevFluids.2.034603. URL http://arxiv.org/abs/1606.07987.

D. C. Wilcox. Formulation of the k-w Turbulence Model Revisited. *AIAA Journal*, 46(11): 2823–2838, 2008. ISSN 0001-1452. doi: 10.2514/1.36541.

J. Wu, R. Sun, S. Laizet, and H. Xiao. Representation of Reynolds Stress Perturbations with Application in Machine-Learning-Assisted Turbulence Modeling. URL https://arxiv.org/pdf/1709.05683.pdf. 2017a.

J. Wu, R. Sun, Q. Wang, and H. Xiao. On the Conditioning of Machine-Learning-Assisted Turbulence Modeling (Abstract). In *70th Annual Meeting of the APS Division of Fluid Dynamics*, 2017b. URL http://meetings.aps.org/link/BAPS.2017.DFD.E31.1.

J.-L. Wu, J.-X. Wang, H. Xiao, and J. Ling. Physics-Informed Machine Learning for Predictive Turbulence Modeling: A Priori Assessment of Prediction Confidence. URL http://arxiv.org/abs/1606.07987. 2016.

H. Xiao, J.-l. Wu, J.-x. Wang, and E. G. Paterson. Are Discrepancies in RANS Modeled Reynolds Stresses Random? URL https://arxiv.org/abs/1606.08131. 2016.

Z. J. Zhang and K. Duraisamy. Machine Learning Methods for Data-Driven Turbulence Modeling. *22nd AIAA Computational Fluid Dynamics Conference*, (June):1–18, 2015. doi: 10.2514/6.2015-2460. URL http://dx.doi.org/10.2514/6.2015-2460.

Appendix A

Convolutional Neural Network for predicting the anisotropy tensor

In this appendix, the preliminary design of the convolutional neural network for predicting b_{ij} will be presented along with some preliminary results. The idea for implementing this type of neural network came forth out of the fact that the machine learning approaches presented so far in this report only take in account local quantities in the flow field to make predictions. The convolutional neural network uses so called filters, which could take a part of the domain as input instead of just on cell of the mesh.

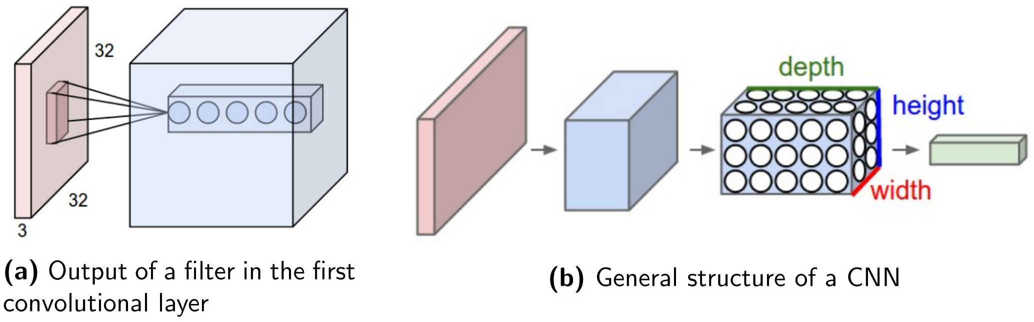
A.1 Theory and Methodology

A good introduction to convolutional neural networks is given in Karpathy (2017). As a quick summary, its basic principles will be explained using Figure A.1. In the case of image processing, the input would have a spatial resolution, e.g. 32×32 in case of a very small image, and a depth of 3 corresponding to the red, blue, and green color channels. When processing such an image, using a fully connected neural network (as presented in e.g. Figure 2.3b) would be very expensive, since a single neuron in the first hidden layer would have $32 \times 32 \times 3$ weights connected to it. Instead, a small region in the domain of the image is connected to a small part of the neurons in the first hidden layer, or convolutional layer. This can be represented by convolutional filters, illustrated in Figure A.1a.

The filters in a CNN have a small spatial resolution, and have the same depth as the input volume. A number of filters can be present in the convolutional layer. Each filter 'scans' or convolves across the spatial dimension of the input. The size of the steps it makes in the spatial dimensions is called the stride. At each location which the filter passes, it produces an output, represented by the blue cube in Figure A.1a. The height and width of the cube (as defined in Figure A.1b) depend on the size of the input, as well as the stride of the filter.

Furthermore, it depends on the operation which the filter performs at the boundary. For example, it can be chosen to use padding, which extends the spatial domain of the input such that it is possible to obtain an output with the same spatial dimensions. The depth of the output depends on the amount of filters used, in Figure A.1a this would correspond to five.

At each consecutive step the CNN convolutes a set of filters with the output produced by the previous convolutional layer. So called pooling layers can be used after convolutional layers, which downsample the spatial resolution (e.g. going from 32×32 in the spatial dimension to 16×16). In the illustration in Figure A.1b it can be seen that the height and width of the output volumes decrease, e.g. due to using no padding, a stride higher than 1, or using intermediate pooling layers. Eventually it creates an output with a very small width and height (e.g. 1×1), and a larger depth. This can serve as an input for a fully connected neural network, to for example classify the image into a certain category.



(a) Output of a filter in the first convolutional layer

(b) General structure of a CNN

Figure A.1: Summarized workings of the convolutional neural network, reproduced from Karpathy (2017)

The convolutional neural network for predicting the anisotropy tensor was implemented using the `keras` machine learning library in `python`. Four flow fields from the square duct were used for training, at $Re = 2,400$, $Re = 2,600$, $Re = 2,900$, and $Re = 3,200$. A total of 11 features were used, based on S_{ij}, Ω_{ij} and ∇k. As of now, the neural network is still trained on the individual components of b_{ij}, meaning the response will have a shape of $9 \times N_x \times N_y$, where N_x and N_y is the spatial resolution of the flow domain.

A lot of possibilities are available with respect to the architecture of the CNN. Since often pooling layers are included in the architecture, this means the 'spatial' resolution gets smaller for consecutive layers of the neural network. What is often seen, is that the amount of filters increases for the consecutive layers. Since the spatial resolution decreases because of the pooling layers, this keeps the increase in the amount of parameters (the weights used for the connections in the neural network) relatively limited. In the case presented here no pooling layers are used, since the spatial resolution of the input (features in the flow domain) should match the spatial resolution of the output (prediction of the anisotropy tensor in the flow domain). This means it will be more expensive to use a large amount of layers compared with a neural network which does use pooling.

For the amount of filters (n_f), and the filter sizes in both spatial directions (n_x, n_y), the following settings were used for the four convolutional layers by trial and error:

- Layer 1: $n_f \times n_x \times n_y = 16 \times 5 \times 5$
- Layer 2: $n_f \times n_x \times n_y = 32 \times 9 \times 9$
- Layer 3: $n_f \times n_x \times n_y = 64 \times 11 \times 11$
- Layer 4: $n_f \times n_x \times n_y = 9 \times 15 \times 15$

As can be seen, the filter sizes increase in consecutive layers, possibly allowing he network to learn more about spatial structures in the flow domain. The amount of filters increases as well with factors of 2, as inspired by CNN architectures presented in literature, see e.g. Simonyan and Zisserman (2015). The last layer of the neural network contains only 9 filters. This results in an output of the network of the size $9 \times N_x \times N_y$, which is the shape of the predicted anisotropy tensor b_{ij}.

For the eventual goal, not the individual components of b_{ij} would be predicted, but for example the ten scalar coefficients of the tensor basis series, $g^{(m)}$. These coefficients present in the output from the last convolutional layer, would then be multiplied with the appropriate basis tensors $T^{(m)}$. This way a Galilean invariant prediction for b_{ij} could be obtained, where training and testing can be done using different flow cases. The layer containing the basis tensors would have a size of $10 \times 9 \times N_x \times N_y$, corresponding to ten basis tensors, with nine components each.

Figure A.2 presents the preliminary design of this convolutional neural network which uses a tensor basis. The part marked in red is already implemented, which is mapping the input features in the flow domain, to an output which has the size of 9 when mapping to the 9 individual components, or 10 in the case the ten scalar coefficients in the tensor basis are predicted. The depth of the output, 10, then corresponds to the amount of filters used in the last convolutional layer. In the figure the case has been depicted when 4 convolutional layers are used, leading to $n_{f,4} = 10$. In the lower part of the architecture, 9 sets of volumes are depicted, each corresponding to one tensor component of the basis tensors. This could be reduced to six, since the anisotropy tensor is symmetric. The 10 layers in each of the volumes need to be multiplied with ten layers from the convolutional neural network. Repeating this 9 times for each component of the tensor, results in the prediction of b_{ij}.

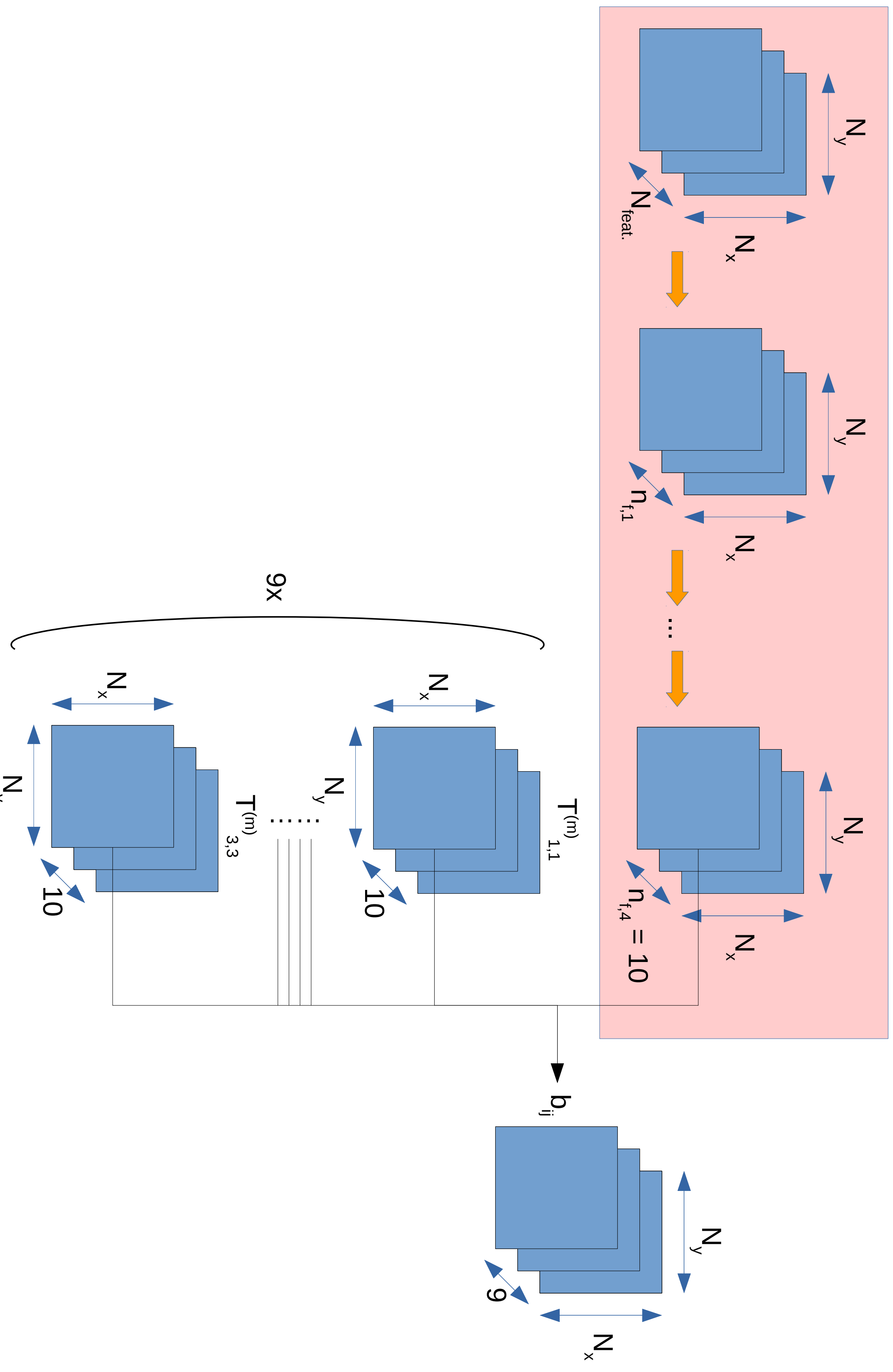
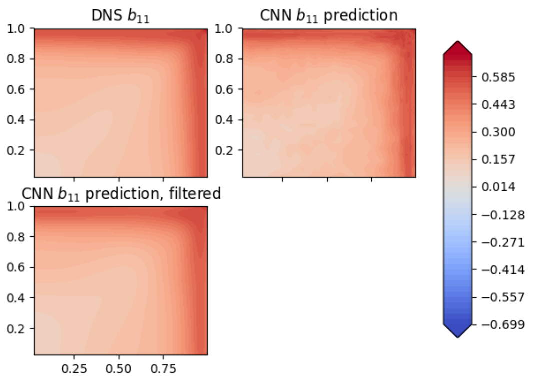


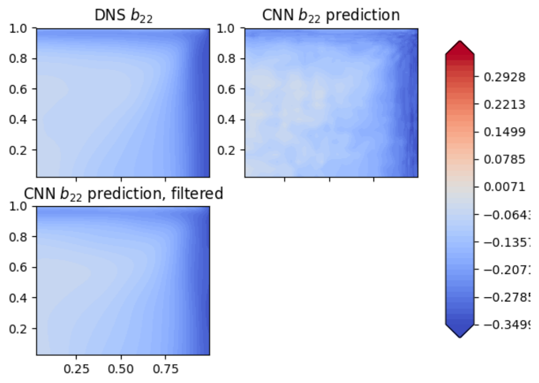
Figure A.2: Preliminary design of the Tensor Basis Convolutional Neural Network

A.2 Results

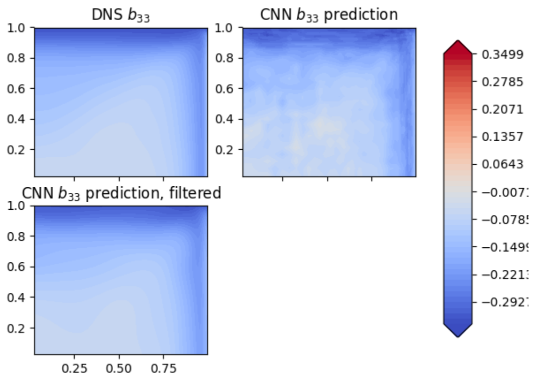
Figure A.3 presents the predictions for the anisotropy tensor by the convolutional neural network. In the upper left plots of the figures the reference data from DNS is represented in the flow domain of the square duct. In the upper right plots the predictions from the CNN are represented for the anisotropy tensor components. As can be seen, a good match is obtained with respect to the DNS data. Some noise is visible however, especially when looking at b_{23}, which has the smallest magnitude overall. After smoothing the predictions spatially by using a Gaussian filter, a very good match with respect to DNS is obtained. Further investigations are necessary to see what causes the noise for the unfiltered predictions. Some possibilities might be adjusting the filter sizes to include more of the flow domain, possibly creating more correlation between the predictions located closely to each other.



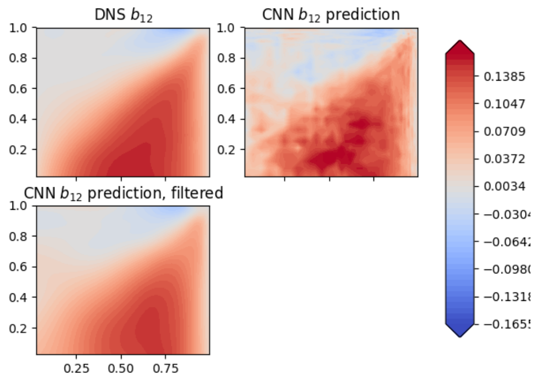
(a) b_{11}



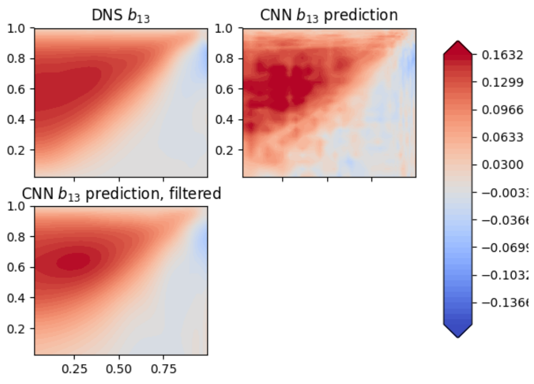
(b) b_{22}



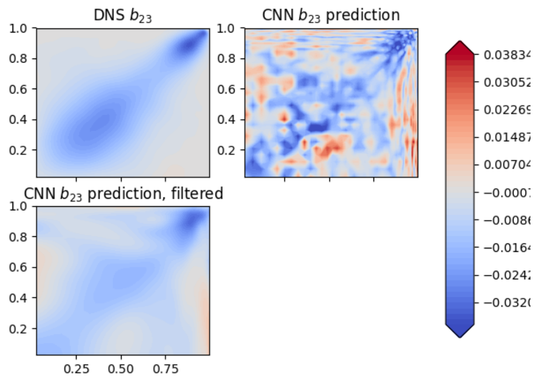
(c) b_{33}



(d) b_{12}



(e) b_{13}



(f) b_{23}

Figure A.3: Results for the preliminary convolutional neural network

A.3 Recommendations for future work

Some recommendations are given when further investigating the proposed CNN for turbulence modelling. Right now, each mesh cell of the computational grid is used as an input, or expressing it in terms of image processing, each mesh cell is represented by a 'pixel'. This raises some issues with the scaling of the convolutional filters, since close to the wall they filter a relatively small part of the flow domain, and further away from the wall a relatively large part. It is likely a uniform scaling should be used, where the data is first interpolated onto a regular grid with equal spacing. When using the CNN between different flow cases, the scaling of the filters should be investigated as well, since the filter in a low Reynolds number flow should capture the same structures as for a high Reynolds number flow. Perhaps scaling the filter with a turbulent length scale could be a solution. Instead of using the convolutional neural network with tensor basis included, first the regular architecture (Figure A.2, marked red) could be used to make predictions on the barycentric map, training and testing on different flow cases to see how well this idea generalizes.

The last part for the implementation of the tensor basis convolutional neural network, connecting the scalar coefficients to the basis tensors, should be relatively trivial when the appropriate functions in `keras` can be found. However, it is likely that it will make training of the network more difficult, which was also observed when comparing a standard multilayer perceptron to the TBNN algorithm from Ling et al. (2016b).

Using a filter which is represented by a rectangle in the flow domain is not necessary at all. Perhaps it could be more interesting to use convolutional filters which combine points in the flow which are likely to influence each other. One could look which points lie upstream of a given location in the flow, and use these in the filter. This could be a nice way of capturing for example the convection of turbulent structures.

The idea presented here is created for 2-dimensional flow cases. In theory this method could also be used for 3-dimensional flow cases however. There are 3-dimensional convolutional filters which can be used, which would create a 4-dimensional output containing the tensor basis coefficients.

Appendix B

Modified OpenFOAM solvers for propagating the Reynolds stress

B.1 Approach 1: inserting the Reynolds stress in the momentum equation

B.1.1 UEqn.H

The modified version of the `UEqn.H` file which accepts a given Reynolds stress field is given below.

```
// Momentum predictor

MRF.correctBoundaryVelocity(U);

tmp<fvVectorMatrix> UEqn
(
      fvm::div(phi, U)
    + fvc::div( R )
    + fvc::laplacian(turbulence->nut(), U, "laplacian(nuEff,U)")
    - fvm::laplacian(nu+turbulence->nut(), U)
    - fvc::div(nu*dev2(T(fvc::grad(U))))
  ==
    fvOptions(U)
);

UEqn().relax();

fvOptions.constrain(UEqn());

solve(UEqn() == -fvc::grad(p));

fvOptions.correct(U);
```

B.1.2 createFields.H

The Reynolds stress field needs to be initialized before it can be used. This is done in the `createFields.H` file, where the following lines were added:

```
   Info<< "Reading field R\n" << endl;
const volSymmTensorField R
(
    IOobject
    (
        "R",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::NO_WRITE
    ),
    mesh
);
```

B.2 Approach 2: inserting the Reynolds stress anisotropy and solving a modified k-equation

B.2.1 UEqn.H

```
    // Momentum predictor

    MRF.correctBoundaryVelocity(U);

    tmp<fvVectorMatrix> UEqn
    (
        fvm::div(phi, U)
    + fvc::div( ((2.0/3.0)*turbulence->k()*I +  2.0*turbulence->k()*bij), "div(
        devRhoReff)")
    + fvc::laplacian(turbulence->nut(), U, "laplacian(nuEff,U)")
    - fvm::laplacian(nu+turbulence->nut(), U)
    - fvc::div(nu*dev2(T(fvc::grad(U))))
      ==
        fvOptions(U)
    );

    UEqn().relax();

    fvOptions.constrain(UEqn());

    solve(UEqn() == -fvc::grad(p));

    fvOptions.correct(U);
```

B.2.2 Adjusted turbulence model

Changes were made in the $k-\omega$ and Launder-Sharma $k-\epsilon$ turbulence model, where the production term in the k-equation was modified. As an example, changes to the Launder-Sharma $k-\epsilon$ turbulence model will be shown below. Only the most relevant changes are be presented, e.g. declarations and initialization of the variables are left out.

```
template<class BasicTurbulenceModel>
void LSKEpsMod2<BasicTurbulenceModel >::correct()
{
    if (!this->turbulence_)
    {
        return;
    }

    // Local references
    const alphaField& alpha = this->alpha_;
    const rhoField& rho = this->rho_;
    const surfaceScalarField& alphaRhoPhi = this->alphaRhoPhi_;
    const volVectorField& U = this->U_;
    volScalarField& nut = this->nut_;

    eddyViscosity<RASModel<BasicTurbulenceModel> >::correct();

    volScalarField divU(fvc::div(fvc::absolute(this->phi(), U)));

    tmp<volTensorField> tgradU = fvc::grad(U);
    volScalarField S2(2*magSqr(dev(symm(tgradU()))));
    volScalarField magS(sqrt(S2));

    volScalarField eta(magS*k_/epsilon_);
    volScalarField C1(max(eta/(scalar(5) + eta), scalar(0.43)));

    volScalarField G(this->GName(), nut*(tgradU() && dev(twoSymm(tgradU()))));
    // Calculate Reynolds stress using the given anisotropy tensor
    volSymmTensorField R = (2.0/3.0)*k_*I + 2.0*k_*bij_;
    // Calculate the adjusted production term for the k-equation
    volScalarField kProd = -(R && fvc::grad(U));

    // Update epsilon and G at the wall
    epsilon_.boundaryField().updateCoeffs();

    // Dissipation equation
    tmp<fvScalarMatrix> epsEqn
    (
        fvm::ddt(alpha, rho, epsilon_)
      + fvm::div(alphaRhoPhi, epsilon_)
      - fvm::laplacian(alpha*rho*DepsilonEff(), epsilon_)
     ==
        C1*alpha*rho*magS*epsilon_
      - fvm::Sp
        (
            C2_*alpha*rho*epsilon_/(k_ + sqrt(this->nu()*epsilon_)),
            epsilon_
        )
      + epsilonSource()
    );

    epsEqn().relax();

    epsEqn().boundaryManipulate(epsilon_.boundaryField());

    solve(epsEqn);
    bound(epsilon_, this->epsilonMin_);


    // Turbulent kinetic energy equation
    tmp<fvScalarMatrix> kEqn
    (
        fvm::ddt(alpha, rho, k_)
      + fvm::div(alphaRhoPhi, k_)
      - fvm::laplacian(alpha*rho*DkEff(), k_)
     ==
        kProd
      - fvm::Sp(alpha*rho*epsilon_/k_, k_)
      + kSource()
    );


    kEqn().relax();
    solve(kEqn);
    bound(k_, this->kMin_);

    correctNut(tgradU(), S2, magS);
}
```

B.3 Approach 3: the continuation solver

For the continuation solver, the `simpleFoam` solver is adjusted to accept the anisotropy tensor predicted by the machine learning algorithm, $b_{ij,ML}$. It blends this prediction with b_{ij} from the $k-\omega$ turbulence model. Code for the adjusted `simpleFoam` solver is presented in Section B.3.1. Code for the adjusted turbulence model is presented in Section B.3.2, where again $b_{ij,ML}$ is blended with b_{ij}, after which it is used for the production term of the k and ω transport equations. Again only the most important changes are presented here.

B.3.1 UEqn.H

```
    MRF.correctBoundaryVelocity(U);

    dimensioned<scalar> tend   = runTime.endTime().value();
    dimensioned<scalar> tnow   = atoi(runTime.timeName().c_str());

    dimensionedScalar xi =  ((xitm - xi0m)/(tend - tstarm)) * tnow + xi0m;

    // Decide whether the final value of xi is reached, or whether further
       ramping is necessary
    if (tnow + tstarm - tend > iteraux)
     {
             xi =  xitm;
          }
    else {
             xi = xi;
    }


    Info<< "Momentum equation xi = " << xi << nl << endl;

    tmp<fvVectorMatrix> UEqn
    (
        fvm::div(phi, U)
      + MRF.DDt(U)
      + (1-xi)*(turbulence->divDevReff(U))
      + xi*fvc::div(2*turbulence().k()*((bijML)+ (1.0/3.0)*I))
      - xi*fvc::div((turbulence().alpha()*nu)*dev2(T(fvc::grad(U))))
      - xi*fvm::laplacian(turbulence().alpha()*nu, U)
     ==
        fvOptions(U)
    );

    UEqn().relax();

    fvOptions.constrain(UEqn());

    solve(UEqn() == -fvc::grad(p));

    fvOptions.correct(U);
```

B.3.2 Adjusted turbulence model

```
template<class BasicTurbulenceModel>
void kOmegaCont2<BasicTurbulenceModel>::correct()
{
    if (!this->turbulence_)
    {
        return;
    }

    // Local references
    const alphaField& alpha = this->alpha_;
    const rhoField& rho = this->rho_;
    const surfaceScalarField& alphaRhoPhi = this->alphaRhoPhi_;
    const volVectorField& U = this->U_;
    volScalarField& nut = this->nut_;

    eddyViscosity<RASModel<BasicTurbulenceModel> >::correct();

    volScalarField divU(fvc::div(fvc::absolute(this->phi(), U)));

    tmp<volTensorField> tgradU = fvc::grad(U);

    dimensioned<scalar> tend_   = this->runTime_.endTime().value();
```

```
dimensioned<scalar> tnow_   = atoi(this->runTime_.timeName().c_str());

xi_ = ((xit - xi0)/(tend_ - tstar)) * tnow_ + xi0;

if (tnow_ + tstar - tend_ > iteraux_)
 {
         xi_ =  xit;
       }
else {
         xi_ = xi_;
}

Info<< "Turbulence model xi_ = " << xi_ << nl << endl;


volScalarField G
(
    this->GName(),
    //nut*(tgradU() && dev(twoSymm(tgradU())))
xi_*(-2)*k_*((bijML_ + (1.0/3.0)*I) && tgradU())                + (1-xi_)*
   nut*(tgradU() && dev(twoSymm(tgradU())))
);

tgradU.clear();

// Update omega and G at the wall
omega_.boundaryField().updateCoeffs();

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

omegaEqn().relax();

omegaEqn().boundaryManipulate(omega_.boundaryField());

solve(omegaEqn);
bound(omega_, this->omegaMin_);


// 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(Cmu_*alpha*rho*omega_, k_)
    );

    kEqn().relax();
    solve(kEqn);
    bound(k_, this->kMin_);

    correctNut();
}
```

Appendix C

Optimizing the TBDT Training Phase

C.1 Parallelization Test

```
import numpy as np
import time
import matplotlib.pyplot as plt

from multiprocessing import Pool
from functools import partial

from scipy.optimize import minimize_scalar

def findJMin(i1, data):
    """
    Calculate best splitting fitness for a given input feature (i1) and input
        array
    where the amount of rows correspond to the amount of features, and the
        amount
    of columns to the amount of samples
    """
    output = 1e10 # splitting fitness, initialized at an arbitrary high value
    #print('start')

    for i2 in range(data.shape[1]): # go through the samples (columns)


        splitPoint_feat = i1
        splitPoint_val = data[i1,i2]

        # divide data into left and right bin
        i_left = data[splitPoint_feat,:]<=splitPoint_val
        data_l = data[:,i_left]
        data_r = data[:,~i_left]

        # Uncomment to check whether process is indeed running in parallel:
```

```
#        if i2%500 == 0:
#            print(i2)

        # some dummy calculations similar to how the decision tree works:
            calculate difference w.r.t. mean
        # and take RMS error. Save splitting fitness if it is lower than best
            fitness so far
        diff_l = data_l - np.mean(data_l)
        diff_r = data_r - np.mean(data_r)

        J = np.sqrt(np.mean(np.square(np.append(diff_l,diff_r))))

        if J < output:
            output= J

    #print('done')
    return(output) # return best splitting fitness found


def findJMin_sort(i1, data):
    """
    Sort the data before finding the minimum: (O(n log n) complexity, instead
        of the O(n^2) complexity of the function above
    """
    asort = np.argsort(data[i1,:])
    data2 = data[:, asort]
    Js = np.empty(data.shape[1]-2)
    for i2 in range(1,data.shape[1]-1):
        data_l = data2[:,:i2]
        data_r = data2[:,i2:]
        diff_l = data_l - np.mean(data_l)
        diff_r = data_r - np.mean(data_r)
        Js[i2-1] = np.sqrt(np.mean(np.square(np.append(diff_l,diff_r))))
    return np.min(Js)


def findJMin_opt(i1, data):
    """
    Use the Brent optimization algorithm for finding the minimum of J
    """
    out = np.inf
    asort = np.argsort(data[i1,:])
    data = data[:, asort]
    def objfn_J(ifloat):
        i = int(ifloat)
        data_l = data[:,:i]
        data_r = data[:,i:]
        diff_l = data_l - np.mean(data_l)
        diff_r = data_r - np.mean(data_r)
        return np.sqrt(np.mean(np.square(np.append(diff_l,diff_r))))
    res = minimize_scalar(objfn_J, args=(), method='brent',
                          bracket=(1, data.shape[1]/2, data.shape[1]-1), tol=
                              None,
                          options={'xtol': 1.e-08,
                                   'maxiter': 200})
    return res.fun
```

```
if __name__ == '__main__':
    np.random.seed(12345)
    nvar = 4
    data = np.random.random([nvar, 40000])
    partial_findJMin = partial(findJMin_opt, data=data)  #Change the function
       for finding the minimum of J here

    print "Serial",
    start = time.time()
    for i in range(nvar):
        partial_findJMin(i)
    print "%.2f s" % (time.time()-start,)

    for nproc in [1,2,4]:
        print "%d-proc" % nproc,
        start = time.time()
        pool = Pool(processes=nproc)
        pool.map(partial_findJMin, range(nvar))
        print "%.2f s" % (time.time()-start,)
```

C.2 Using an optimization algorithm for finding the optimum splitting feature/value

Figure C.1 presents the parameter study for using an optimization algorithm instead of a brute force search for finding the optimum splitting feature and value. As described in Section 3.6.2, the Brent optimization algorithm is used, which is activated when the amount of samples present is larger than the given threshold, $\theta_{optim.}$. For various values of $\theta_{optim.}$ a TBRF was constructed using 30 TBDT's. The square duct flow case was used for training and testing. This kept the amount of samples relatively low, which is necessary due to time constraints, since training the TBRF using a brute force search takes a considerable amount of time. Features used for the analysis came from FS1 and FS2, yielding a total of 11 usable features based on ∇U and ∇k.

One random forest was constructed using data for $Re = 3,200$ only, and one was constructed using four different Reynolds numbers, ranging from $Re = 2400$ to $Re = 3200$. As can be seen, no consistent patterns can be identified when increasing the value for $\theta_{optim.}$. When training only on $Re = 3,200$ it seems to be better to use a small value for $\theta_{optim.}$, and when training on the four different flow cases a optimum seems to be present around $\theta_{optim.} = 500$. It was decided to select a intermediate value for $\theta_{optim.}$ which performs okay in both cases of $\theta_{optim.} = 250$.

A visualization of the optimization process using the Brent's method is presented in Figure C.2. This is done for the first feature presented in Figure 3.4. The corresponding iterations of the algorithm are presented in Figure C.3. The starting values for the algorithm are defined by the bounds of feature 1, which ranges from $[\mathbf{X}]_1 = -2.21$ to $[\mathbf{X}]_1 = 0.80$. The first iteration

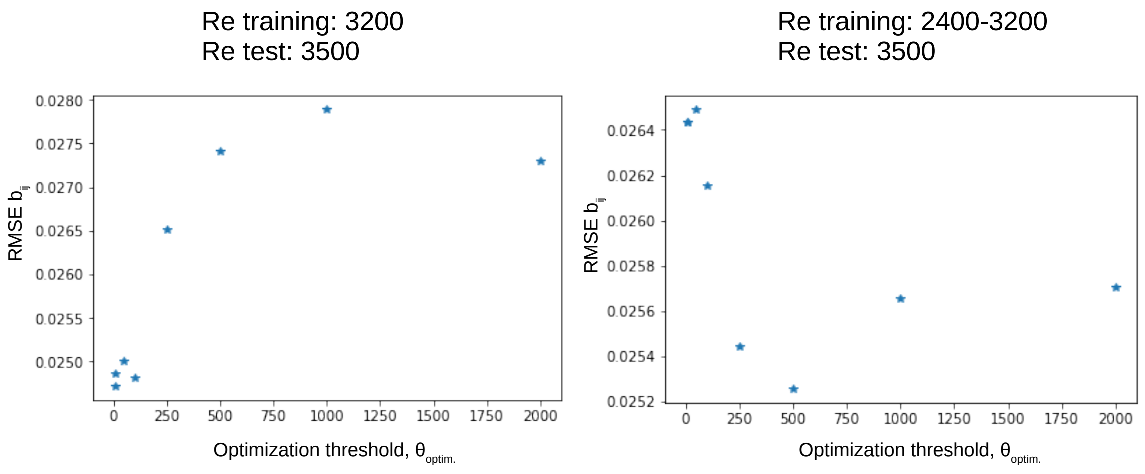


Figure C.1: Parameter study: influence of using the optimization algorithm at a given amount of samples ($\theta_{optim.}$) instead of a brute force search for finding the optimum splitting feature and value. Square duct flow case.

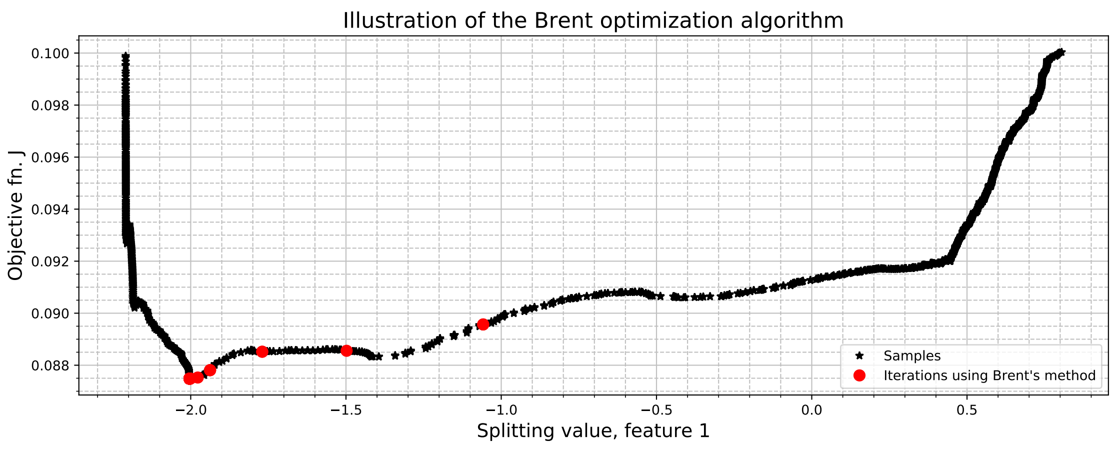


Figure C.2: Visualization of the optimization process by the Brent algorithm

```
Func-count      x          f(x)          Procedure
    1       -1.05815    0.0895679       initial
    2      -0.346606    0.0906332       golden
    3        -1.4979    0.0885551       golden
    4       -1.76969    0.0885191       golden
    5       -1.65548     0.088558       parabolic
    6       -1.93766    0.0878105       golden
    7       -2.04147    0.0885502       golden
    8       -1.90421    0.0881555       parabolic
    9       -1.97731    0.0875276       golden
   10       -2.00182    0.0874855       golden
   11       -1.99972    0.0874878       parabolic
   12       -2.01697    0.0880654       golden
   13       -2.00761    0.0878742       golden
   14       -2.00403    0.0875514       golden
   15       -2.00266    0.0874846       golden
```

Figure C.3: Iterations by the Brent algorithm corresponding to the case in Figure C.2

evaluates the function at two locations given by the golden ratio, resulting in $[\mathbf{X}]_1 = -1.06$ and $[\mathbf{X}]_1 = -0.35$. Since the function evaluation for $[\mathbf{X}]_1 = -0.35$ is larger, it is used as the new definition for the upper bound of the golden section search, after which this process is repeated, resulting in a new evaluation at $[\mathbf{X}]_1 = -1.50$. The golden sections search is continued, in combination with inverse quadratic interpolation in an attempt to speed up the convergence of the optimization process. The attempts of the algorithm to use inverse quadratic interpolation do not work out in this case, making it fall back on the more robust golden section search. Eventually the algorithm converges to its minimum where $J = 0.0874$ at $[\mathbf{X}]_1 = -2.00$.

Appendix D

Influence of the TBRF hyperparameters

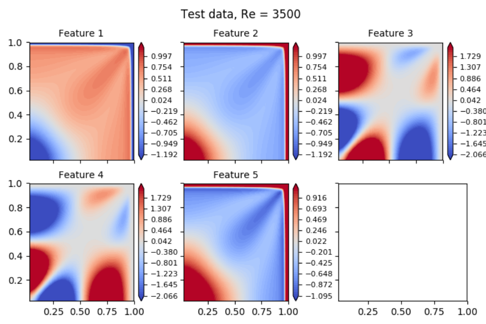
In this appendix results for the preliminary study regarding the influence of various hyperparameters of the TBRF algorithm will be presented. First of all, the influence of including different features will be analysed in Section D.1. Afterwards, the effect of the filters used t process the results will be discussed in Section D.2. Finally the effect of regularization will be discussed in Section D.3.

D.1 Amount of features

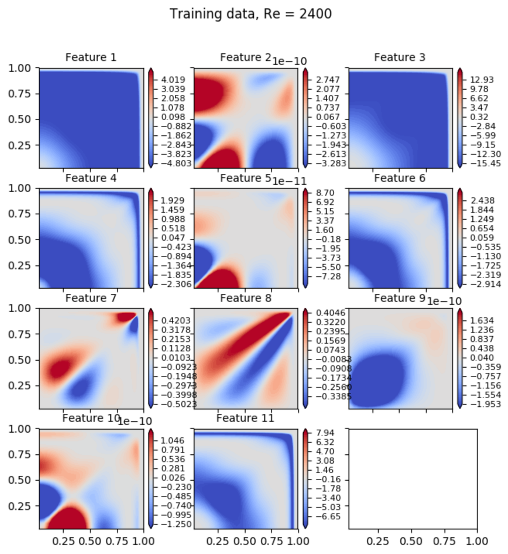
In a preliminary study it was found that only using the 5 invariant features based on S_{ij} and Ω_{ij}, as done in Ling et al. (2016b), would not be sufficient to make good predictions for the anisotropy tensor. From these findings, it was decided to introduce more features, first by also introducing invariants based on ∇k, and later by also using physically interpretable features from Wu et al. (2016).

Features based on S_{ij}, Ω_{ij}, and ∇k are presented in Figure D.1 for the square duct flow case. Figure D.1a presents the 5 different features based on S_{ij} and Ω_{ij} only. As can be seen, effectively there are only 2 or 3 different features present in this data-set. Feature 1 and 2 mirror each other, and feature 3 and 4 mirror each other. While feature 5 is slightly different from feature 2, the differences are minimal. When one would want to discern two separate points in the square duct flow using these 5 features, it would be quite difficult when looking for example at the area in the middle of the domain. Feature 3 and 4 are almost zero in this region. While feature 1,2, and 5 show some small gradients, they are symmetric around the diagonal ranging from $(y, z) = (0.0, 0.0)$ to $(y, z) = (1.0, 1.0)$. As an example, it would be difficult for the machine learning algorithm to discern a point at $(y, z) = (0.8, 0.5)$ from a point at $(y, z) = (0.5, 0.5)$, since feature 1, 2, and 5 have approximately similar magnitudes at both locations, and feature 3 and 4 are almost zero. When training the TBDT this could mean that the responses from both of these locations, which could be significantly different, end up in the same leaf node of the TBDT.

Introducing more features allows the algorithm to more easily discern responses in different areas of the flow. Figure D.1b presents the features which can be used when also introducing the gradient of the turbulent kinetic energy, see Table 3.1. The duplicate features from Figure D.1a have been removed. As can be seen some of the features could be helpful in separating the response around the middle of the domain. For example, feature 8 can be useful to separate a response on the lower right of the diagonal from a response on the upper left of the diagonal.



(a) 5 features, based on S_{ij} and Ω_{ij}

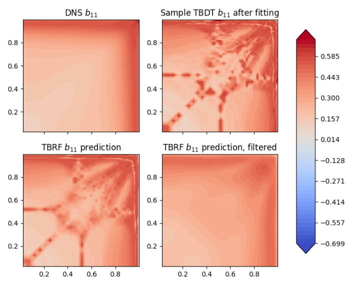


(b) 11 features, based on S_{ij}, Ω_{ij}, and ∇k

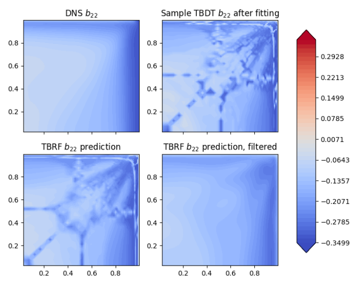
Figure D.1: Comparison between the features based on S_{ij} and Ω_{ij} only, and those when also introducing ∇k

A TBRF was trained on the square duct, using four data-sets ranging from Re = 2,400 to Re = 3,200; after which it was tested on a square duct at Re = 3,500. Results for three different components of the anisotropy tensor are presented in Figure D.2. For each prediction of the anisotropy component, the upper left plot presents the DNS data, and the upper right plot presents one sample TBDT prediction from the TBRF. The mean over all the TBDT's in the TBRF is plotted on the lower left figure. Finally the predictions filtered with a Gaussian filter are presented in the lower right figures.

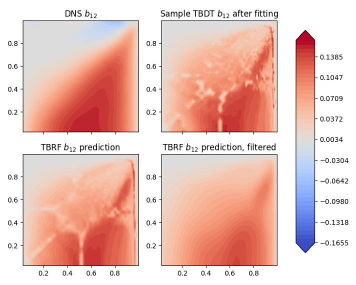
As can be seen from these figures, the algorithm has a lot of difficulty when predicting b_{ij} using only the five features based on S_{ij} and Ω_{ij}, see Figure D.2a to Figure D.2c. Along the diagonal from the lower left to the upper right corner of the duct a lot of noise can be identified. A lot of noise is also present along the horizontal and vertical lines in the middle of the domain. When inspecting Figure D.1a, one can see that these locations of noise show good correspondence to the locations at which feature 4 and feature 5 are close to zero. As explained earlier, this makes it difficult for the algorithm to discern points from each other, since the other three features are also symmetric around the diagonal ranging from the lower left to the upper right corner. When introducing the extra features based on ∇k, predictions in the domain become less noisy on average. Results are presented in Figure D.2d to Figure D.2f. As can be seen predictions are quite accurate, except from two spots around $(y, z) = (0.8, 0.5)$ and $(y, z) = (0.5, 0.8)$. Most likely some of the decision trees are still not able to properly discern data around these spots with data at the corner of the duct, where predictions are off as well. The basis tensors close to the wall are significantly different from those near the middle of the domain due to the scaling with $1/\omega$. When there are not enough discriminative features, this can mean the TBRF will give predictions for the basis tensor coefficients $g^{(m)}$ in the middle of the domain, which were obtained from responses close to the wall. In this case the predicted magnitude for $g^{(m)}$ will be too large, as well as the magnitude of the components of b_{ij}, which is exactly what is observed in Figure D.2d and Figure D.2e. As can be seen, applying a Gaussian filter does not work properly to remove the outliers, since the discrepancies are too large. This is the main reason the median filter was implemented, as presented in Section 3.8.



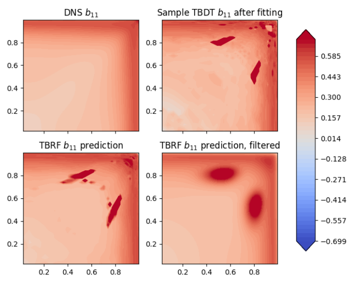
(a) b_{11}, 5 features



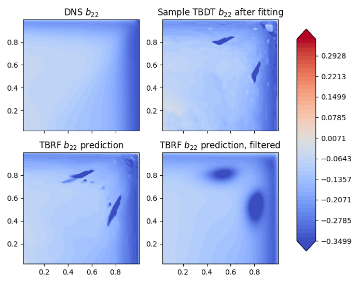
(b) b_{22}, 5 features



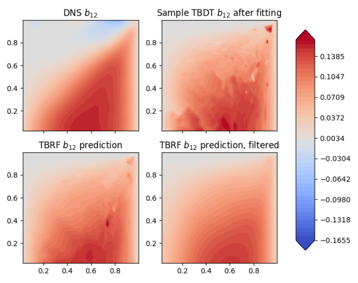
(c) b_{12}, 5 features



(d) b_{11}, 11 features



(e) b_{22}, 11 features

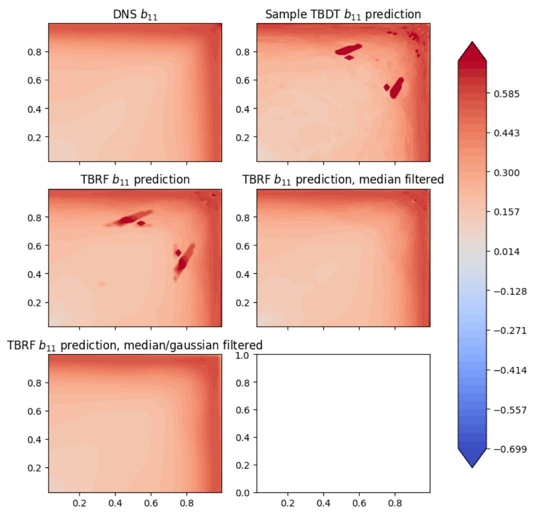


(f) b_{12}, 11 features

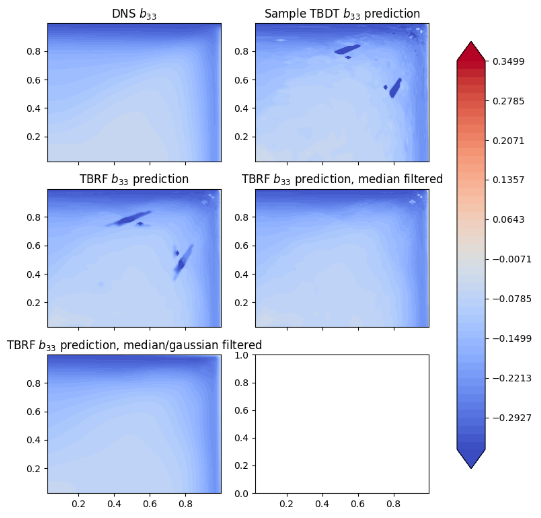
Figure D.2: Comparison between using 5 features for predicting b_{ij}, and expanding the amount of features to 11

D.2 Filtering

In order to demonstrate the effectiveness of the implemented filters, the results as given by a TBRF trained on the 11 features given in Figure D.1b are post-processed. Again the TBRF is trained on 4 different data-sets of the square duct ($Re = 2,400$ to $Re = 3,200$) and tested on the square duct at $Re = 3,500$. Results are presented in Figure D.3 for the b_{11} and b_{33} components of the anisotropy tensor. In the figures, the following quantities are presented, left to right, top to bottom: the DNS reference data, one sample TBDT prediction from the TBRF, the TBRF prediction when no filters would be used (i.e. the average over all the TBDT's), the prediction of the TBRF after applying the median filter ($\theta_{med.} = 3$), and the TBRF prediction after applying the median filter and applying a Gaussian filter to smoothen the results spatially. As can be seen, the results after applying the median filter show great correspondence to the DNS data. Many of the TBDT's have trouble predicting b_{ij} at the aforementioned locations at approximately $(y, z) = (0.8, 0.5)$ and $(y, z) = (0.5, 0.8)$, which can be seen in the sample TBDT prediction, and the TBRF predictions without using any filters. However, different TBDT's will give these discrepancies at slightly different locations in the flow field. As long as the majority of TBDT's give a good prediction at a given location on the mesh, the filtered result will be good as well. After filtering out the high frequency noise in the domain by using the Gaussian filter, the predictions look almost identical to the DNS data. The improvements can be quantified in terms of the RMSE of b_{ij}: the TBRF without any filtering gives a RMSE of 0.050, after applying the median filter it reduces to 0.035, and after applying the Gaussian filter it reduces further to 0.029.



(a) b_{11}

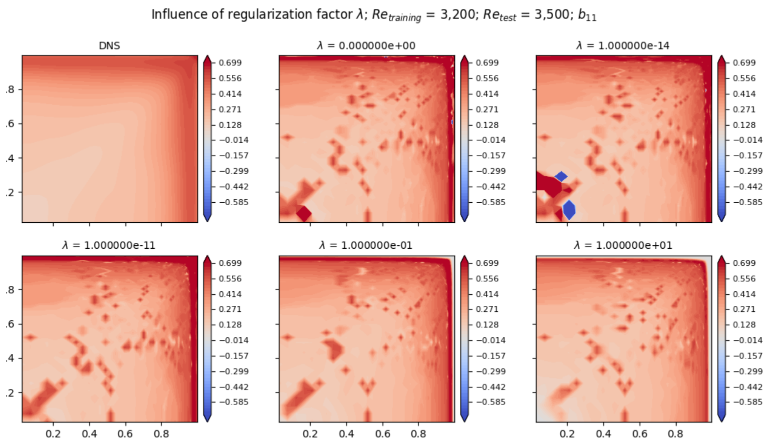


(b) b_{33}

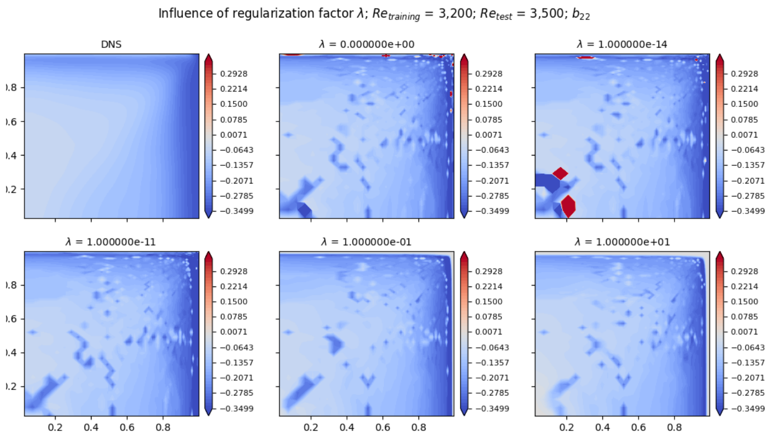
Figure D.3: Demonstration of the median filter on the TBRF results of the square duct presented in Figure D.2 for 11 features

D.3 Regularization

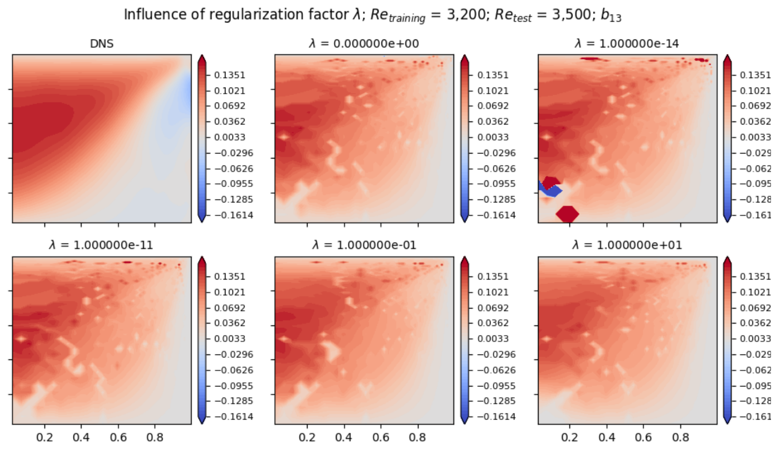
In Figure D.4, the influence of the regularization parameter λ on TBDT predictions is presented, for the b_{11}, b_{22}, and b_{12} components of the anisotropy tensor for the square duct. The upper left plot presents the DNS data. Then, from left to right, top to bottom, results are presented for $\lambda = 0$, $1{\times}10^{-14}$, $1{\times}10^{-11}$, 0.1, and 10. The TBDT's were trained on four different square duct flow cases, ranging from $Re = 2,400$ to $Re = 3,200$, and using the five features based on S_{ij} and Ω_{ij}. As can be seen, all flow cases show a significant amount of noise. The magnitude of this noise decreases when increasing λ, although it does not disappear entirely. When using no regularization, or just a very small amount ($1{\times}10^{-14}$), some large discrepancies can be observed near the centerline of the duct which are completely out of the bounds specified by the realizability conditions. When increasing λ up to a certain level, which is in this case $1{\times}10^{-11}$, these large discrepancies do not show up any more. What can also be observed however, is that predictions close to the wall get inaccurate when λ increases too much. Close to the wall, the tensor basis coefficients $g^{(m)}$ should be able to attain large magnitudes compared to the tensor basis coefficients further away from the wall. This is due to the fact that the basis tensors are scaled by $1/\omega$, see the discussion in Section 8.2. When introducing a regularization factor which is too large, the tensor basis coefficients are therefore not allowed to vary enough in able to represent the flow closer to the walls. In this case, a regularization factor which is big enough to get rid of the major discrepancies, and which is small enough to not have too much influence on predictions close to the wall is required, for which something around $1{\times}10^{-11}$ would be a good candidate when looking at the results.



(a) b_{11}



(b) b_{22}

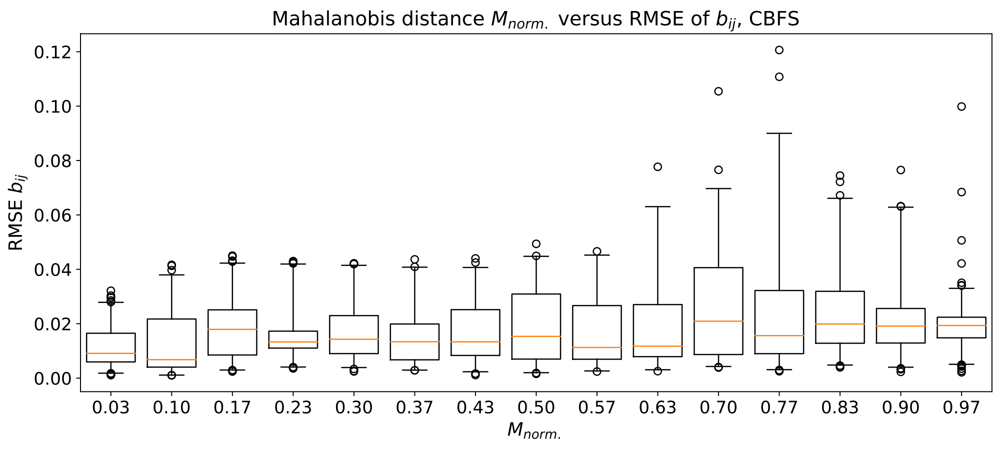


(c) b_{13}

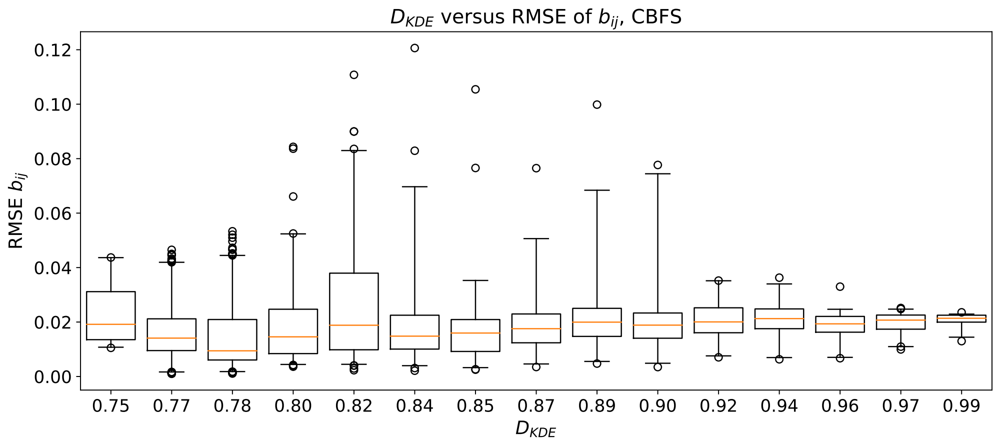
Figure D.4: Demonstration of the effect of including regularization

Appendix E

Correlation between the a priori confidence parameters and the error of the anisotropy tensor

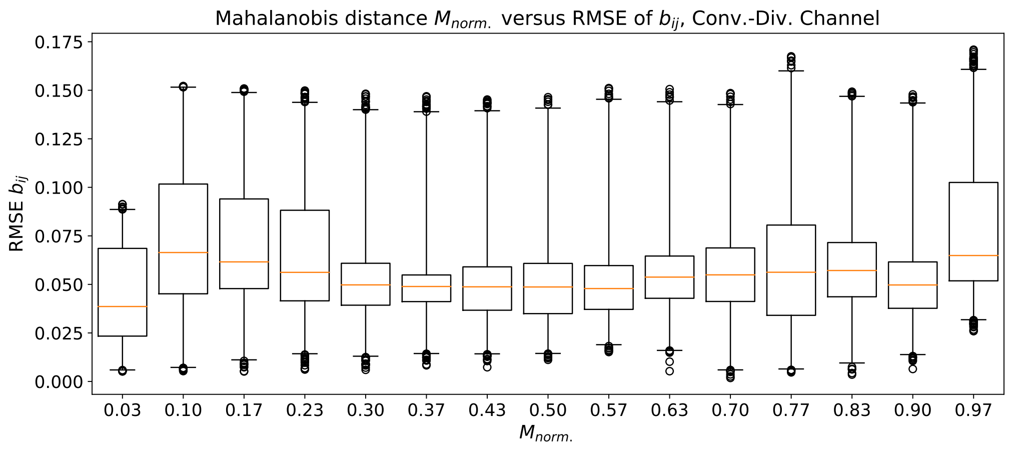


(a) $M_{norm.}$

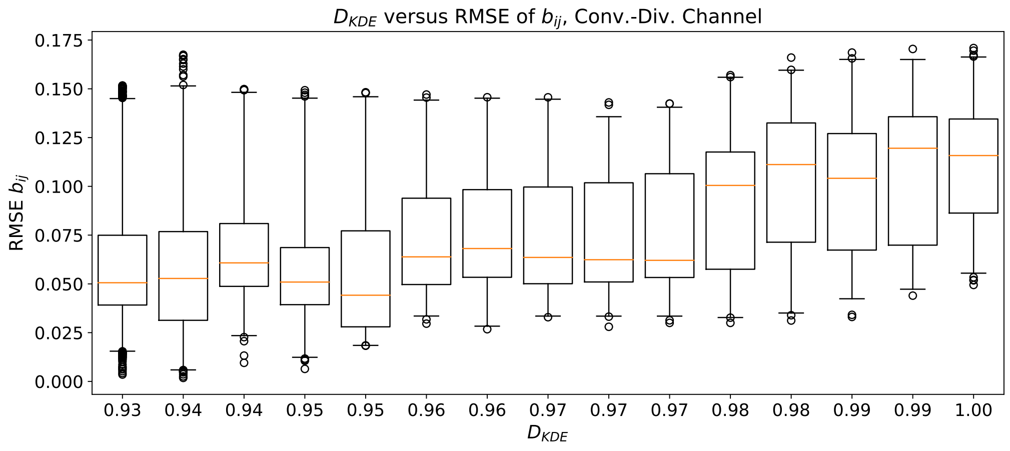


(b) D_{KDE}

Figure E.1: Correlation study, curved backward facing step: boxplots of the a priori confidence parameters versus the RMSE of b_{ij}



(a) $M_{norm.}$



(b) D_{KDE}

Figure E.2: Correlation study, converging-diverging channel: boxplots of the a priori confidence parameters versus the RMSE of b_{ij}

Appendix F

Visualization of the features

In this appendix the features in the flow domain of the different flow cases will be visualized. The definition of the features is found in Table 3.1. All features in the flow domain are visualized in Figure F.1. From the total set of features, the ones with a low variance ($<1\times10^{-4}$) were removed, since these either did not contain any information at all, or showed spurious behaviour, such as FS1,3, FS1,4, and FS2,11. The leftover features for the periodic hills are presented in Figure F.2. The same features for the converging-diverging channel, curved backward facing step, backward facing step, and square duct are presented in Figure F.3, Figure F.4, Figure F.5, and Figure F.6 respectively.

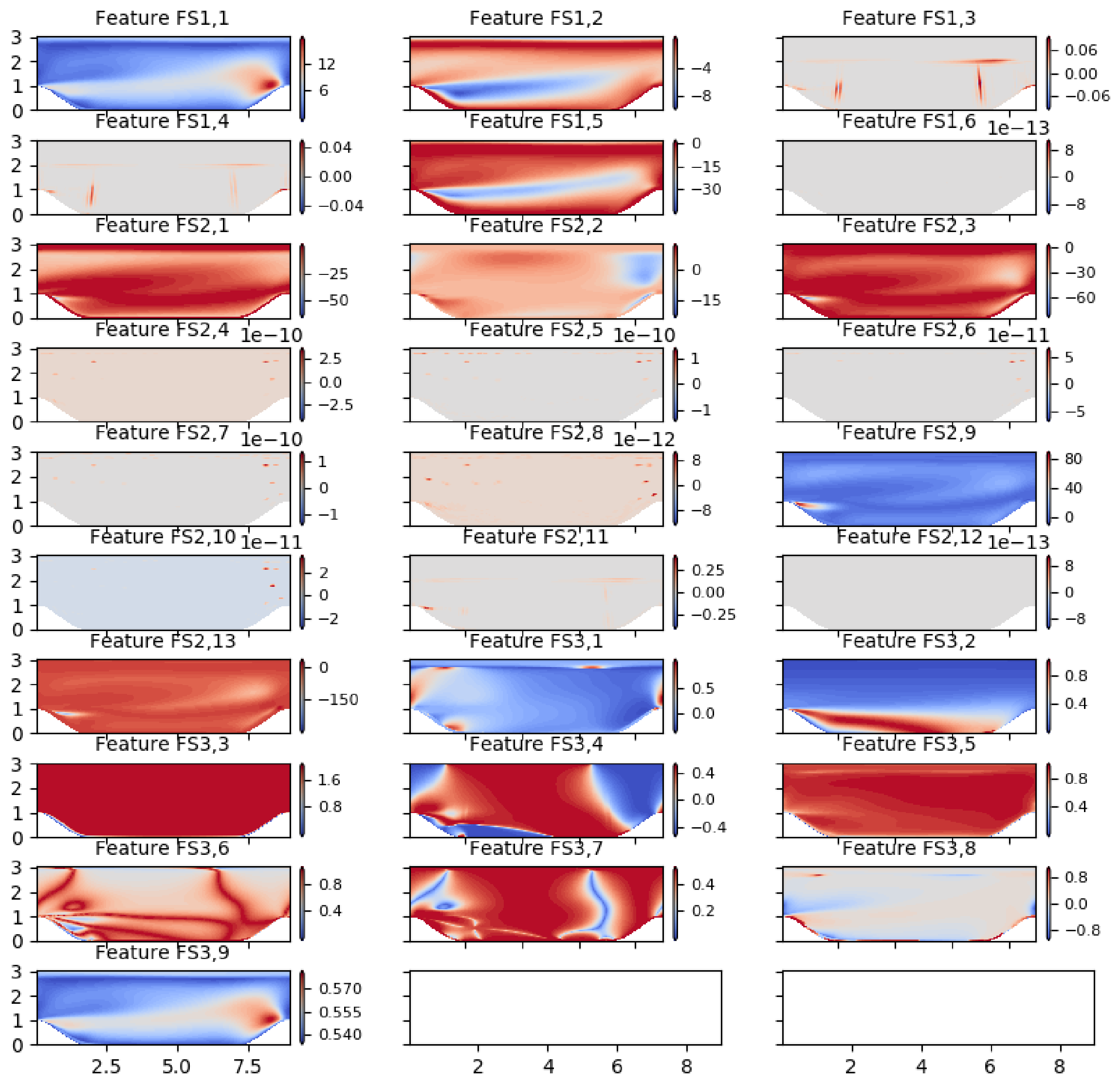


Figure F.1: All features in the flow domain of the periodic hills ($Re = 10,595$)

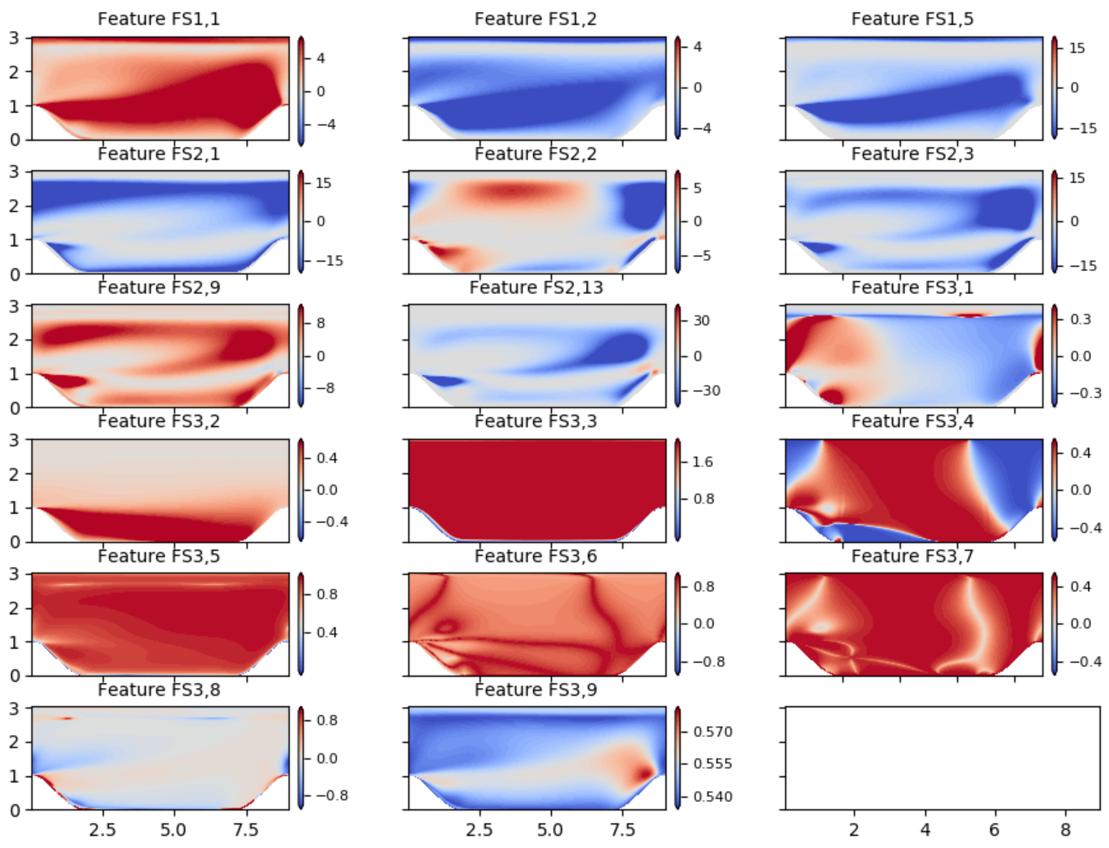


Figure F.2: Features in the flow domain of the periodic hills ($Re = 10,595$)

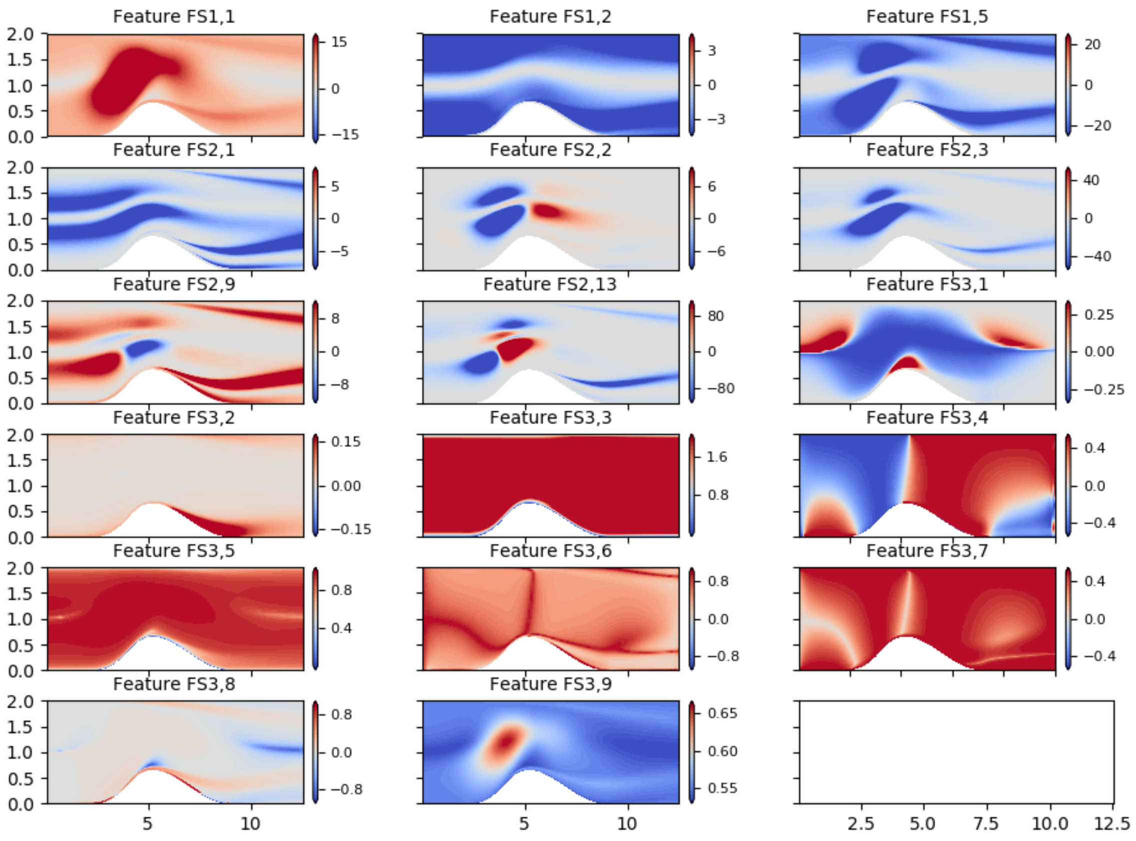


Figure F.3: Features in the flow domain of the converging-diverging channel

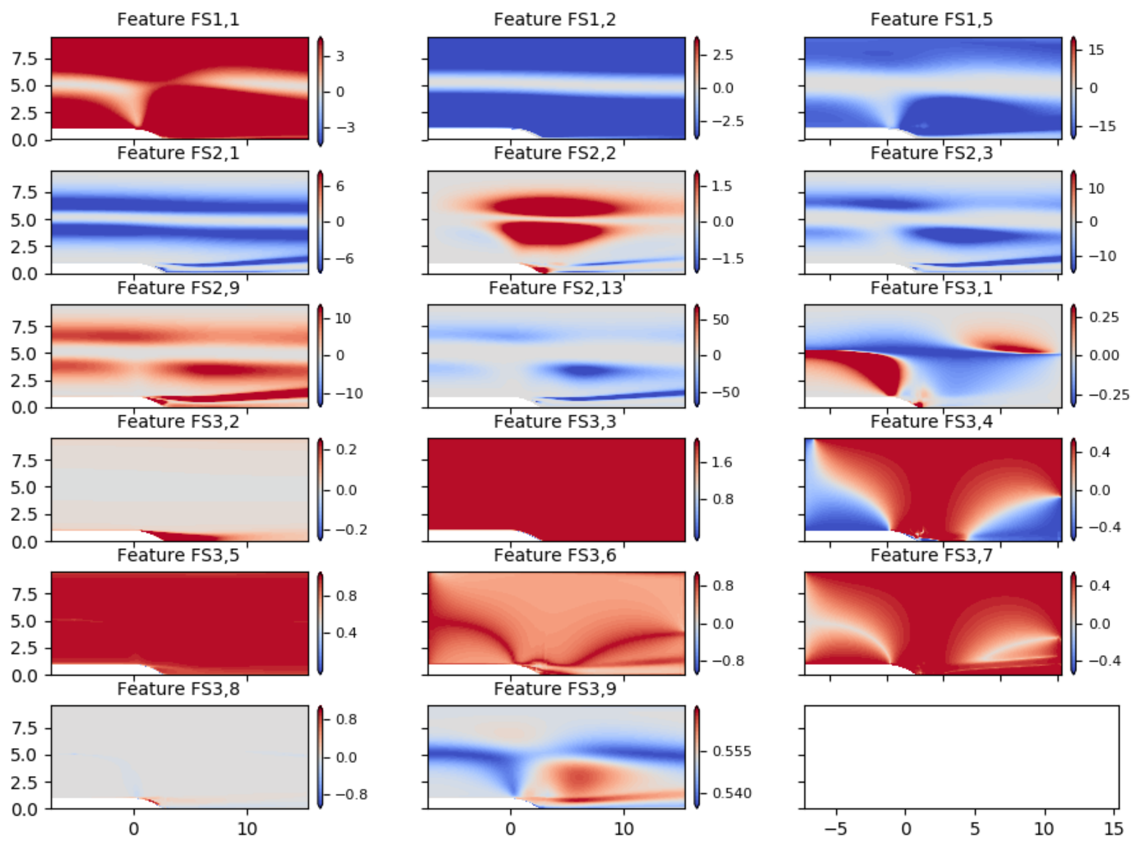


Figure F.4: Features in the flow domain of the curved backward facing step

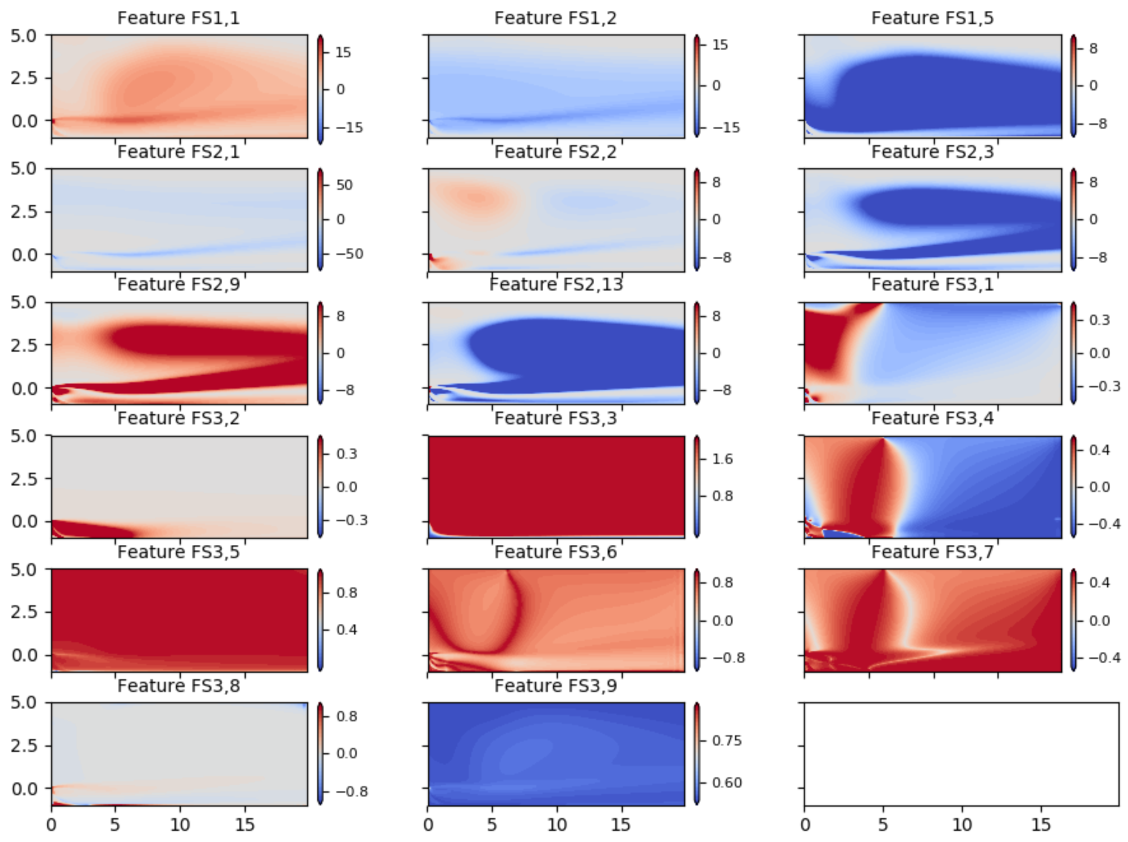


Figure F.5: Features in the flow domain of the backward facing step

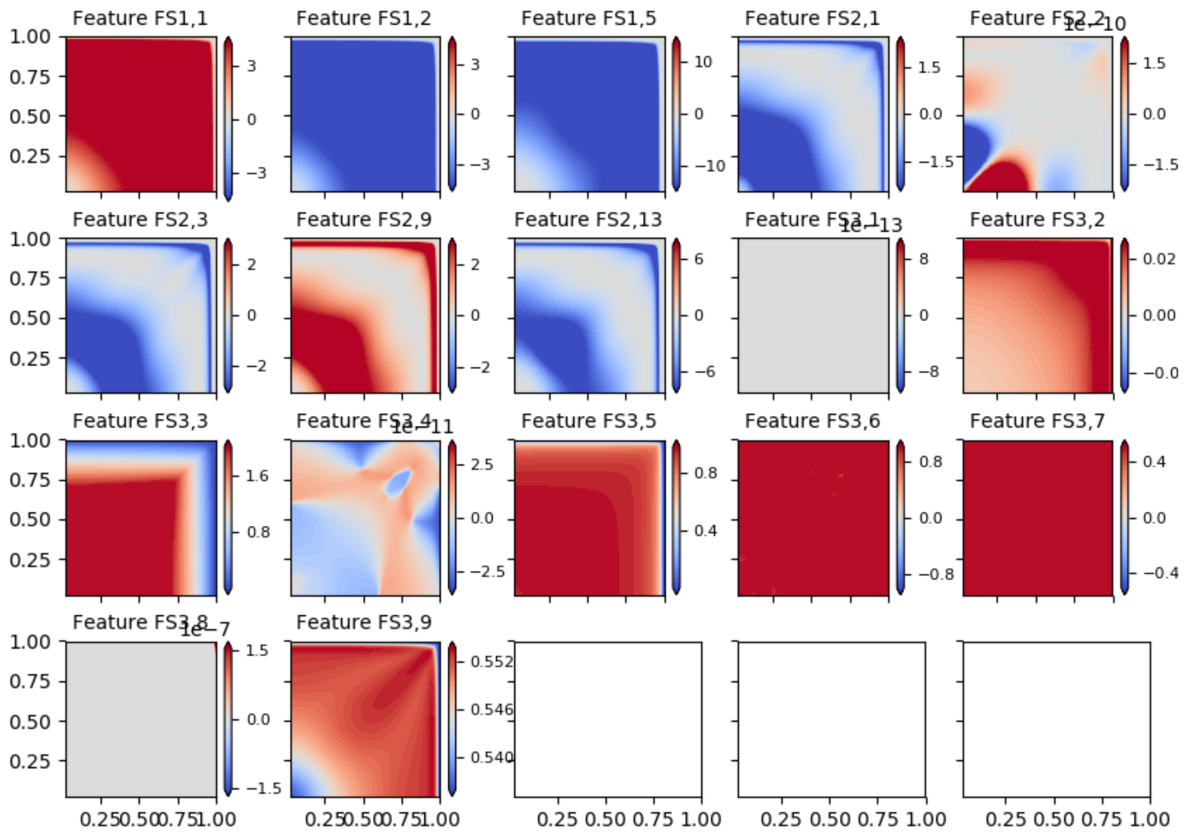


Figure F.6: Features in the flow domain of the square duct $(Re = 3,500)$

Appendix G

Visualization of the Basis Tensors

In this appendix the basis tensors for the square duct and curved backward facing step will be presented, in order to clarify the results which were presented in Section 8.2. The upper rows will present the diagonal components of the tensors, the lower rows the off-diagonal components.

G.1 Square Duct

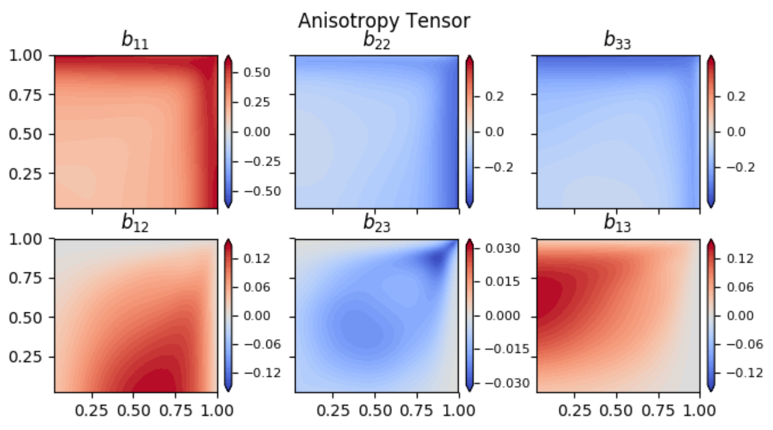
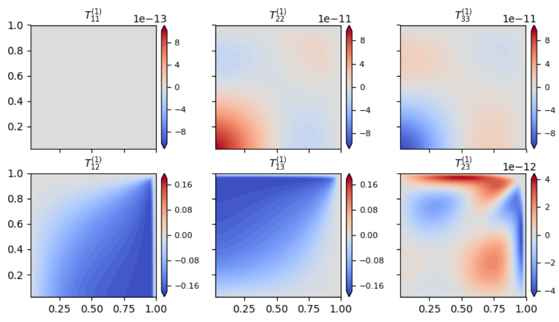
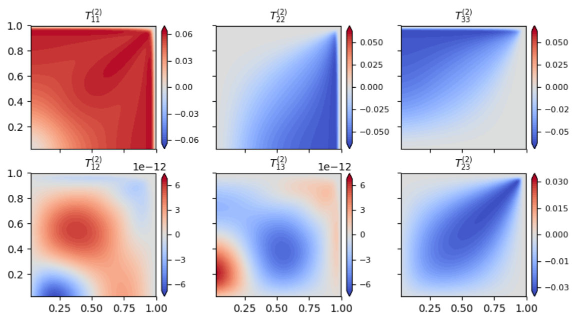


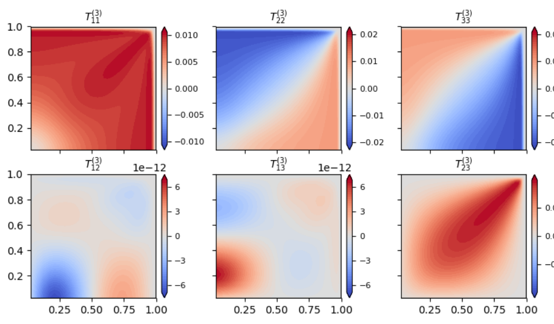
Figure G.1: b_{ij} corresponding to the tensor basis coefficients in Figure 8.4b



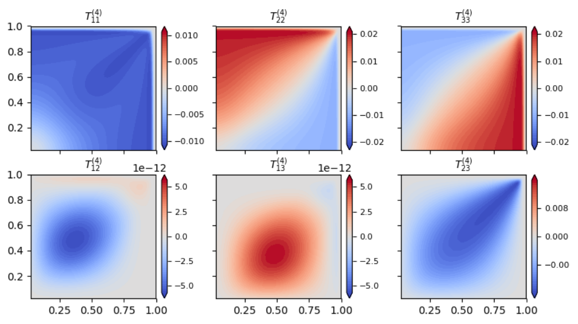
(a) $T^{(1)}$



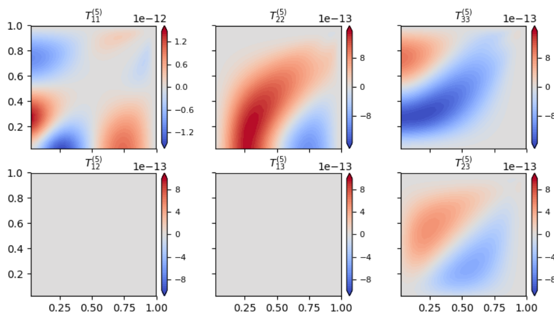
(b) $T^{(2)}$



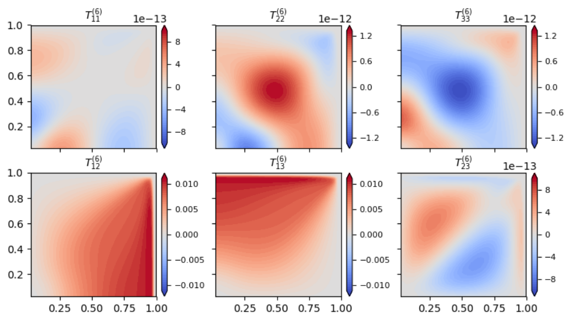
(c) $T^{(3)}$



(d) $T^{(4)}$

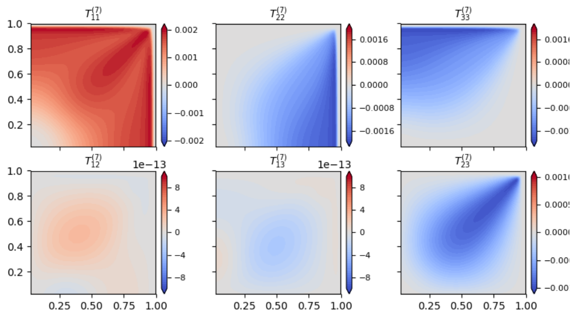


(e) $T^{(5)}$

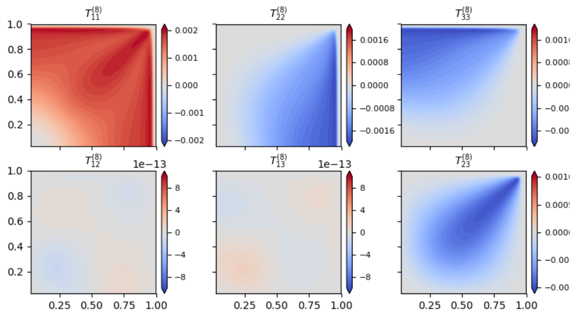


(f) $T^{(6)}$

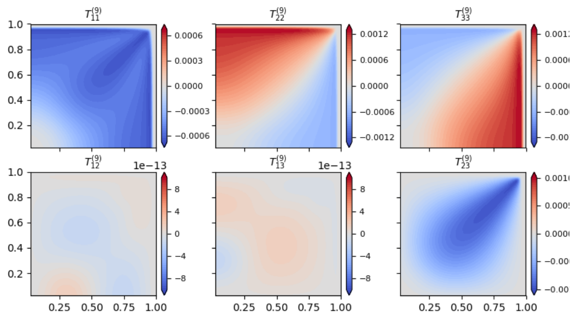
Figure G.2: Square duct, basis tensors $T^{(1)}$ to $T^{(6)}$. Left-to-right, top-to-bottom: $T^{(m)}_{11}$, $T^{(m)}_{22}$, $T^{(m)}_{33}$, $T^{(m)}_{12}$, $T^{(m)}_{13}$, $T^{(m)}_{23}$.



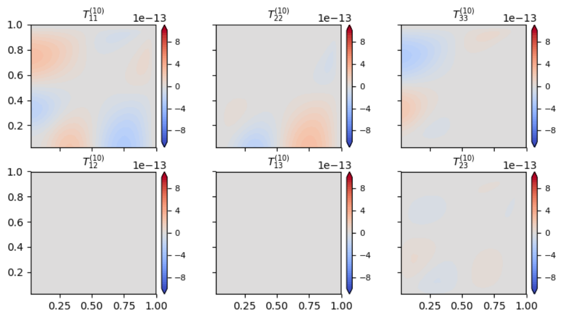
(a) $T^{(7)}$



(b) $T^{(8)}$



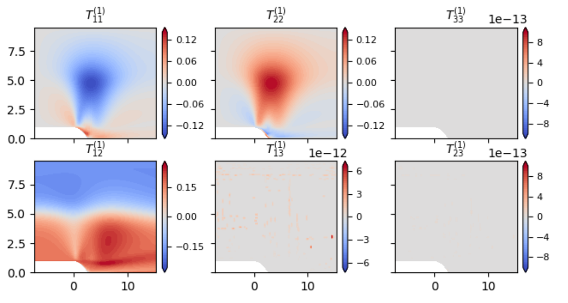
(c) $T^{(9)}$



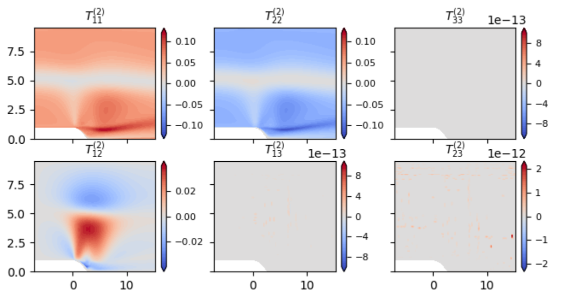
(d) $T^{(10)}$

Figure G.3: Square duct, basis tensors $T^{(7)}$ to $T^{(10)}$. Left-to-right, top-to-bottom: $T_{11}^{(m)}$, $T_{22}^{(m)}$, $T_{33}^{(m)}$, $T_{12}^{(m)}$, $T_{13}^{(m)}$, $T_{23}^{(m)}$.

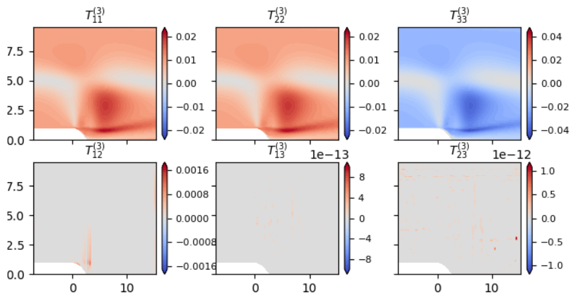
G.2 Curved Backward Facing Step



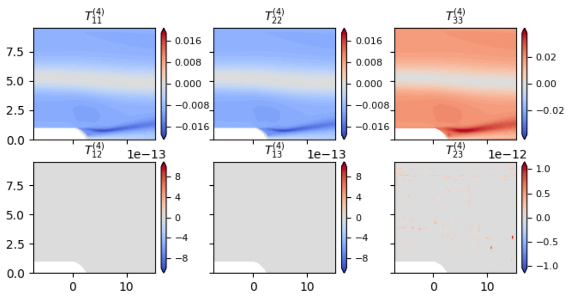
(a) $T^{(1)}$



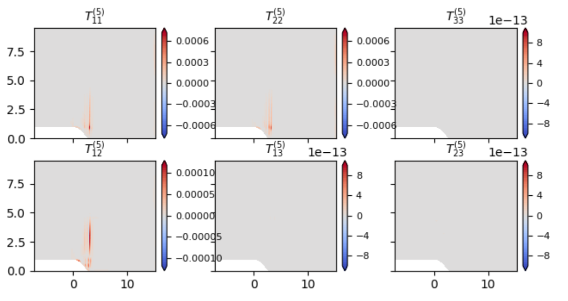
(b) $T^{(2)}$



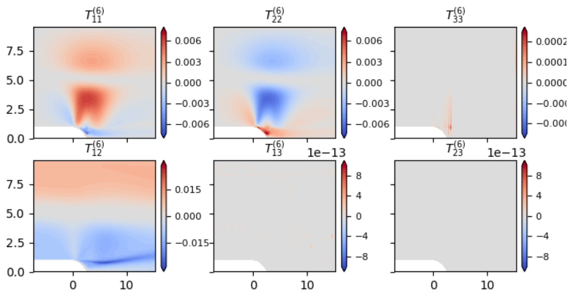
(c) $T^{(3)}$



(d) $T^{(4)}$

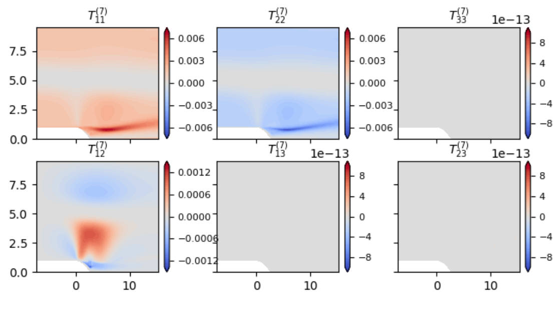


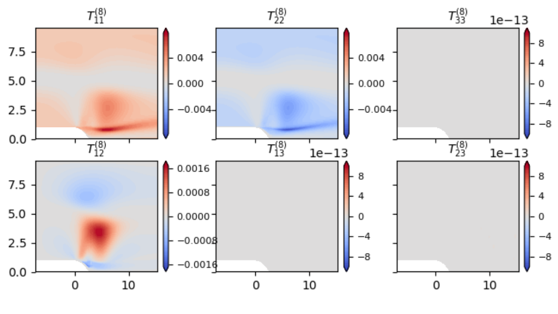
(e) $T^{(5)}$

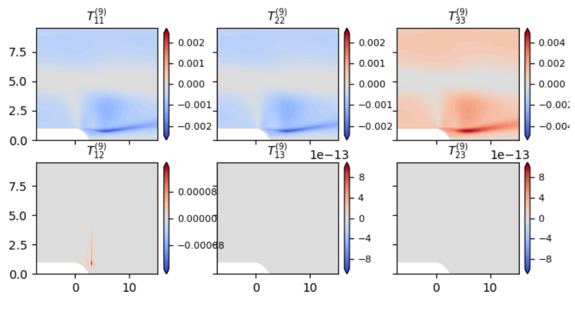


(f) $T^{(6)}$

Figure G.4: Curved backward facing step, basis tensors $T^{(1)}$ to $T^{(6)}$. Left-to-right, top-to-bottom: $T_{11}^{(m)}$, $T_{22}^{(m)}$, $T_{33}^{(m)}$, $T_{12}^{(m)}$, $T_{13}^{(m)}$, $T_{23}^{(m)}$.


(a) $T^{(7)}$


(b) $T^{(8)}$


(c) $T^{(9)}$

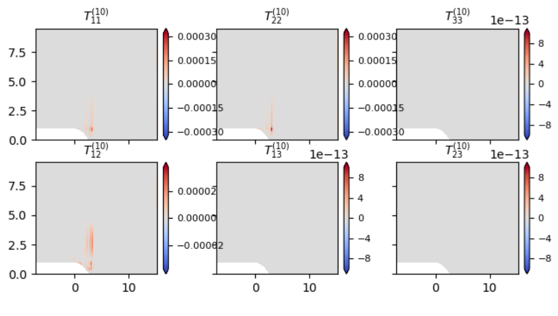

(d) $T^{(10)}$

Figure G.5: Curved backward facing step, basis tensors $T^{(7)}$ to $T^{(10)}$. Left-to-right, top-to-bottom: $T^{(m)}_{11}$, $T^{(m)}_{22}$, $T^{(m)}_{33}$, $T^{(m)}_{12}$, $T^{(m)}_{13}$, $T^{(m)}_{23}$.

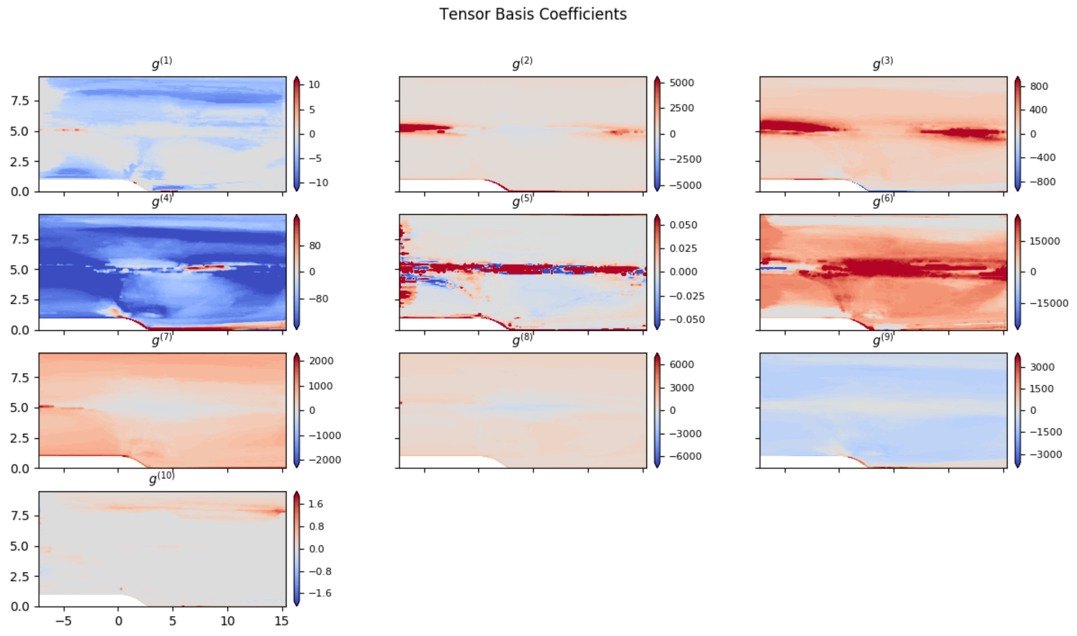


Figure G.6: Tensor basis coefficients $g^{(m)}$ for the curved backward facing step