Implementation and assessment of subgrid-scale models for large eddy simulations of incompressible turbulent flows

by

JOAKIM BØ

THESIS

for the degree of

Master of Science

(Master i anvendt matematikk og mekanikk)

Faculty of Mathematics and Natural Sciences
University of Oslo

November 2015

Det matematisk-naturvitenskapelige fakultet
Universitetet i Oslo

Abstract

The aim of this work has been to implement a range of Large Eddy Simulation (LES) turbulence models into Oasis, a Computational Fluid Dynamic (CFD) solver for incompressible flows based on the Finite Element method (FEM), developed in-house at the University of Oslo. The focus has been on subgrid-scale models that apply the eddy viscosity hypothesis to close the equation set, where both static and dynamic types have been investigated. Verification and assessment of the implementation is performed applying the Method of Manufactured Solutions (MMS), and the classic case of fully developed turbulent channel flow in an $x - z$ periodic channel. The work is further validated through the U.S. Food and Drug Administration (USFDA/FDA)'s computational round robin #1.

MMS does in general return good results, where the convergence rate in time is correctly computed to $r = 2$ for some constructed eddy viscosity expressions, the Smagorinsky model, and the WALE model. As for the channel flow case, the Smagorinsky model returns mean velocity profiles that are closer to those of resolved Direct Numerical Simulations (DNS), compared to what is obtained with under-resolved DNS. The WALE model, the Sigma model, and the Dynamic Smagorinsky model, which all have the correct wall behaviour, return results where little or no improvements are seen compared to the under-resolved profiles. It turns out that a net contribution of eddy viscosity close to the wall is extremely important for this specific case, as this in some sense controls the whole quality of the simulation. The Sigma model and the Dynamic Smagorinsky model do, in despite of small improvements to the mean velocities, return good profiles for some selected Reynolds stresses.

For the FDA case good results are obtained for both uniform and non-uniform meshes for all LES models, where the mean velocity profiles in general are between 50 to 80 percent closer to experimental data, compared to what is obtained with under-resolved DNS. The only model that, as expected, erroneously predicts the breaking position of the jet is the Smagorinsky model. The best validation metric is obtained with the WALE model on the non-uniform mesh of three million cells; on the other hand, the best percentage improvement over under-resolved DNS is obtained for the Dynamic Smagorinsky model with Lagrangian averaging on the non-uniform mesh of two million cells.

Acknowledgements

First, I would like to thank the Department of Mathematics, and especially the section for Mechanics, at the University of Oslo (UiO) for the opportunity to carry out this master's thesis.

A very special thank you goes to my supervisor Mikael Mortensen, for his enthusiasm and interest in the field of CFD and turbulence, and for all the supervising and mail answering, he has done. Without your help, theoretical input, and motivational aura, this master's thesis would not have existed today. Thank you also for pushing me onward, both when it comes to this master's thesis, and for the conference contribution we had together at MekIT'15 in May.

Thank you to both my parents Asle and Martha for always being there for me, encouraging me to follow my own heart and dreams at all times. I would certainly not have been the person I am today without you. I would also like to thank both my brother Sondre and my sister Veslemøy for being a part of my life.

Thank you to my parents-in-law, Stine, Sven, Dag, and Ruth, for your great support. Also thank you to my fantastic brother-in-law Isak for being a great friend, and brother.

I would also like to send a special thank you to prof. Hans Petter Langtangen at the Department of Informatics at UiO for his great support. You are an excellent motivator, lecturer, and guitar player; thank you for sharing your vast knowledge and enthusiasm with all the students who surround you. Thank you to prof. Atle Jensen at the section of Mechanics for his motivational support. For cooperation, case files, help, and fruitful discussion regarding the FDA-case, thank you to my fellow student Aslak Bergersen. Also thank you to my good friend Jørgen Høgberget for his support.

Finally yet importantly, thank you to my beloved wife Tiril for always being there for me. Thank you for all the lovely weekdays, weekends and evenings we have together; thank you for encouraging me and pushing me onwards when motivation has been scarce; thank you for always believing in me, and thank you for the wonderful wedding day we had together in June. You are the best wife a man could wish for, and have. I love you of all my heart.

Joakim Bø
Oslo, November 2015

Contents

Chapter 1

Introduction

Understanding turbulent flows, how they behave, and why they behave the way they do, what they actually are, and when they transpire, are questions that have fascinated, amazed, dazzled, and riddled scientists for many centuries. Is turbulence totally random and stochastic, or is it possible to describe using only deterministic equations? Perhaps is it just an utter chaos of extreme complexity, so intricate and involved that we may not simply understand or predict the phenomenon, nor any of its processes? Though many of the aspects regarding turbulence have been solved, refined, and developed over the last century, there are still many ingredients that remain unsolved, greatly troubling the minds of today's scientists.

The famous Italian polymath Leonardo da Vinci did, in between painting sessions, also have an interest in observing, understanding, and sketching turbulence (see Figs. 1.1 and 1.2); or "la turbolenza", as he named it. In his search of understanding the phenomenon, he wrote in his diaries [1]:

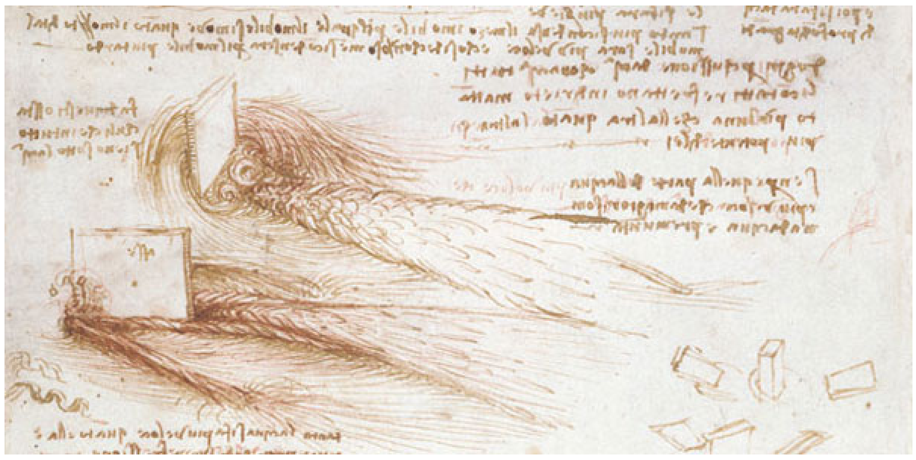


Figure 1.1: A rigid obstacle in flowing water, as sketched by Leonardo da Vinci in 1508-1509 (Nature).

"Observe the motion of the surface of the water, which resembles that of hair, which has two motions, of which one is caused by the weight of the hair, the other by the direction of the curls; thus the water has eddying motions, one part of which is due to the principal current, the other to random and reverse motion."

In other words, da Vinci noticed that the turbulent flow could be described by two parts. It was not until the year of 1895, nearly 400 years after da Vinci, that the British scientist Osborne Reynolds would develop this idea further by introducing the field of turbulence modelling.

In the early 19th century the British scientist George Gabriel Stokes and the French scientist Claude-Louis Navier did, independently of each other, develop the so-called Navier-Stokes equations; four partial differential equations completely describing the flow of a viscous fluid in terms of velocity and pressure. Later, in 1895, Reynolds [2] postulated that the same velocity and pressure fields could be split into two parts: one describing the mean parameters of the flow, the other describing the flows random, or fluctuating, behaviour. When these decompositions are coupled with the Navier-Stokes equations, one eventually arrives at a new equation set that describes the flow in terms of a mean velocity and a mean pressure. These Reynolds-Averaged Navier-Stokes (RANS) equations, in addition to being recast into equations for the mean fields, contain six new, unknown terms, represented by the fluctuating velocity components. To close the equation set models, which correctly computes these unknown components,

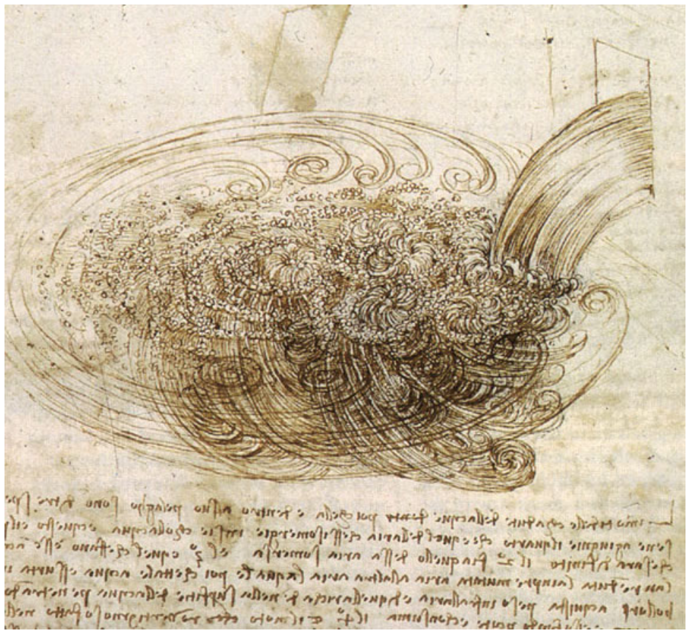


Figure 1.2: A free water jet flowing through a square hole into a pool, as sketched by Leonardo da Vinci in 1508-1509 (Wikimedia Commons).

must be invoked, such that the RANS equations may be solvable for their mean quantities.

Leaving the 19th century, instead turning towards the computer age and the present time, the field of Computational Fluid Dynamics (CFD), where the Navier-Stokes equations are numerically solved, has, as a result of the massive leap in computational power and resources seen the last decades, become an extremely important tool for analysing flow phenomena of all types and settings. Though CFD has become more and more popular, the traditional method of Experimental Fluid Dynamics (EFD) is, and will continue to be, an even more important branch of fluid mechanics. Installations such as e.g. wind tunnels, wave tanks or pipe systems are coupled with lasers, high-speed cameras or similar, such that complex flows may be experimentally studied. However, in despite of experimental work being the traditional and more reliable approach, it is being more and more combined with CFD and numerical experiments such that the pros and cons from each field may be combined into one good package.

In despite of the huge leap seen in the computational department the last decades, so-called Direct Numerical Simulations (DNS), where one fully resolves and simulates all the scales in the flow, is still, and will continue to be, practically impossible. Even for those who have extremely powerful super computers at hand, DNS is limited to flows of moderate Reynolds numbers only (see section 2.1), as the total amount of required computational time simply becomes so vast, that one may start to measure it in decades, centuries, or perhaps even millennia. This DNS problem has eventually resulted in turbulence modelling becoming a crucial tool for analysing complex and chaotic flows, where RANS methods, because of their efficiency and maturity, are very much the industry standard today.

As an alternative to RANS a slightly different approach was developed in the 1960s, then for use with simulations of atmospheric air currents [3]. The basic idea in so-called Large Eddy Simulation (LES) is to abandon Reynolds' mean variables, and instead work with filtered velocity and pressure fields. Decomposition of the velocity and pressure fields is still done similarly to what was done by Reynolds, the difference being that the fields are split into a filtered and a residual part. The original Navier-Stokes equations are then transformed into a set of equations for the filtered velocity and pressure, where again, as with RANS, a new term containing six unknowns is introduced into the equation set. For the equations to be uniquely solvable for the filtered velocity and pressure, LES turbulence models, which model all the effects caused by the unknown residual velocities, must be introduced.

In terms of computational requirements, LES positions itself somewhere between RANS and DNS. It is more feasible than DNS since the smallest fluctuating scales are modelled, and not directly computed; on the other hand, it is more demanding than RANS, since the largest fluctuating scales are directly computed, and not modelled. Because of this, LES has not been as thoroughly investigated and developed the last decades as RANS; nonetheless, as a result of the mentioned increase in computational power, combined with it having become more available overall, the field of LES has grown substantially the last years, becoming a good competitor to the well-established RANS methods.

1.1 Thesis Outline

As the main purpose of this work has been on implementing a general Large Eddy Simulation framework and a range of turbulence models, this thesis has been divided into three parts:

1. Theory.
2. Implementation.
3. Verification and validation of the implementations, again divided into a formal and an applied part.

It should be heavily stressed that all the ingredients in the first section are well known work and theory, whereas sections two and three are mainly based on my own research. References to the original article and authors will be given at all times. Where no references are given (for more general theory) books like those of Pope [4] and Sagaut [5] have been applied.

The first section, found in Ch. 2 and Ch. 3, is for constructing the base theory, both in terms of general mathematical and numerical methods, and for turbulence modelling and LES. In Ch. 2 the governing fluid equations are presented, CFD and the numerical method of choice for this thesis is discussed, discretisation of the Navier-Stokes equations is done, followed by a short introduction to both the Finite Element method (FEM) framework FEniCS, and the *Oasis* solver. As for the third chapter a short introduction to DNS and turbulence modelling methods is given, justifying why turbulence models are needed, and presenting a small selection of modelling methods. The rest of the chapter is devoted mainly to LES, where first the governing theory is introduced, followed by the presentation of a range of subgrid-scale models based on the eddy viscosity hypothesis. To end the chapter some problems with LES are discussed, and a very brief state-of-the-art presentation of the field is given.

The second section, mainly found in Ch. 4, covers all the implementational aspects of both the general eddy viscosity terms, and the LES models. First the general residual stress tensor, and subsequently the eddy viscosity formulation, is discretised with FEM, followed by a discussion of its implementation into the *Oasis* solver. This is succeeded by implementational details for all the LES models presented in Ch. 3. Thorough discussions are done, especially for the parts of the work that have been the most challenging. In addition, some very short code snippets illustrating essential parts of the different implementations are included.

The third section, found in Ch. 5 and Ch. 6, contains all the work done in terms of verification and validation of the implementations. It may be split into two parts: a formal one, and an applied one. The formal part is constructed of traditional convergence tests for the general implementation applying three types of fabricated eddy viscosities, and convergence tests for both the Smagorinsky, and the WALE models. In addition, a set of simple test cases are included to assess and further verify the implementation of the LES models. As for the applied part, it is based on two simulations of choice: the standard benchmark case of fully developed turbulent channel flow, and a more untraditional case consisting of blood flow in a pipe followed by a nozzle, a narrow pipe, and a sudden expansion. The former is a case that is much used in the CFD community when solvers are to be verified, that being either DNS solvers (both resolved and under-resolved ones), or turbulence solvers based on either RANS, LES, or other methods. Obtained mean data for velocities and some Reynolds stresses will be compared to data obtained for simulations of fully resolved DNS. As for the latter case it is based on the U.S. Food and Drug Administration (USFDA/FDA)'s computational round robin #1, where also here obtained mean data will be compared to what is measured by Particle Image Velocimetry (PIV) methods in the laboratory.

The whole thesis will be wrapped up in a concluding chapter Ch. 7, where, in addition to concluding remarks, some general limitations, the significance of this work, and future research topics of interest are discussed.

1.2 Motivation

Fluid mechanics, and more specifically turbulence modelling, in combination with computer programming and numerical methods, did not become a topic of interest for myself until two and a half, maybe three years ago. I have always had a great interest in computers and how they work, and I still remember well the first Hewlett-Packard my family acquired back in '95 or '96. In despite of a hard drive of 1.5 gigabytes, a 300 MHz processor, and about 8 megabytes of memory (which eventually was upgraded to 16 megabytes), it allowed us to do a lot of exciting things. Compared to my current privately built desktop computer, which boasts a 4.5 GHz processor, 16 gigabytes of memory, and a total hard drive capacity of around three terabytes, it becomes quite clear that a massive technological leap has found place during the last 20 to 30 years.

I have certainly not always had a great interest in physics and mathematics (compared to computers), as it was not until I started high school that I got substantially more interested in the two fields, then thanks to my great maths and physics teacher Olav. Out of the two physics has always been my favourite; though exceptionally cliché, it is in fact so that the quest for understanding and describing the world around us is a very exciting one (at least speaking for myself). In addition, mathematics was interesting, especially when combined with physics, or applying it for other practical purposes. When leaving high school for the university, it became quite clear that the so-called MIT program at the University of Oslo (UiO), which mixes informatics, applied mathematics, science and more into one exciting package, was the right way to go.

When I started at the MIT program, and during the succeeding semesters, it is without a doubt clear that the UiO professors Hans Petter Langtangen, Knut Mørken, and Anders Malthe-Sørenssen, all did a tremendously good job in promoting computational science and computational physics. I remember especially well the first semester course I had with Langtangen, where he during one of the introductory classes presented a slide containing the picture of an airplane with some nicely coloured streamlines, a set of highly advanced (partial differential) equations, and some computer code of some type. “This is what we do!” he said. “And this is something you can do too!”. This was one of the reasons for why I choose the mechanics direction of the MIT program, where I followingly developed a greater interest for flowing fluids, compared to what I did for solid mechanics.

One last question remains unanswered: why turbulence modelling, and more specifically: why Large Eddy Simulation? This is the hardest question to answer, because of there being no clear or good answers. I am, as so many others, simply fascinated by turbulence; it is just so intricate, chaotic, captivating, overwhelming, enchanting and extremely beautiful, all at the same time. Coupling turbulence with numerical methods and CFD seemed like an exciting way to go, where again Large Eddy Simulation seemed like the more exciting way, compared to other paths. Maybe it is because of the name of the method, or because of the fascinating names to some of the contributors to the field (I remember reading some related literature where names as Smagorinsky, Deardorff and Germano fascinated me). Large Eddy Simulation also seemed like a more modern an interesting method, not as old and “simple” as RANS, in addition to it being up-and-coming, i.e. still in its earlier stages in terms of maturity and development. Nevertheless, in despite of there being no clear reasons, this is where I am today, this is the way I have chosen, and it feels like the correct way. This will hopefully be reflected through this thesis.

Originally the implementations of the LES equations and the models, had been done for a homemade solver that applied the same numerical methods as *Oasis*, roughly speaking. In January ’15, after a meeting which included my supervisor, the co-developer of *Oasis* Kristian V.-Sendstad, his master student Aslak Bergersen, and myself, it was planned that we were going to contribute with two articles to the MekIT’15 conference in Trondheim in May. This did eventually result in the transition from my homemade solver to *Oasis*, the article “Implementation, Verification and Validation of Large Eddy Simulation Models in *Oasis*“ [6], and a presentation at the conference.

Chapter 2

General Theory and Governing Equations

2.1 The Incompressible Navier-Stokes Equations

Named after their fathers Claude-Louis Navier and George Gabriel Stokes, the so-called Navier-Stokes equations are a set of completely general partial differential equations (PDEs) that describe any type of viscous fluid flow in a continuum. The velocity vector

$$u_i\left(x_i\left(t\right), t\right) = \left[u, v, w\right], \tag{2.1}$$

where u, v and w represents fluid velocities in the x, y, and z-directions, respectively, does, together with the pressure p, represent the four unknown terms in the equations. In addition, one for incompressible flows has two flow parameters: the dynamic viscosity μ representing the "thickness" of the fluid, and the constant ρ representing the density of the fluid. If one are working with fluid flow in gases at high velocities, e.g. for supersonic aircraft or jet engines, one has to take into account that the air is compressible, leading to ρ becoming a function of time and space. However, when working with flows of water, blood, oil or similar, one then assumes that the fluid is incompressible, and ρ followingly becomes a constant parameter.

The Navier-Stokes equations are derived through conservation of mass and momentum for a fixed control volume V, where one may apply e.g. Newton's second law to arrive at the final equations. Leaving out the details, the full Navier-Stokes equation set for an incompressible fluid can be written on summation notation as

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} + f_{b,i}, \tag{2.2}$$

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

where we now have four momentum equations (one for each velocity component), and the incompressible constraint requiring that u_i is divergence free at all times. Here ρ and p have been combined, such that one directly solves the equation set for a modified pressure as $p^* = p/\rho$, where p^* is written as p for simplicity. In addition, the kinematic viscosity $\nu = \mu/\rho$ has been defined, and the equations include a force vector $f_{b,i}$ for body forces like gravity or similar.

If the Navier-Stokes equations (2.2) and (2.3) are non-dimensionalised, a dimensionless parameter known as the Reynolds number, named after Osborne Reynolds, presents itself. The Reynolds number, denoted Re, is constructed of a problem specific length scale d, a problem specific mean or typical velocity U, and the kinematic viscosity ν, as

$$Re = \frac{Ud}{\nu}. \tag{2.4}$$

Said in another way, the Reynolds number describes the ratio of momentum forces to viscous forces, where large Re means that the momentum forces are dominating, whereas low Re means that the viscous forces are dominating. Its value is somewhat problem specific: it will vary, then depending on the geometry, the typical velocity, and ν, though one may in general say that for large Re one has turbulent flows, whereas one for lower Re has weakly turbulent, or even laminar, flows. It is a parameter that effectively may give us an indication on the flowing state of the fluid we are analysing.

2.1.1 Boundary and Initial Conditions

Inlet

Very often one operates with a type of inlet in fluid flow; that may be e.g. a pipe inlet, air entering into an airplane engine, a river inlet, or similar. For incompressible flows the velocity may be set to a given function as

$$u_i = u_{i,\text{in}}(x, y, z, t) \text{ on } \partial\Omega_{\text{in}} \tag{2.5}$$

describing the inlet flow according to the problem. The notation $\partial\Omega_{\text{in}}$ is here used to state that the condition is applied to the inlet boundary only. This type of boundary condition is known as a Dirichlet condition.

Outlet

Where one may set a fixed velocity profile at the inlet, one at the outlet do not know the velocity profile or anything similar. It is normal to do two things at an outlet: first, require that the normal derivative of the velocity vector is equal to zero

$$\frac{\partial u_i}{\partial x_j} n_{j,\text{out}} = 0 \text{ on } \partial\Omega_{\text{out}}, \tag{2.6}$$

also known as a Neumann boundary condition. Secondly, it is common to fix the pressure at the outlet as e.g.

$$p = 0 \text{ on } \partial\Omega_{\text{out}}. \tag{2.7}$$

Solid Boundaries

At a solid boundary, one often assumes that no fluid passes through it, i.e.

$$u_i n_{i,\text{Solid Boundary}} = 0 \text{ on } \partial\Omega_{\text{Solid Boundary}}, \tag{2.8}$$

and that the velocity parallel to the wall is equal to the speed of the wall itself as

$$u_i t_{i,\text{Solid Boundary}} = u_{\text{Solid Boundary}} \text{ on } \partial\Omega_{\text{Solid Boundary}}. \tag{2.9}$$

If it is a non-moving solid boundary one applies the traditional no-slip condition as

$$u_i = 0 \text{ on } \partial\Omega_{\text{wall}}, \tag{2.10}$$

simply stating that all the velocity components are equal to zero at the wall.

Periodic

Periodic boundary conditions are applied in simulations where the flow field are of a periodically repeating nature. For the case of fully developed turbulent channel flow, where one assumes that the flow is between to plates infinitely long in both the x- and the z-directions, periodic conditions are applied as

$$u_i(x = 0, y, z) = u_i(x = L_x, y, z), \tag{2.11}$$

and

$$u_i(x, y, z = -L_z/2) = u_i(x, y, z = L_z/2). \tag{2.12}$$

Simplified by words: what flows out on one side, flows in on the other side.

Initial Conditions

Several possibilities exist, where the simplest one just initiates the fluid to be fully at rest at start-up, and the flow is then subsequently activated by some inlet condition or moving wall. Another possibility is to apply perturbed steady state solutions to trigger turbulent behaviour, e.g. in a pipe or similar. The turbulent behaviour would most likely have been triggered naturally after some time; however, perturbing an initial flow may eventually result in such turbulent behaviour at start-up, reducing computational time.

2.2 Numerical Methods

For analysing chaotic flow phenomena in three-dimensional complex domains, analytical work on the Navier-Stokes equations is severely limited because of their advanced nature and general complexity. This, together with the great need of analysing such flow situations, has led to the numerical tool of Computational Fluid Dynamics (CFD) becoming an important companion to experimental laboratory work. Often nicknamed Colourful Fluid Dynamics (see Fig. 2.1), the method is often, but not necessarily, developed applying the Navier-Stokes equations. All three velocity components and the pressure are solved for, where subsequent analysis can be done regarding flow speed, velocities, drag, lift or impact forces on objects or walls, particle tracking in the flow field, and much more.

When it comes to discretisation of the Navier-Stokes equations, there exist many techniques, each method having their own pros and cons. In general, for a type of discretisation to be applicable, it must be able to handle complex geometries (unstructured meshes), the solution it returns must be correct to some expected order of the discretisation parameters $\Delta t, \Delta x, \Delta y$ and Δz (higher order is preferable), it needs to be optimized and quick, and must include good numerical properties as e.g. little or no numerical diffusion and conservation of quantities.

The most well-known discretisational technique when it comes to CFD is the Finite Volume method (FVM), it being the fundamental approach in well-known software packages as ANSYS Fluent [7], Star-CD [8], and OpenFOAM [9], among others. The Finite Volume method is distinct in that one operates with control volumes as computational cells, where surface integrals over these volumes are

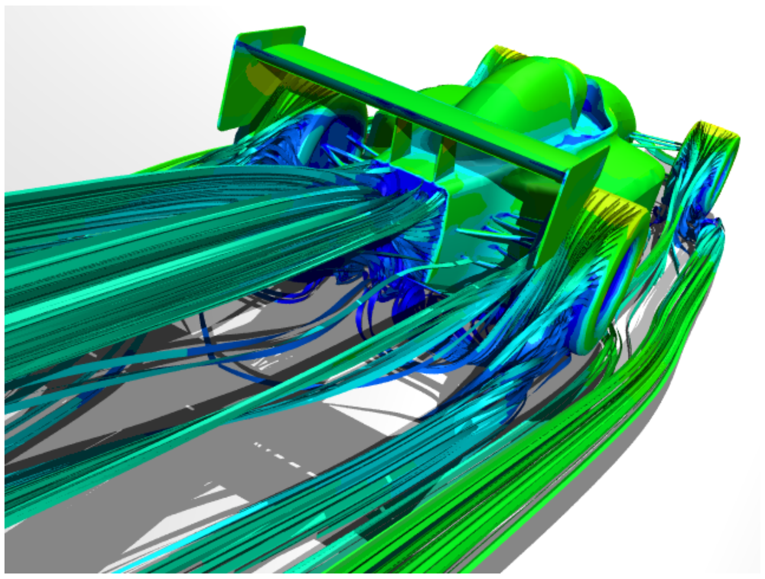


Figure 2.1: Streamlines for flow around a F1 race car (Wikimedia Commons).

discretised, resulting in discretised equations with good properties in terms of conservation. In addition, the method is easily extended to three-dimensional unstructured meshes, thereby becoming generally applicable for complex problems.

For this thesis, the discretisational method the Finite Element method (FEM) has been chosen. Instead of control volumes FEM operates with a finite number of elements, or cells, in the domain, where again the solution to some PDE is constructed by a linear combination of locally supported basis functions. The Finite Element method, though rather heavy in terms of mathematics, is a completely general and a very elegant method, capable of handling unstructured meshes and complex domains, where again the accuracy of the solution can easily be controlled through the chosen basis functions. In addition, the Galerkin projection method applied when minimising the residuals in FEM is very conservative when it comes to numerical diffusion, compared to FVM where this phenomenon is more problematic.

In addition to FVM and FEM, one has the Spectral methods (SMs, see e.g. [10, 11]), which in many ways are similar to FEM because of their decomposition applying basis functions. Nevertheless, where FEM applies piecewise locally supported basis functions, SMs apply globally supported basis functions through Fourier series, together with Fast Fourier Transform (FFT) algorithms. SMs are known for their excellent error properties, but they are problematic when being applied with complex meshes or for cases where shock waves or similar discontinuities are present. Other methods includes mesh-free approaches such as Smoothed-Particle Hydrodynamics (SPH) [12], or the Lattice-Boltzmann method (LBM) [13, 14], both of them being particle based approaches.

2.2.1 The Incremental Pressure Correction Scheme

Instead of solving the Navier-Stokes equations by a computationally demanding, though robust, coupled solver, a much better, faster, and still robust method may be introduced by uncoupling the equations by a fractional step method. A classic uncoupling is the method known as Chorin's scheme, put forward by A. Chorin in 1967 [15]. *Oasis* applies the Incremental Pressure Correction Scheme (IPCS) put forward by Simo et al. [16], a slightly improved version of Chorin's method, where also the pressure, or a guessed pressure, is included in the equation for the tentative velocity.

We now assume that we have an equation for a divergence free velocity u^{n+1} as a function of the correct pressure p, and one for a non-divergence free tentative velocity $u^\star$ as a function of a guessed pressure $p^\star$. This leads to the equations

$$\frac{\partial u_i^{n+1}}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} + f_{b,i}, \tag{2.13}$$

$$\frac{\partial u_i^\star}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p^\star}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} + f_{b,i}, \tag{2.14}$$

$$\frac{\partial u_i^{n+1}}{\partial x_i} = 0, \tag{2.15}$$

$$\frac{\partial u_i^\star}{\partial x_i} \neq 0. \tag{2.16}$$

Discretisation in time is done by applying the Crank-Nicolson method, where one generally assumes that the discrete derivative between time points n and $n+1$ is valid for time point $n+1/2$. This results in

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + u_j^{n+1/2} \frac{\partial u_i^{n+1/2}}{\partial x_j} = \left(-\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} - \frac{\partial \tau_{ij}^r}{\partial x_j} + f_{b,i} \right)^{n+1/2}, \tag{2.17}$$

and

$$\frac{u_i^\star - u_i^n}{\Delta t} + u_j^{n+1/2} \frac{\partial u_i^{n+1/2}}{\partial x_j} = \left(-\frac{\partial p^\star}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} - \frac{\partial \tau_{ij}^r}{\partial x_j} + f_{b,i} \right)^{n+1/2}. \tag{2.18}$$

The pressure term $p^{\star,n+1/2}$ is now approximated to be equal to the previous midpoint pressure as $p^{n-1/2}$. Notice now how equation Eq. (2.18) is solvable for the tentative velocity $u^\star$ if we apply the midpoint average as

$$u_i^{n+1/2} \simeq \frac{1}{2}\left(u_i^\star + u_i^n\right). \tag{2.19}$$

To avoid the nonlinearity and coupling problem for the convective term the Adams-Bashforth projection of $u_j^{n+1/2}$ is applied as

$$u_{j,AB}^{n+1/2} \simeq \frac{3}{2} u_j^n - \frac{1}{2} u_j^{n-1}. \tag{2.20}$$

When Eqs. (2.19) and (2.20) have been inserted into Eq. (2.18), we may solve for $u_i^\star$.

To proceed we subtract Eq. (2.18) from Eq. (2.17), which after some rearrangement yields

$$u_i^{n+1} = u_i^\star - \Delta t \frac{\partial}{\partial x_i}\left(p^{n+1/2} - p^{n-1/2}\right). \tag{2.21}$$

The divergence operator is now applied together with Eqs. (2.15) and (2.16) resulting in

$$\frac{\partial^2 p^{n+1/2}}{\partial x_i \partial x_i} = \frac{\partial^2 p^{n-1/2}}{\partial x_i \partial x_i} - \frac{1}{\Delta t} \frac{\partial u_i^\star}{\partial x_i}. \tag{2.22}$$

This equation is solvable for $p^{n+1/2}$, where $p^{n+1/2}$ then may be applied in Eq. (2.21) such that u_i^{n+1} may be computed.

The IPCS method can now be summarized as a three-step procedure:

1. Solve Eq. (2.18) for the tentative velocity $u_i^\star$.
2. Solve the pressure equation Eq. (2.22) for the pressure $p^{n+1/2}$.
3. Correct the velocity applying Eq. (2.21) to obtain u_i^{n+1}.

2.2.2 The Finite Element Method

In the Finite Element Method the domain Ω is split into a finite number of smaller subdomains, or elements, in addition to defining local basis functions for each element. The methods main ansatz is that any function may be written as a linear combination of a finite set of basis functions, e.g. for a general function f dependent on both time and space one in the finite dimensional case has

$$f\left(t_n, x\right) \simeq \hat{f} = \sum_{j=0}^{N} F_j^n \phi_j\left(x\right), \tag{2.23}$$

where F_j^n are coefficients at computational node j for time step t_n, and ϕ_j are the locally defined basis-, or trial functions, at node j. Subsequently, a set of test functions ψ_i are defined as

$$\left\{\psi_i : i = 0, \ldots, N\right\}. \tag{2.24}$$

Minimization of the arising error $e = f - \hat{f}$ is then done by Galerkin's projection method, where one requires that the error is orthogonal to all test functions ψ_i as

$$\langle e, \psi_i \rangle = \int_\Omega e\psi_i dx = 0. \tag{2.25}$$

The ansatz in Eq. (2.23) may now be inserted into Eq. (2.25), eventually resulting in a global matrix system that may be solved for the coefficients F_j^n by some appropriate method.

As for the trial- and test functions ϕ_j and ψ_i they are often, but not necessarily, equal to locally defined polynomial functions of some order. The most famous of these, it being known as "the" finite element, is the so-called P_1 Lagrange element. Computational nodes are located at the vertices of each element, where again first order linear polynomials are the basis functions of choice. In general for a P_N element the local basis functions are polynomials of order N, and the total number of nodes per element depends on both the element type and the polynomial order. It will in the rest of this thesis be referred to P_N elements and P_N basis functions/function space, where we then are talking about the element type or the basis functions, respectively, for a P_N Lagrange element.

For more details regarding FEM, it is referred to the FEniCS book [17].

2.2.3 Spatial Discretisation of the Tentative Velocity Equation

Spatial discretisations of all the equations presented for the IPCS scheme have already been done in the original Oasis paper [18]; nevertheless, since some important ingredients of the discretised tentative velocity equation will be used when the Large Eddy Simulation (LES) contribution is to be implemented into the *Oasis* solver, it has been included here for completeness.

We now apply the standard FEM recipe on Eq. (2.18), that is multiply the equations by a test function v, and integrate them over the domain Ω (the force term has been left out for simplicity), This yields

$$\left\langle \frac{u_i^\star - u_i^n}{\Delta t}, v \right\rangle + \left\langle u_j \frac{\partial u_i}{\partial x_j}, v \right\rangle^{n+1/2} = -\left\langle \frac{\partial p^{n-1/2}}{\partial x_i}, v \right\rangle + \left\langle \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j}, v \right\rangle^{n+1/2} . \quad (2.26)$$

Integration by parts on the Laplacian term is now done by applying Green's first identity as

$$\left\langle \frac{\partial^2 u_i}{\partial x_j \partial x_j}, v \right\rangle = \int_{\partial\Omega} \frac{\partial u_i}{\partial x_j} n_j v ds - \left\langle \frac{\partial u_i}{\partial x_j}, \frac{\partial v}{\partial x_j} \right\rangle . \quad (2.27)$$

Neumann boundary conditions may now be incorporated through the surface integral term; therefore, by applying Eq. (2.6) the term can be removed from the equation.

We now apply the results from Eq. (2.27) and Eq. (2.6), together with substituting the velocities at time step $n + 1/2$ from Eqs. (2.19) and (2.20) into Eq. (2.26). After some rearrangement of unknown and known terms to the left and right hand sides, respectively, one arrives at the equation

$$\frac{1}{\Delta t} \langle u_i^\star, v \rangle + \frac{1}{2} \left\langle u_{j,AB}^{n+1/2} \frac{\partial u_i^\star}{\partial x_j}, v \right\rangle + \frac{1}{2}\nu \left\langle \frac{\partial u_i^\star}{\partial x_j}, \frac{\partial v}{\partial x_j} \right\rangle =$$
$$\frac{1}{\Delta t} \langle u_i^n, v \rangle - \frac{1}{2} \left\langle u_{j,AB}^{n+1/2} \frac{\partial u_i^n}{\partial x_j}, v \right\rangle - \frac{1}{2}\nu \left\langle \frac{\partial u_i^n}{\partial x_j}, \frac{\partial v}{\partial x_j} \right\rangle - \left\langle \frac{\partial p^{n-1/2}}{\partial x_i}, v \right\rangle . \quad (2.28)$$

As a next step the velocity vector notation is abandoned in favour of the standard FEM ansatz, resulting in

$$u_i^n = [u^n, v^n, w^n] = \left[\sum_{j=0}^{N} U_j^n \phi_j, \sum_{j=0}^{N} V_j^n \phi_j, \sum_{j=0}^{N} W_j^n \phi_j \right] , \quad (2.29)$$

$$p^n = \sum_{j=0}^{N} P_j^n \psi_j, \quad (2.30)$$

where ϕ_j and ψ_j are the trial functions for the velocity and the pressure spaces, respectively. Inserting the decompositions, applying $v = \phi_i$ for the test functions, and rewriting on matrix notation results in an equation for $U_j^\star$ as

$$\begin{aligned}\left(\frac{1}{\Delta t}M_{ij} + \frac{1}{2}\nu K_{ij} + \frac{1}{2}X_{ij}\right) U_j^\star = \left(\frac{1}{\Delta t}M_{ij} - \frac{1}{2}\nu K_{ij} - \frac{1}{2}X_{ij}\right) U_j^n \\ - Q_{ij,1}P_j^{n-1/2}.\end{aligned} \tag{2.31}$$

Five new matrices have been defined as

$$M_{ij} = \langle \phi_i, \phi_j \rangle \,, \tag{2.32}$$

$$K_{ij} = \left\langle \frac{\partial \phi_i}{\partial x_k}, \frac{\partial \phi_j}{\partial x_k} \right\rangle , \tag{2.33}$$

$$X_{ij}(t) = \left\langle u_{k,AB}^{n+1/2} \frac{\partial \phi_j}{\partial x_k}, \phi_i \right\rangle , \tag{2.34}$$

$$Q_{ij,l} = \left\langle \frac{\partial \psi_j}{\partial x_l}, \phi_i \right\rangle , \tag{2.35}$$

where index k in K_{ij} and X_{ij} defines summation from 1 to 3, and the index l in $Q_{ij,l}$ is equal to either 1, 2 or 3, dependent on which velocity component we are solving for. The equations for the two other velocity components are identical to Eq. (2.31), the only difference is for the pressure term where $Q_{ij,l}$ depends on which velocity component one solves for. Now all the matrices in Eq. (2.31) may together with the velocity specific right hand sides be computed, where the subsequent linear systems can be solved for the three tentative velocity components.

2.2.4 FEniCS

The framework FEniCS, developed as a joint work between several research institutions including the Fornebu situated Simula Research Laboratory, is a completely general FEM framework for numerically solving PDEs of all types in both time and space. The framework comes as a package named *dolfin*, which includes all the functionality needed to solve a problem applying FEM. The whole discretisation process is done by the framework, making it fast and easy to solve simple problems, in addition to the user having the ability to control many aspects of the ongoing procedure if wanted. The user may (but is not restricted to) apply the library through high level Python scripting.

FEniCS is heavily dependent on the so-called Unified Form Language (UFL), which renders the user able to write Python code that translates nearly one-to-one to mathematical notation. To specify and solve a problem with FEniCS the user has to follow some simple steps:

1. Import the *dolfin* package.
2. Provide a mesh.
3. Choose function spaces for the trial- and test functions.
4. Input the weak form of the PDE to the solver applying the UFL.
5. Provide boundary conditions.
6. Start the solver.

As a simple example we apply the Poisson equation in three dimensions, given on its weak form as

$$\begin{aligned}\int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v dx \text{ in } \Omega,\\ f = x^2 + y^2 + z^2,\\ u = 0 \text{ on } \partial\Omega.\end{aligned} \tag{2.36}$$

First we import the packages and provide a mesh for the solver as

```
from dolfin import *

# Mesh -> 3D Unit Cube, 32 nodes in each direction
mesh = UnitCubeMesh(32, 32, 32)
```

Then we pick the wanted function space, and obtain the test and trial functions as

```
 # Function space of P1 Lagrange elements
 V = FunctionSpace(mesh, "Lagrange", 1)
 u,v = TrialFunction(V), TestFunction(V)
```

The function space V now applies the P_1 Lagrange elements described in section 2.2.2. We now input the right hand side function f by applying the dolfin function $Expression$, and define the UFL of the weak form as

```
 # Define f = x**2 + y**2 + z**2
 f = Expression("pow(x[0],2)+pow(x[1],2)+pow(x[2],2)")
 # Define UFL of a and b
 a = inner(grad(u),grad(v))*dx
 b = f*v*dx
```

Notice the UFL code applied here: *inner(a,b)* denotes the inner product of a and b; *grad(u)* denotes the gradient of u; dx tells the framework that this expression must be integrated over the whole domain. We also apply the mathematical operators +, -, * and / as usual. Assembly (computation of the integrals, and subsequently the matrix and right hand side vector) is now done as

```
Aij = assemble(a)   #LHS matrix
bi = assemble(b)   #RHS vector
```

In the end the boundary conditions are defined and applied to A and b, followed by initiation of the solver as

```
# Create DirichletBC u0(x,y,z) = 0 on the whole boundary
u0 = Constant(0)
bc = DirichletBC(V, u0, "on_boundary")
# Apply boundary condition to A and b
bc.apply(Aij,bi)

# Solve the problem and add solution to uj
uj = Function(V)
solve(Aij, uj.vector(), bi)
```

The solution to this problem is now stored in the function uj, where e.g. it may be stored to file and plotted by an appropriate visualization tool.

As previously mentioned one may with FEniCS solve all types of problems easy and fast (as the one above). However, for more complicated problems like e.g. the Navier-Stokes equations, one has to be smart when developing the code such that CPU usage, total simulation time, and memory usage is kept to a minimum. One such optimization, which in general is applicable for certain time dependent problems, is to, instead of just defining the UFL of the weak forms and then assemble the matrices each time step, exploit the fact that these matrices do not change in time. Thus, for the tentative velocity equation presented in the previous section, both M_{ij} and K_{ij} can be assembled prior to the time loop, resulting in a great speed up compared to if they had been computed each time step. On the other hand, the convective matrix X_{ij} is a function of time since the Adams-Bashforth projected velocity is changing with time, eventually resulting in assembly of this matrix being required each time step.

For more details regarding FEniCS it is referred the FEniCS book [17]. The FEniCS Project can be accessed online at [19].

2.2.5 Oasis

Oasis, developed in-house at the University of Oslo by Mikael Mortensen and Kristian Valen-Sendstad, is a freely available open source package for solving the incompressible Navier-Stokes equations by FEM through the FEniCS framework. The highly optimized solver applies energy conserving numerical schemes, combined with it being able to tackle fully unstructured meshes, such that complex flow problems and phenomena may be analysed. The solver and all its components are scripted in Python, where the user scripts the subsequent problem to be solved through the same programming language. Both FEniCS and *Oasis* support parallel computing through the Message Passing Interface (MPI) and the Python package “MPI for Python” (MPI4Py).

As the discretisational and implementational details regarding *Oasis* have been thoroughly covered in Mortensen et al. [18], more details will not be given here. Source code for the *Oasis* project is available online at [20], where the LES development branch, and followingly all the code developed as a part of this thesis, can be accessed through [21].

2.3 Conclusions

In this chapter the general analytical and numerical foundations for use in this thesis have been constructed. The Navier-Stokes equations describing fluid flow in a continuum were presented, together with some elaboration regarding initial and boundary conditions for the equations.

A short introduction to Computational Fluid Dynamics (CFD) was given, where the general idea was put forward, together with a brief discussion regarding some well-known discretisational techniques. The IPCS approach, where the Navier-Stokes equations are split into a three-equation procedure, was introduced, followed by a short introduction to FEM, and discretisation of IPCS' tentative velocity equation by the same method. In addition, a short introduction to the FEniCS project was given, followed by a simple implementational example that hopefully will work as a base of the somewhat more advanced code snippets being presented in Ch. 4. The chapter was wrapped up with a short introduction to the *Oasis* CFD solver.

Chapter 3

Turbulence Modelling and Large Eddy Simulation

3.1 Introduction

Discretising the Navier-Stokes equations by some nice methods in time and space, programming them into the computer, then solving them numerically, may at first sight seem like a simple task, possibly giving us the ability to tackle all types of flows, including those that consist of chaotic and turbulent phenomena. In general it is believed, but not confirmed, that the Navier-Stokes equations do indeed describe all types of flows, both laminar and turbulent, where the continuous representation contains all fluctuations, perturbations, nonlinearities, eddies of all sizes, and so on. From a continuous point of view, this is amazing; however, because of the transformation to the discrete description and general computational limitations, the obtained numerical results may be wrong compared to the exact description one possibly could have obtained in the continuous case.

So what is the problem? Well, the fact that the equations are so complex to handle that obtaining continuous solutions are at most times impossible, combined with the fact that there are huge computational limitations when solving the discretised equations, eventually leads to no continuous solutions, and no good numerical solutions. For the more complex cases that is. To cope with this problem something has to be done, such that we indeed may be able to tackle these cases, and followingly obtain solutions that are good, realistic, and actually reflect real life situations in some way.

3.1.1 Direct Numerical Simulation

In theory (and sometimes practically) it is indeed possible to do so-called Direct Numerical Simulation (DNS): just refine the mesh and the time step as much as possible, until one is assured that all scales, from the smallest Kolmogorov length scales to the largest scales, are resolved. The Kolmogorov length scale is defined as

$$\eta = \left(\nu^3/\epsilon\right)^{1/4}, \tag{3.1}$$

where ν is the kinematic viscosity and ϵ is the rate of dissipation of turbulent kinetic energy. Since ϵ is a function of u_i, the Kolmogorov scale is implicitly dependent on the Reynolds number Re, where higher Re eventually results in smaller length scales η. This means that the computational requirements of DNS to some extent is directly dependent on Re, where it can be shown that the total computational cost in terms of floating point operations (FLO) increases as approximately $160Re_L^3$ (Pope, [4]), where $Re_L = k^2/(\epsilon\nu)$ is the turbulent Reynolds number, and k is the turbulent kinetic energy. Similar requirements for the number of time steps and the number of grid points can be deduced, clearly illustrating that DNS is limited to moderate, or even low, Re.

In despite of the methods extreme requirements it has become more and more available for use, as a result of the huge leap seen in computational power the last decades. Most likely it will become even more feasible as the development of the transistors will continue, in addition to the possible introduction of new technologies as optical computing (electrons abandoned in favour of photons), or quantum computing. Both of these technologies are still in development, were the optical computing method is said to be available to the regular consumer in ten to twenty years. Quantum computing may arrive farther into the future; it is, nonetheless, still on the theoretical stage.

DNS computations for low enough Re is absolutely possible, and it has become an important tool for validating solvers, helping with understanding general turbulent behaviour and the Navier-Stokes equations, and for both *a priori* and *a posteriori* validation of turbulence models. Good DNS results have been achieved by applying Spectral Methods (section 2.2), where both Moser, Kim and Mansour [22] and Jimenez et al. [23] applied spectral solvers to obtain their DNS data for fully developed turbulent channel flow. The spectral methods have advantages in terms of efficiency and good error properties; hence, they work well for some specific DNS simulations. Nonetheless, if we are to simulate flows of increased Re, DNS must be abandoned in favour of more efficient methods. These methods aim towards computing some mean part of the flow only, instead accounting for the effects caused by the fluctuating scales by some appropriate turbulence models.

3.1.2 Reynolds-Averaged Navier-Stokes

The first method to be developed, and still the most popular turbulence modelling approach today, is the Reynolds-Averaged Navier-Stokes (RANS) method. RANS was first presented by Osborne Reynolds [2] in the 1890s, where his idea was to decompose the velocity and pressure fields into one mean and one fluctuating part as

$$u_i = \overline{u}_i + u'_i, \tag{3.2}$$

$$p = \overline{p} + p', \tag{3.3}$$

where the mean here is equal to an ensemble average operation. If we insert these decompositions into the Navier-Stokes equations followed by applying the mean operation, one arrives at the unclosed RANS equations as

$$\frac{\partial \overline{u}_i}{\partial t} + \overline{u}_j \frac{\partial \overline{u}_i}{\partial x_j} = -\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{u}_i}{\partial x_j \partial x_j} - \frac{\partial}{\partial x_j}\left(\overline{u'_i u'_j}\right). \tag{3.4}$$

Notice the new term $R_{ij} = \overline{u'_i u'_j}$, denoted the Reynolds stress, in general containing 6 unknown terms that need to be modelled by turbulence models, eventually rendering the equation set closed and solvable for the mean quantities $\overline{u}_i$ and $\overline{p}$.

As for modelling R_{ij} the bottom line is that it is difficult, especially in a completely general way that works for turbulent flows of all types and in all geometries. The most known methods of modelling are the ones based on the linear eddy viscosity assumption of Boussinesq [24], given as

$$\overline{u'_i u'_j} = -2\nu_T \overline{S}_{ij} + \frac{1}{3}\delta_{ij}\overline{u'_k u'_k}. \tag{3.5}$$

For modelling the eddy viscosity ν_T several approaches exist:

1. **Algebraic models**, which tends to model the eddy viscosity by simple, algebraic expressions.

2. **One equation models**, which introduce one extra PDE to be solved for some required quantity. Examples here are e.g. Prandtl's model, which solves an extra PDE for the turbulent kinetic energy k, and then models the eddy viscosity as $\nu_T = k^{1/2}l$ for some length scale l; or the Spalart-Allmaras model, which solves an extra PDE for a viscosity-like term $\widetilde{\nu}$.

3. **Two equation models**, which model the eddy viscosity by introducing two extra PDEs. Here we find the well known $k - \epsilon$ model, where PDEs for k and the turbulent dissipation ϵ are solved, then ν_T is computed as $\nu_T = C_\nu \frac{k^2}{\epsilon}$ for some constant C_ν. A different model is the $k - \omega$ model, where instead PDEs for k and the specific dissipation ω are solved, and $\nu_T = k/\omega$. Both these models are very common and substantially used in industry, many variants of them exist, and research is still being done today.

A second way of modelling the unknown terms in R_{ij} is by so-called Reynolds Stress Models (RSM). Then one abandons the eddy viscosity assumption and instead solve PDEs for all the terms in R_{ij} directly. In addition to the introduction of six extra PDEs, these equations are also constructed of new, unknown terms, which require further modelling and specification.

3.1.3 Large Eddy Simulation

Developed in the 1960s and 70s, mainly by Smagorinsky [3] (for simulating atmospheric flows), Lilly [25], Leonard [26] and Deardorff [27, 28], the turbulence modelling method of Large Eddy Simulation (LES) has become a good alternative to RANS methods the last decades. The main idea when doing LES is to filter out the smallest scales in the flow, resolve (simulate) the largest scales and eddies, and followingly model the effects caused by the smallest scales. Compared to RANS where the turbulence models must account the effects caused by all the fluctuations in the flow, the LES models need to take a substantially smaller amount of left out effects into account.

As a result of LES resolving a larger part of the flow field, it is also more demanding in terms of computational requirements compared to RANS. In addition, where RANS presents us with mean data directly, the LES equations are fully unsteady in time; thus, for good mean data to be obtained with LES the total simulation time needs to be increased. On the other hand LES presents us with much more detailed data in terms of velocity and pressure, together with the LES simulations, most likely, being much more "general" compared to RANS, since a much larger part of the flow field is resolved.

Since LES is the turbulence modelling method applied in this thesis it is thoroughly covered in sections 3.2 to 3.4.

3.1.4 Detached Eddy Simulation

As discussed in section 3.4.1 there are some problems with turbulent boundary layers when it comes to LES, where the velocity field in these regions actually may be better represented by RANS solutions. Hence, the method of Detached Eddy Simulation (DES), introduced by Spalart [29], which aim is to combine the best of two worlds: LES is applied in the regions that are "detached" from the wall, combined with the shift to RANS near walls. These hybrid methods require less computational resources compared to pure LES, but are still more demanding than RANS as the grid needs to be sufficiently fine for LES in certain regions.

For more details regarding DES it is referred to the original paper [29]. A wide selection of papers regarding other hybrid RANS-LES methods can be found in [30].

3.2 The Methodology of Large Eddy Simulation

3.2.1 Formal Definition of the Filtering Operation

As a first measure when doing LES one splits the velocity and pressure fields into a filtered and a residual quantity, equivalent to what is done in Eqs. (3.2) and

(3.3). The filtering operation is a type of spatial low-pass filter, defined as

$$\overline{u}_i(\mathbf{x}) = \int_{-\infty}^{\infty} u_i(\mathbf{r})\overline{G}(\mathbf{x}-\mathbf{r})d\mathbf{r}, \tag{3.6}$$

where $\overline{G}$ now denotes some type of filtering kernel. This filtering kernel is constructed in such a way that scales, or frequencies, smaller than a cutoff scale Δ are removed from the filtered quantity. Some examples are the box filter kernel

$$\overline{G} = \begin{cases} \frac{1}{\Delta}, & \text{if } |\mathbf{x}-\mathbf{r}| \leq \frac{\Delta}{2} \\ 0, & \text{else} \end{cases}, \tag{3.7}$$

the Gaussian filter kernel

$$\overline{G} = \left(\frac{6}{\pi\Delta^2}\right)^{1/2} \exp\left(-\frac{6\left(\mathbf{x}-\mathbf{r}\right)^2}{\Delta^2}\right), \tag{3.8}$$

and the sharp spectral filter kernel

$$\overline{G} = \frac{\sin\left(\pi\left(\mathbf{x}-\mathbf{r}\right)/\Delta\right)}{\pi\left(\mathbf{x}-\mathbf{r}\right)}. \tag{3.9}$$

The filter operation in Eq. (3.6) satisfies linearity,

$$\overline{f+g} = \overline{f} + \overline{g}, \tag{3.10}$$

given two general fields f and g, and commutation with derivatives (only for homogeneous filters) as

$$\overline{\frac{\partial f}{\partial x_i}} = \frac{\partial \overline{f}}{\partial x_i}, \tag{3.11}$$

Compared to RANS and its mean operation, the filtering operation of LES includes some differences. First, where we for RANS apply that $\overline{\overline{u}}_i = \overline{u}_i$, we for LES have that $\overline{\overline{u}}_i \neq \overline{u}_i$, since the filtered fields still are fluctuating in some manner. In addition, one for RANS applies that $\overline{u_i'} = 0$, something which neither is correct when applying the LES filtering operation. For a simple illustration see Fig. 3.1. There a random noised sine function has been filtered by applying the Gaussian kernel given in Eq. (3.8), together with a cutoff width $\Delta = 0.05$. Notice how the filtering operation removes the fluctuations and produces a much smoother profile in $\overline{u}$. The residual u' has been plotted at the bottom, where also the (nonzero) filtered residual $\overline{u'}$ is computed and plotted. The RANS counterpart of $\overline{u}$ illustrates the difference: RANS gives an overall mean value, LES filters perturbations below Δ only.

It should be noted that this filtering operation is not explicitly used in the type of Large Eddy Simulation implemented in this thesis. One instead assumes that the numerical solutions represent the filtered velocity and the filtered pressure directly; hence, the mesh together with the numerical scheme act together as the filter. There will, however, be some discussion regarding a second filtering kernel $\widehat{G}$, which is explicitly needed in the Dynamic Smagorinsky model (section 3.3.4).

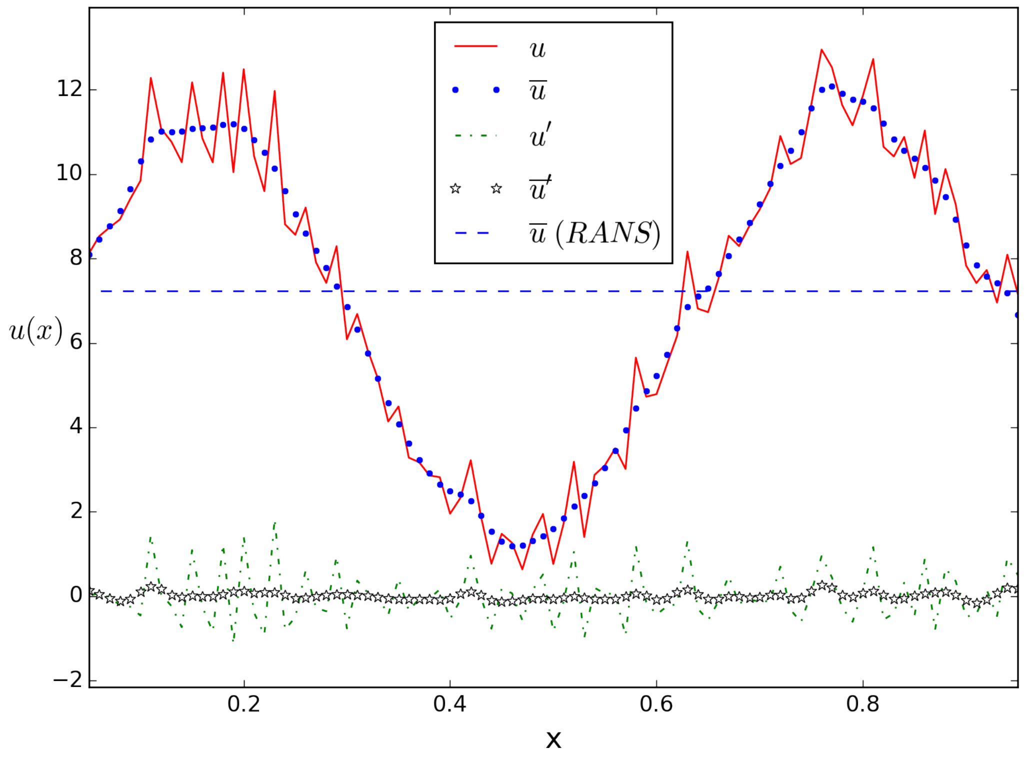


Figure 3.1: Gaussian filter on 1D function; $\Delta = 0.05$.

3.2.2 The Filtered Continuity Equation

Starting with the continuity equation Eq. (2.3) one applies the filter followed by commutation as

$$\overline{\frac{\partial u_i}{\partial x_i}} = \frac{\partial \overline{u}_i}{\partial x_i} = 0. \tag{3.12}$$

Thus, as for the original equation set, also the filtered velocity field $\overline{u}_i$ needs to be divergence free. This can again be used to show that the fluctuating velocity has the same constraint as

$$\frac{\partial}{\partial x_i}\left(u_i - \overline{u}_i - u_i'\right) = \frac{\partial u_i'}{\partial x_i} = 0. \tag{3.13}$$

3.2.3 The Filtered Momentum Equations

Now we apply the filter to the momentum equations in Eq. (2.2) as

$$\overline{\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} + f_{b,i}}, \tag{3.14}$$

which by linearity and commutation results in

$$\frac{\partial \overline{u}_i}{\partial t} + \frac{\partial}{\partial x_j}\overline{u_i u_j} = -\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{u}_i}{\partial x_j \partial x_j} + \overline{f}_{b,i}. \tag{3.15}$$

Notice how the convective term has been rewritten by applying the continuity equation as

$$\frac{\partial}{\partial x_j}\left(u_i u_j\right) = u_i \frac{\partial u_j}{\partial x_j} + u_j \frac{\partial u_i}{\partial x_j} = u_j \frac{\partial u_i}{\partial x_j}. \tag{3.16}$$

Further manipulation is done by adding the term $\overline{u}_j \frac{\partial \overline{u}_i}{\partial x_j}$ to both sides of the equation, and moving the convective term in Eq. (3.15) to the right hand side. This results in a slightly modified equation as

$$\frac{\partial \overline{u}_i}{\partial t} + \overline{u}_j \frac{\partial \overline{u}_i}{\partial x_j} = -\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{u}_i}{\partial x_j \partial x_j} - \frac{\partial \tau^r_{ij}}{\partial x_j} + \overline{f}_{b,i}, \tag{3.17}$$

where the new residual stress tensor, defined as

$$\tau^r_{ij} = \left(\overline{u_i u_j} - \overline{u}_i \overline{u}_j\right), \tag{3.18}$$

has been introduced.

Notice now how the equation set is unclosed since the residual stress tensor τ^r_{ij} is unknown. To close the system models for τ^r_{ij} need to be introduced, eventually rendering the equation set closed and solvable for both the filtered velocity vector $\overline{u}_i$ and the filtered pressure $\overline{p}$.

3.2.4 Decomposing the Residual Stress Tensor

Following [4] the residual stress tensor can be decomposed by inserting the velocity decomposition into the definition of the tensor, resulting in

$$\begin{aligned} \tau^r_{ij} &= \overline{u_i u_j} - \overline{u}_i \overline{u}_j = \overline{\left(\overline{u}_i + u'_i\right)\left(\overline{u}_j + u'_j\right)} - \left(\overline{\overline{u}_i + u'_i}\right)\left(\overline{\overline{u}_j + u'_j}\right) \\ &= \overline{\overline{u}_i \overline{u}_j} - \overline{\overline{u}}_i \overline{\overline{u}}_j + \overline{\overline{u}_i u'_j} + \overline{u'_i \overline{u}_j} - \overline{\overline{u}}_i \overline{u'}_j - \overline{u'}_i \overline{\overline{u}}_j + \overline{u'_i u'_j} - \overline{u'}_i \overline{u'}_j. \end{aligned}$$

Notice now how we in the last equality have regrouped the terms containing only the filtered velocity, a combination of the filtered and the residual velocity, and factors containing the residual velocity only. Now the Leonard, Clark and Reynolds tensors are defined as

$$L_{ij} = \overline{\overline{u}_i \overline{u}_j} - \overline{\overline{u}}_i \overline{\overline{u}}_j, \tag{3.19}$$

$$C_{ij} = \overline{\overline{u}_i u'_j} + \overline{u'_i \overline{u}_j} - \overline{\overline{u}}_i \overline{u'}_j - \overline{u'}_i \overline{\overline{u}}_j, \tag{3.20}$$

$$R_{ij} = \overline{u'_i u'_j} - \overline{u'}_i \overline{u'}_j, \tag{3.21}$$

where the expression for the residual stress tensor followingly is written as

$$\tau_{ij}^r = L_{ij} + C_{ij} + R_{ij}. \tag{3.22}$$

It is now clear that the residual stress tensor needs to model interactions between the larger scales, as seen through the Leonard tensor, interactions among large and small scales as seen through the Clark tensor, and interactions among the subgrid-scales, as seen through the Reynolds tensor. Notice also how the Leonard tensor actually is directly computable if the velocity vector $\overline{u}_i$ is known and an explicit filtering procedure together with a filtering kernel $\overline{G}$ is specified. In so-called scale-similar, or mixed, subgrid-scale models, one takes into account that the Leonard tensor is directly computable; hence, the LES model needs to account for the cross- and subgrid-scale interactions only. Some mixed models are presented in section 3.3.7, but neither implementational details, computational analysis, nor any validation will be done here, as the emphasis will be on models that model τ_{ij}^r in total.

3.3 Models for the Residual Stress Tensor

The most popular approach for modelling the residual stress tensor is to apply the eddy viscosity hypothesis of Boussinesq [24], and then assume that all the subgrid-scale effects in τ_{ij}^r may be modelled by a fictional eddy viscosity term. More precisely this hypothesis assumes that

$$\tau_{ij}^r = -2\nu_T \overline{S}_{ij} + \frac{1}{3}\tau_{kk}^r \delta_{ij}, \tag{3.23}$$

where

$$\overline{S}_{ij} = \frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right), \tag{3.24}$$

is the filtered rate of strain tensor, and ν_T is a general expression for the eddy viscosity. The trace term on the right hand side of Eq. (3.23) is combined with the pressure, such that one instead solves for a modified pressure as $\overline{p}^* = \overline{p} + \frac{1}{3}\tau_{kk}^r \delta_{ij}$ (for simplicity defined as $\overline{p}$). As for the eddy viscosity one can by dimensional reasoning show that it must have the dimensions of $\nu_T = \frac{\mathrm{m}^2}{\mathrm{s}}$; therefore, some appropriate length and time scale need to be defined such that ν_T can be computed, subsequently closing the equation set.

All the models, which are presented here (excluding the mixed models in section 3.3.7), have been implemented into the *Oasis* framework. Details regarding the implementation can be found in Ch. 4, the following sections will present the models and some theory surrounding them.

3.3.1 The Smagorinsky Model

The first LES model to be introduced, developed by Smagorinsky [3], and first used by Deardorff [27] in numerical LES simulations, is the so-called Smagorinsky model. It is constructed of a length scale term, a time scale term, and a general, dimensionless constant C_s named the Smagorinsky constant. Starting with the length scale it is assumed that it may be connected to the filter cutoff scale Δ; thus, since the mesh acts as the general filter, the idea is to apply some function of the mesh spacing $\overline{\Delta}$ as the general length scale term. A simple and widely applied solution is to apply the local cell volume as

$$\overline{\Delta} = V_{cell}^{1/d}, \tag{3.25}$$

where V_{cell} is the area/volume of a computational cell or element, and d equals the geometrical dimension of the problem. As for the time-scale term (or differential operator), it is set to be the magnitude of the filtered rate of strain tensor (3.24) as

$$|\overline{S}| = \sqrt{2\overline{S}_{ij}\overline{S}_{ij}}, \tag{3.26}$$

This results in the eddy viscosity being modelled as

$$\nu_T = \left(C_s\overline{\Delta}\right)^2 |\overline{S}|, \tag{3.27}$$

where now the term $\overline{\Delta}$ has been squared since the required length scale is m^2.

The Smagorinsky model is the simplest LES model available, and it is, in despite of its well-known flaws, still being used for a range of problems today. The constant C_s has been found to be in the range $[0.1, 0.2]$ (Pope [4]), where Deardorff [27] applied a constant value of $C_s = 0.1$ for turbulent channel flow. Modelling ν_T by this constant C_s, together with the use of the differential operator $|\overline{S}|$, is known to overpredict turbulent viscosity in near-wall regions and regions of simple shear flow. This behaviour may lead to excessive damping, and subsequent elimination of transitional effects, eventually rendering the model unsuitable for certain transitional flows. Improvements have been made by implementing damping functions near walls (e.g. Van Driest's damping function [31]), but these solutions do not fix the models problems in other regions, thus other routes for eliminating these problems have been found.

3.3.2 The Wall-Adapting Local Eddy-Viscosity Model

The Wall-Adapting Local Eddy-Viscosity model (WALE), developed by Nicoud et al. [32], attempts to fix the shortcomings with the Smagorinsky model by directly incorporating the correct behaviour into the eddy viscosity expression. Instead of defining appropriate functions near the wall or introducing dynamic procedures (see section 3.3.4), the idea is to create an expression for ν_T where the correct behaviour is baked into a suitable differential operator. The WALE model is very similar to Smagorinsky, where the eddy viscosity now is defined as

$$\nu_T = \left(C_w \overline{\Delta}\right)^2 D_W, \tag{3.28}$$

for $\overline{\Delta}$ in Eq. (3.25), and the WALE constant C_w shown by [32] to take the value

$$C_w \approx 3.26 C_s. \tag{3.29}$$

The differential operator D_W is still a function of the filtered rate of strain tensor in some form; it, however, has some major modifications compared to the Smagorinsky model. More precisely it is defined as

$$D_W = \frac{\left(S^d_{ij} S^d_{ij}\right)^{3/2}}{\left(\overline{S}_{ij}\overline{S}_{ij}\right)^{5/2} + \left(S^d_{ij} S^d_{ij}\right)^{5/4}}, \tag{3.30}$$

where

$$S^d_{ij} = \frac{1}{2}\left(g^2_{ij} + g^2_{ji}\right) - \frac{1}{3}\delta_{ij} g^2_{kk}. \tag{3.31}$$

Here

$$g_{ij} = \frac{\partial \overline{u}_i}{\partial x_j} \quad \text{and} \quad g^2_{ij} = g_{ik} g_{kj}. \tag{3.32}$$

The eddy viscosity for the WALE model can now be written as

$$\nu_T = (C_w \overline{\Delta})^2 \frac{\left(S^d_{ij} S^d_{ij}\right)^{3/2}}{\left(\overline{S}_{ij}\overline{S}_{ij}\right)^{5/2} + \left(S^d_{ij} S^d_{ij}\right)^{5/4}}. \tag{3.33}$$

The development of this new term D_W is quite extensive, it is referred to the original article [32] for a full justification.

3.3.3 The σ Model

The third differential operator model applied and implemented here is the so-called σ model, developed by Baya Toda et al. [33] as an improvement to the WALE model. Its form is similar to the Smagorinsky and the WALE models, it consisting of a free constant, found by [33] to have the value

$$C_\sigma \simeq \frac{3}{2}, \tag{3.34}$$

in addition to the filter width $\overline{\Delta}$ defined in Eq. (3.25). The eddy viscosity is then modelled as

$$\nu_T = \left(C_\sigma \overline{\Delta}\right)^2 D_\sigma, \tag{3.35}$$

for its differential operator D_σ.

This model attempts to incorporate all the wanted behaviour directly into D_σ, such that both the appropriate wall behaviour, and some other effects, are tackled correctly (see section 5.2). In the original paper [33] the authors propose to apply the gradient tensor of the filtered velocity in some way, similar to what the WALE model does. However, where WALE further applied the square of the gradient velocity tensor, the D_σ operator is instead based on the singular values $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq 0$ of the gradient tensor. The wanted behaviour for the model is actually found in these singular values, as they include properties that directly represents some special flow phenomena where the eddy viscosity should be equal to zero. The differential operator for the σ model is given by [33] as

$$D_\sigma = \frac{\sigma_3 \left(\sigma_1 - \sigma_2\right)\left(\sigma_2 - \sigma_3\right)}{\sigma_1^2}. \tag{3.36}$$

For further justification and derivation of this D_σ term it is referred to the original article [33].

3.3.4 The Dynamic Smagorinsky Model

The Dynamic Smagorinsky model, developed by Germano [34] and further refined by Lilly [35], attempts to tackle the problems experienced with the Smagorinsky model in a slightly different manner, compared to the previously presented differential operator models. Instead of incorporating the wanted effects through some differential operator, the general idea is to compute C_s dynamically as a function of the flow field. Germano applied the general expression for τ_{ij}^r in Eq. (3.18), together with a similar expression for a coarser filter level as

$$T_{ij}^r = \widehat{\overline{u_i u_j}} - \widehat{\overline{u}}_i \widehat{\overline{u}}_j, \tag{3.37}$$

where $\widehat{\overline{()}}$ equals a double filtering operation with a corresponding cutoff scale $\widehat{\overline{\Delta}}$. Combining Eq. (3.18) and Eq. (3.37), one may apply the Germano identity as

$$L_{ij} = T_{ij}^r - \widehat{\tau}_{ij}^r = \widehat{\overline{u_i u_j}} - \widehat{\overline{u}}_i \widehat{\overline{u}}_j - \widehat{\overline{u_i u_j}} + \widehat{\overline{u}_i \overline{u}_j} = \widehat{\overline{u}_i \overline{u}_j} - \widehat{\overline{u}}_i \widehat{\overline{u}}_j. \tag{3.38}$$

Notice that if a second filtering procedure $\widehat{()}$ is defined, the tensor L_{ij} is computable as a function of the resolved velocity field $\overline{u}_i$ only.

Applying the Smagorinsky model in Eq. (3.27) for both τ_{ij}^r and T_{ij}^r together with the general eddy viscosity model in Eq. (3.23), results in the deviatoric residual tensor at the mesh filter level being modelled as

$$\tau_{ij}^r = -2(C_s \overline{\Delta})^2 |\overline{S}| \overline{S}_{ij} + \frac{1}{3}\tau_{kk}^r \delta_{ij}, \tag{3.39}$$

and the similar deviatoric residual tensor at the coarser filter level being modelled as

$$T_{ij}^r = -2(C_s \widehat{\overline{\Delta}})^2 |\widehat{\overline{S}}| \widehat{\overline{S}}_{ij} + \frac{1}{3} T_{kk}^r \delta_{ij}, \tag{3.40}$$

where $\widehat{\overline{\Delta}}$ is defined as the filter width at the test filter level, and

$$\widehat{\overline{S}}_{ij} = \frac{1}{2}\left(\frac{\partial \widehat{\overline{u}}_i}{\partial x_j} + \frac{\partial \widehat{\overline{u}}_j}{\partial x_i}\right); \; |\widehat{\overline{S}}| = \sqrt{2\widehat{\overline{S}}_{ij}\widehat{\overline{S}}_{ij}}. \tag{3.41}$$

We now combine Eq. (3.38) with Eq. (3.39) and (3.40), which equates to

$$L_{ij} - \frac{1}{3}L_{kk}\delta_{ij} \equiv L^d_{ij} = \left(\widehat{2(C_s\overline{\Delta})^2|\overline{S}|\overline{S}_{ij}} - 2(C_s\widehat{\overline{\Delta}})^2|\widehat{\overline{S}}|\widehat{\overline{S}}_{ij}\right), \tag{3.42}$$

where now the deviatoric tensor L^d_{ij} must be applied as a result of the trace-terms in Eqs. (3.39) and (3.40). It is now assumed that C_s is independent of scale, and can thus, together with $\overline{\Delta}$, be moved outside the test filtering operation, resulting in the equation

$$L^d_{ij} = C_s^2 M_{ij}, \tag{3.43}$$

for

$$M_{ij} = 2\overline{\Delta}^2\left(\widehat{|\overline{S}|\overline{S}_{ij}} - \alpha^2|\widehat{\overline{S}}|\widehat{\overline{S}}_{ij}\right), \tag{3.44}$$

and

$$L_{ij} = \widehat{\overline{u}_i\overline{u}_j} - \widehat{\overline{u}}_i\widehat{\overline{u}}_j. \tag{3.45}$$

Notice the new constant $\alpha = \widehat{\overline{\Delta}}/\overline{\Delta}$ in M_{ij}, which equals the mesh-filter to test-filter ratio, a value often taken as $\alpha = 2$. Additionally, the main assumption of the Dynamic Smagorinsky model has been done, in that one assumes that C_s varies sufficiently small, and can hence be extracted from the filtering operation. This is not always true, as the Scale-Dependent model takes into account scale-dependency of C_s (see section 3.3.5).

A problem with Eq. (3.43) is that it is a tensor equation, resulting in six equations that need to be solved for one unknown C_s. To manipulate Eq. (3.43) into one equation for one unknown, Germano [34] originally proposed to multiply it by $\overline{S}_{ij}$, resulting in

$$C_s^2 = \frac{L_{ij}\overline{S}_{ij}}{M_{kl}\overline{S}_{kl}}. \tag{3.46}$$

This was further refined by Lilly [35] who did a least squares analysis of Eq. (3.43), obtaining a slightly modified expression as

$$C_s^2 = \frac{L_{ij}M_{ij}}{M_{kl}M_{kl}}. \tag{3.47}$$

Notice that we apply L_{ij} and not L^d_{ij} in Eqs. (3.46) and (3.47), that as a result of M_{ij} and $\overline{S}_{ij}$ both being deviatoric tensors for incompressible flows. It was discovered that large fluctuations and both extreme and negative values were present for C_s if computed from Eq. (3.47), hence further averaging was introduced for both the numerator and the denominator as

$$C_s^2 = \frac{|L_{ij}M_{ij}|}{|M_{kl}M_{kl}|}, \tag{3.48}$$

for some averaging procedure $|\cdot|$. In addition to the averaging procedure some clipping of C_s^2 may need to be included if unwanted values still are present. A common tweak that is applied a lot in literature, is to simply extract $\overline{\Delta}^2$ from M_{ij}, subsequently resulting in one being able to solve for $\left(C_s\overline{\Delta}\right)^2$ directly. This is a positive result as one then avoids explicit specification of the length scale, instead being able to retrieve it implicitly through the dynamic procedure.

When it comes to the averaging procedure there are different solutions which have been proposed and tested. Originally Germano [34] averaged in planes of homogeneous directions, but as this phenomenon is only present in some types of flows it is not generally applicable for more complex situations. Other solutions include planar averaging, local averaging over a few grid cells, or global averaging (not applicable with the differential operator $|\overline{S}|$).

As for this thesis a more general averaging approach has been chosen and implemented, such that the solver may be as generally applicable as possible. The method, proposed by Meneveau et al. [36], is based on an approach of averaging along fluid particle trajectories, thereby its name: Lagrangian averaging. Leaving out the details one eventually arrives at the following expression for C_s:

$$C_s^2 = \frac{J_{LM}}{J_{MM}}, \tag{3.49}$$

where J_{LM} and J_{MM} are obtained by solving two transport equations as

$$\frac{\partial J_{LM}}{\partial t} + \overline{u} \cdot \nabla J_{LM} = \frac{1}{T}\left(L_{ij}M_{ij} - J_{LM}\right), \tag{3.50}$$

$$\frac{\partial J_{MM}}{\partial t} + \overline{u} \cdot \nabla J_{MM} = \frac{1}{T}\left(M_{ij}M_{ij} - J_{MM}\right), \tag{3.51}$$

$$T = \theta\overline{\Delta}\left(J_{LM}J_{MM}\right)^{-1/8}, \tag{3.52}$$

for $\theta = 3/2$. The discretisation of these equations will be discussed in the next chapter, but the implementation has been done similarly to what is done in the original paper, then by applying some implicit scheme, such that the amount of extra resources required is decreased to a minimum. All the details will be given in section 4.3.4.

A General Dynamic Procedure

Assume that we now have a general eddy viscosity expression on the form

$$\nu_T = \Theta\Lambda\Gamma, \tag{3.53}$$

where Θ equals the fixed constant, Λ equals the length scale term, and Γ equals the differential operator of the model. For Smagorinsky we have that $\Theta = C_s^2$ and $\Gamma = |\overline{S}|$, where we for WALE similarly have $\Theta = C_w^2$ and $\Gamma = D_W$, and for σ model we have $\Theta = C_\sigma^2$ and $\Gamma = D_\sigma$. For all these models $\Lambda = \overline{\Delta}^2$.

Now the dynamic procedure is applied together with Eq. (3.53), leading to a set of completely general equations as

$$L_{ij} = \widehat{\overline{u}_i\overline{u}_j} - \widehat{\overline{u}}_i\widehat{\overline{u}}_j, \tag{3.54}$$

$$M_{ij} = 2\left(\widehat{\Gamma\overline{S}_{ij}} - \beta\widehat{\Gamma}\widehat{\overline{S}}_{ij}\right), \tag{3.55}$$

$$\beta = \widehat{\Lambda}/\Lambda, \tag{3.56}$$

$$\Theta\Lambda = \frac{|L_{ij}M_{ij}|}{|M_{kl}M_{kl}|}. \tag{3.57}$$

The dynamic procedure may now easily be coupled with whichever eddy viscosity model that fits the form shown in Eq. (3.53); thus, it becomes evident that, if the dynamic procedure has been implemented once, extending it to other models is a simple task. The only ingredient that require some extra attention is the differential operator, or time scale term Γ, since, for M_{ij} to be computable, Γ must be explicitly computed at both the mesh level and the test level. How challenging this is depends on the complexity of Γ, where one e.g. sees that the Smagorinsky model comes packed with a simpler term $|\overline{S}|$ compared to the D_W operator of WALE.

As for the averaging $|\cdot|$, the same methods as previously discussed may be applied. However, it is by Baya Toda et al. [37] discussed if this dynamic procedure is indeed applicable with models that already satisfy the wall decaying behaviour as y^3 for ν_T (this is a required property for the eddy viscosity, see Chapman [38]), like WALE and the σ model. This problem is the main motivation of the global dynamic model, where one simply averages the tensor contractions over the whole domain, thus eliminating the need for clipping procedures, subsequently arriving at a constant for Θ or $\Theta\Lambda$. This means that this global approach is not suitable for the Smagorinsky model, since that again would lead to problems in wall-regions; however, for WALE and the σ model, this approach is applicable as they have the correct behaviour directly baked into their differential operators.

The bottom line here is: extensions of the dynamic procedure to other eddy viscosity models that may be written as in Eq. (3.53) is a simple task, where followingly some other solutions have to be applied in terms of the averaging procedure. For some more details regarding the mentioned global procedure see section 3.4.2.

3.3.5 The Scale-Dependent Dynamic Smagorinsky Model

The Dynamic Smagorinsky model has been refined by Bou-Zeid et al. [39], where they take into account that C_s indeed is scale-dependent, and followingly the fact that the computed Smagorinsky constant C_s obtained from Eq. (3.48) actually is valid at the filter level $\widehat{\overline{u}}_i$, and not at level $\overline{u}_i$, as assumed above in the original dynamic procedure. Leaving out details in terms of derivation (it is referred to the original article [39]), they subsequently introduce a second test filtering level denoted $\widetilde{()}$ where $\widetilde{\overline{\Delta}}/\overline{\Delta} = \alpha^2$, and apply the same recipe as done in the previous section. This results in two new tensors

$$Q_{ij} = \widetilde{\overline{u}_i\overline{u}_j} - \widetilde{\overline{u}}_i\widetilde{\overline{u}}_j, \tag{3.58}$$

$$N_{ij} = 2\overline{\Delta}^2\left(\widetilde{|\overline{S}|\overline{S}_{ij}} - \alpha^4|\widetilde{\overline{S}}|\widetilde{\overline{S}}_{ij}\right), \tag{3.59}$$

and the equation for $C_{s,\widetilde{\Delta}}$ as

$$Q^d_{ij} = C_{s,\widetilde{\Delta}} N_{ij}. \tag{3.60}$$

The averaging is again done along fluid trajectories, thus one arrives at a similar set of transport equations to be solved as

$$\frac{\partial J_{QN}}{\partial t} + \overline{u}\cdot\nabla J_{QN} = \frac{1}{T_{\widetilde{\Delta}}}\left(Q_{ij}N_{ij} - J_{QN}\right), \tag{3.61}$$

$$\frac{\partial J_{NN}}{\partial t} + \overline{u}\cdot\nabla J_{NN} = \frac{1}{T_{\widetilde{\Delta}}}\left(N_{ij}N_{ij} - J_{NN}\right), \tag{3.62}$$

for $T_{\widetilde{\Delta}} = 1.5\overline{\Delta}\left(J_{QN}J_{NN}\right)^{-1/8}$. One is now able to obtain the Smagorinsky constant at the second test filtering level as

$$C^2_{s,\widetilde{\Delta}} = \frac{J_{QN}}{J_{NN}}. \tag{3.63}$$

To obtain the Smagorinsky constant C_s at the mesh level one assumes a constant ratio between scales such that

$$\beta = \frac{C^2_{s,\widehat{\Delta}}}{C^2_s} = \frac{C^2_{s,\widetilde{\Delta}}}{C^2_{s,\widehat{\Delta}}}, \tag{3.64}$$

where, when $C^2_{s,\widehat{\Delta}}$ and $C^2_{s,\widetilde{\Delta}}$ have been computed from Eq. (3.48) and Eq. (3.63), β and followingly C^2_s may be computed.

3.3.6 A Model Based on the Subgrid-scale Kinetic Energy

A model, which is slightly different from the previous ones, originally proposed by Yoshizawa and Horiuti [40], aims to apply the subgrid-scale kinetic energy k_{SGS} in the expression for the eddy viscosity, instead of some differential operator function. This subgrid-scale kinetic energy is defined as

$$k_{\mathrm{SGS}} = \frac{1}{2}\left(\overline{u_i u_i} - \overline{u}_i \overline{u}_i\right), \tag{3.65}$$

which, when this term is included in the expression for ν_T, results in

$$\nu_T = C_k \overline{\Delta} k_{\mathrm{SGS}}^{1/2}, \tag{3.66}$$

for some constant C_k. The kinetic subgrid-scale energy can then be computed from the transport equation

$$\frac{\partial k_{\mathrm{SGS}}}{\partial t} + \overline{u}_j \frac{\partial k_{\mathrm{SGS}}}{\partial x_j} = -\tau_{ij}^r \frac{\partial \overline{u}_i}{\partial x_j} - C_\epsilon \frac{k_{\mathrm{SGS}}^{3/2}}{\overline{\Delta}} + \frac{\partial}{\partial x_j}\left(\frac{\nu_T}{\sigma}\frac{\partial k_{\mathrm{SGS}}}{\partial x_j}\right), \tag{3.67}$$

for some constant C_ϵ, and $\sigma = 1$.

C_k and C_ϵ must now be determined, either by some analysis resulting in two constants, or by further developing this method by introducing dynamic computation of them (see e.g. [41, 42]). In this work the implementation has been done applying constant values for C_k and C_ϵ only. This model will be referred to as the Kinetic-Energy SGS model in the succeeding sections.

3.3.7 Scale Similarity / Mixed Models

Another interesting way of modelling the stress tensor was proposed by Bardina et al. [43], where they by applying so-called scale similarity assumes that

$$\tau_{ij}^r = \overline{u_i u_j} - \overline{u}_i \overline{u}_j \sim \overline{\overline{u}_i \overline{u}_j} - \overline{\overline{u}}_i \overline{\overline{u}}_j, \tag{3.68}$$

which evidently is equal to the Leonard tensor in Eq. (3.19). Hence, this Leonard term can be explicitly be computed, whereas the Clark and Reynolds terms C_{ij} and R_{ij} must be modelled.

Applying this scale similarity model as it stands in Eq. (3.68) is rarely done; instead, one combines it with other eddy viscosity models, such as e.g. the Smagorinsky model, producing the so-called Mixed Smagorinsky model as

$$\tau_{ij}^r = \left(\overline{\overline{u}_i \overline{u}_j} - \overline{\overline{u}}_i \overline{\overline{u}}_j\right)^d - 2\left(C_s \overline{\Delta}\right)^2 |\overline{S}| \overline{S}_{ij}, \tag{3.69}$$

where the d denotes the deviatoric part of the scale similarity term. Notice how a second filtering operation needs to be introduced: it is somewhat similar to the test filter in the dynamic model, the main difference being that the it needs to be equivalent to the implicitly defined mesh filter, and hence apply the cutoff width $\overline{\Delta}$. This mixed model has several positive properties compared to the pure Smagorinsky model (and other eddy viscosity models that are able to produce positive values only), one of them being the allowance of negative values, i.e. backscatter (energy transferred backwards from the smaller to the larger scales, opposite of forward scatter) directly through the scale similarity term.

Further work on this Mixed Smagorinsky model was done by Zang et al. [44], where they applied the same method as Germano did for the Dynamic Smagorinsky model to arrive at a Dynamic Mixed Smagorinsky model. This derivation leads to an equation very similar to Eq. (3.43), now modified to include a second tensor H_{ij} as

$$L_{ij}^d = C_s^2 M_{ij} + H_{ij}^d, \tag{3.70}$$

where

$$H_{ij} = \widehat{\overline{\overline{u}}_i \overline{\overline{u}}_j} - \widehat{\overline{\overline{u}}}_i \widehat{\overline{\overline{u}}}_j. \tag{3.71}$$

Again some least squares analysis can be done, either the classic one as done by Lilly [35], or the Lagrangian Averaging approach, to arrive at an equation for C_s. This dynamic mixed model was further refined by Vreman et al. [45], who presented the idea of expressing the residual tensor at the test filter level in terms of $\widehat{\overline{u}}_i$ velocities only, and not $\overline{\overline{u}}_i$ velocities as done by [44]. This leads to the same expression as in Eq. (3.70), where the tensor H_{ij} now is modified to

$$H_{ij} = \widehat{\widehat{\overline{u}}_i \widehat{\overline{u}}_j} - \widehat{\widehat{\overline{u}}}_i \widehat{\widehat{\overline{u}}}_j - \left(\widehat{\overline{\overline{u}_i \overline{u}_j}} - \widehat{\overline{\overline{u}}_i \overline{\overline{u}}_j} \right). \tag{3.72}$$

Compared to the Dynamic Smagorinsky model the mixed variant requires specification of the “mesh filter” operation, here denoted with an overline. Applying the Smagorinsky model combined with the scale similarity term is just one possible solution, one could in theory apply whichever eddy viscosity model one wants to e.g. arrive at a mixed dynamic σ model, or a mixed dynamic WALE model.

3.4 Closing Notes Regarding LES

3.4.1 Boundary Conditions and Near Wall Challenges

When doing LES the applied boundary conditions are mostly equal to those presented in section 2.1.1. In general, as a method of triggering turbulent behaviour in the flow, one can perturb the initial and mean inlet boundary conditions to some extent and of some correct order, something that may help initiate turbulent flow phenomena in the simulation. This may sound like an easy thing do, but it is actually a rather challenging problem, as determining these perturbed components, their magnitudes, how they vary, and so on, is hard to do, especially in a completely general and simple way. For a good review on this subject, it is referred to [46].

As for the boundary conditions for walls, the traditional no-slip condition is often applied, then in combination with mesh grading towards the walls, or refinement of the mesh in the same regions. For flows containing boundary layers that are turbulent and followingly includes eddies, back-flow, extreme gradients and other chaotic phenomena on the macroscopic scale, LES has actually been proven quite problematic, and difficult. It is important that boundary layers of this type are sufficiently resolved as many of the effects occurring here are important for the outer flow itself, this eventually leading to a computational requirement close to DNS. As a result of this, to avoid this huge increase in computational requirement, some type of modelling of the flow in these regions is required when turbulent boundary layers are present. To cope with this problem there exists a couple of strategies, e.g. DES or other hybrid RANS-LES methods, briefly discussed in section 3.1.4. Another solution is to model the wall shear stress τ_w directly by some appropriate wall model, where again this shear stress is set as a boundary condition for the equations. This problem has not been emphasised in this work, it is instead referred to e.g. Piomelli [47] for an overview.

For a more thorough discussion regarding both boundary conditions and near-wall modelling, it is referred to Ch. 10 in Sagaut [5] and Ch. 13 in Pope [4].

3.4.2 State-of-the-Art

In despite of the massive research and improvements done the last decades, Large Eddy Simulation may still be said to be in its younger ages. However, as a consequence of the massive leap in speed, memory, and overall computational availability seen the last twenty years, LES has eventually become a tool not only available for those who possess, or have access to, super computer resources. As for industrial purposes, RANS is, without a doubt, still the most widely applied method of modelling turbulent, complex behaviour, as it comes packed with good efficiency, robust and well-researched models, and may handle simulations of high Re. Nevertheless, as LES has become more and more feasible the last years, the interest from the industry has also grown substantially.

Global problems with the LES procedure still exist; one of them being the near-wall challenge mentioned in the previous section, where wall modelling still is a topic of great interest and research. Other topics of debate includes the development of new and better subgrid-scale models, LES for compressible flows, turbulent combustion that includes chaotic phenomena at the subgrid-scale, research in terms of inlet boundary conditions, and more.

One of the most successful procedures when it comes to subgrid-scale modelling is still the dynamic procedure of Germano, it then being applied together with either the Smagorinsky model or some of the other differential operator models presented in this work. As discussed in section 3.3.4 a major drawback of this procedure is that the computed C_s may vary strongly in space and time, in addition to possible occurring negative values. Hence, refinement strategies as local averaging, clipping procedures and the Lagrangian averaging method has been introduced.

In 2006, as an improvement to the classic dynamic procedure, a new global dynamic equilibrium procedure, which was based on the global equilibrium hypothesis (da Silva et al. [48]), was introduced by Park et al. [49], then coupled with the differential operator subgrid-scale model of Vreman [50]. There a global instead of a local modelling constant is computed, resulting in the stability problems occurring in the classic dynamic approach being eliminated. Correct eddy viscosity behaviour, such as no contributions in wall-regions, is then accounted for through the differential operator of the coupled model. Refining of this model was further performed by You and Moin [51] in 2007, and by Singh and You [52] in 2013, with the addition of a scale-similarity term, leading to improved results for a selection of cases.

A slightly modified version of the Dynamic Smagorinsky model was also introduced by Park et al. [49] in the same paper. There the classic approach of Germano was applied together with the subgrid-scale model of Vreman, where averaging of the tensor contractions was done globally over the entire computational domain, instead of locally or in planes. Applying that method yielded better results for transitional flows (Lee et al. [53]), compared to those obtained with the global dynamic equilibrium model. This global method is actually coupled with the σ model in the original paper [33], where excellent results were obtained for a turbulent plane jet. As pointed out by [33], the subgrid-scale model that is coupled with such a global procedure is required to have the correct behaviour near walls, for two-dimensional flows and so on; hence, the Smagorinsky, WALE, and Vreman differential models are in fact unsuitable for this purpose, as none of them include all the required properties, whereas the σ model does.

LES is currently available through most commercial CFD packages as Ansys Fluent [7], Star-CD [8], OpenFOAM [9], and many more, making it available for a wider audience. One of the flagships in terms of Large Eddy Simulation is the Finite Volume method-based CDP project [54], developed by the LES group at the Center for Turbulence Research at Stanford University. The code has been, and still is applied for large LES simulations, where a milestone in terms of parallel computing was achieved in 2013 when their CharLES solver was run on over 1 million cores on IBM Sequoia, the third most powerful supercomputer available today [55].

The short overview given here is extremely brief, and do in no way justify LES and all its details. Especially when it comes to the subgrid-scale models, the focus has here been on eddy viscosity types only, though there exist a range of other approaches, as e.g. models which involves transport equations for the SGS stresses, filtered density function methods, deconvolution methods, implicit LES methods (where the numerical dissipation from the applied numerical scheme acts as an implicit subgrid-scale model), to mention a few. For different perspectives and further thoughts and discussion regarding LES and its current state it is referred to e.g. [56, 57, 58, 59, 60].

3.5 Conclusions

In this chapter the turbulence modelling method of Large Eddy Simulation, including a range of eddy viscosity models, both static and dynamic ones, have been thoroughly presented and discussed. Initially an introduction to DNS was given, together with a short discussion and justification on why turbulence modelling is severely needed, such that computations of turbulent flows may be feasible. The first proposed method, and still the most popular one today, is the so-called RANS approach, where the Navier-Stokes equations are recast into PDEs for ensemble averaged values $\overline{u}_i$ and $\overline{p}$. Large Eddy Simulation was presented as a midpoint between RANS and DNS, the method being motivated by RANS' problems in that its turbulence models must model a large range of effects, whereas the LES models only need to model a smaller portion of the flow. The hybrid RANS-LES method Detached Eddy Simulation was briefly discussed, it combining the best of the LES and RANS worlds into one good approach.

The formal parts of Large Eddy Simulation has been thoroughly discussed through the definition of the filtering operator, the derivation of the filtered Navier-Stokes equations, and decomposition of the subgrid-scale tensor τ_{ij}^r. This was followed by the presentation of the general assumption in that all the subgrid-scale effects may be lumped into a turbulent eddy viscosity term, which again needs to be modelled by appropriate terms. Models for this eddy viscosity was introduced, starting with the classic Smagorinsky model, followed by refined differential operator models in WALE and the σ model, continuing with the Dynamic Smagorinsky model and its refined Lagrangian averaged and scale-dependent companions, and wrapping it up with a model based on the subgrid-scale kinetic energy, and some details regarding scale-similarity / mixed models.

As a last part of this chapter, some details regarding boundary and initial conditions together with wall problems were discussed, including an extremely brief presentation of state-of-the-art in LES. It is referred to a range of papers for a more thorough discussion regarding state-of-the-art and current challenges in the LES field.

Chapter 4

Discretisation and Implementation

4.1 Spatial Discretisation of the Eddy Viscosity Contribution

Our first task is to discretise the new residual stress tensor term in Eq. (3.17) by FEM, then segregate it such that it can be combined with the approach presented in section 2.2.3. When Eqs. (2.17) and (2.18) are recast into equations for the filtered variable $\overline{u}_i$ and $\overline{p}$, the residual stress tensor will lead to a new right hand side contribution to both of them, but it is eventually only the equation for the tentative velocity Eq. (2.18) that is altered. Both the pressure Poisson equation and the velocity update step in the Incremental Pressure Correction Scheme (IPCS) are unchanged.

To proceed we now apply the standard FEM recipe on the residual stress tensor; that is, multiply it by a test function v, and integrate it over the whole domain. It is also required that the term should be valid for time step $n+1/2$ such that it is consistent with the Crank-Nicolson approach applied with IPCS. This leads to

$$\tau^r_{ij,\text{weak}} = -\left\langle \frac{\partial \tau^r_{ij}}{\partial x_j}, v \right\rangle^{n+1/2}, \tag{4.1}$$

which is added to the right hand side of Eq. (2.26) (where Eq. (2.26) now are partial differential equations (PDEs) for the filtered variables). We now insert the eddy viscosity expression from Eq. (3.23) to obtain

$$\begin{aligned} \tau^r_{ij,\text{weak}} = -\left\langle \frac{\partial \tau^r_{ij}}{\partial x_j}, v \right\rangle^{n+1/2} &= \left\langle \frac{\partial}{\partial x_j}\left(\nu_T \left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right)\right), v \right\rangle^{n+1/2} \\ &= \left\langle \frac{\partial}{\partial x_j}\left(\nu_T \frac{\partial \overline{u}_i}{\partial x_j}\right), v \right\rangle^{n+1/2} + \left\langle \frac{\partial}{\partial x_j}\left(\nu_T \frac{\partial \overline{u}_j}{\partial x_i}\right), v \right\rangle^{n+1/2}. \end{aligned} \tag{4.2}$$

As mentioned when the eddy viscosity assumption was presented, the trace term of τ^r_{ij} is combined with the pressure. This has not been strictly written here, but it is done "behind scenes" when we insert for τ^r_{ij}. Again, as seen from Eq. (4.2), we encounter the problem of coupled velocity components, where it is evident that the term $\frac{\partial \overline{u}^\star_j}{\partial x_i}$ enters the equation for component $\overline{u}^\star_i$. When the convective term in the IPCS scheme was discretised, two problems were solved as a result of the appliance of the Adams-Bashforth scheme: the term was linearised, and it was decoupled in that the equation for $\overline{u}^\star_i$ became independent of the coupled velocity terms $\overline{u}^\star_j$. For this case there are no nonlinearities, we, however, need to avoid the coupled approach, thus the Adams-Bashforth projected velocity will be applied for the last term that contains $\overline{u}^\star_j$.

Starting with the first term on the right hand side of Eq. (4.2) we insert the Crank-Nicolson midpoint approximation, then integrate it by parts (and remove the surface integral term), insert the discrete representation of the velocities from Eq. (2.29), segregate it, and write it on matrix notation. This eventually results in new left and right hand side contributions to all the tentative velocity equations as

$$\left\langle \frac{\partial}{\partial x_j}\left(\nu_T \frac{\partial u_i}{\partial x_j}\right), v\right\rangle^{n+1/2} = \begin{cases} -\frac{1}{2}K_{ij,\nu_T}\left(\overline{U}^\star_j + \overline{U}^n_j\right) & \text{for } \overline{U}^\star_j \\ -\frac{1}{2}K_{ij,\nu_T}\left(\overline{V}^\star_j + \overline{V}^n_j\right) & \text{for } \overline{V}^\star_j \\ -\frac{1}{2}K_{ij,\nu_T}\left(\overline{W}^\star_j + \overline{W}^n_j\right) & \text{for } \overline{W}^\star_j \end{cases} \tag{4.3}$$

where

$$K_{ij,\nu_T} = \left\langle \widetilde{\nu}_T \frac{\partial \phi_i}{\partial x_k}, \frac{\partial \phi_j}{\partial x_k}\right\rangle, \tag{4.4}$$

for $\widetilde{\nu}_T = \nu_T^{n+1/2}$. The matrix K_{ij,ν_T} is quite similar to the stiffness matrix K_{ij} in Eq. (2.33), the difference is that $\nu_T = \nu_T\left(x, y, z, t\right) \neq \text{const}$. This means that since ν_T is changing with time it must be a part of the integration and the matrix assembly; hence, the matrix K_{ij,ν_T} is changing with time, and therefore needs to be assembled each time step. This leads to an extra computational requirement, which, unfortunately, cannot be avoided. It, however, only needs to be assembled once each time step, similar to the convective matrix X_{ij}, as a result of both of these matrices being common for the three equations for $\overline{U}^\star_j$, $\overline{V}^\star_j$ and $\overline{W}^\star_j$. From Eq. (4.3) it is clear that one gets a general contribution to the common left and right hand sides of the linear systems to be solved, this contribution being nearly identical to the one of the viscous part in Eq. (2.31).

As for the second part on the right hand side of Eq. (4.2), we apply the Adams-Bashforth projection to avoid the coupling problem. Before we proceed, it should be noted that this term caused some stability problems when discretised by the same procedure as the one applied for the previous term. This may have something to do with the fact that discretisation of these viscous terms normally should be done fully implicit, or semi-implicit, e.g. by applying the Crank-

Nicolson method, and not fully explicit as with the Adams-Bashforth projection. If they are discretised explicitly as a contribution to the linear systems right hand sides only, stability issues are often encountered, and one gets a stricter criterion on the time step Δt. In the start-up phase of the work with this thesis, a simpler Navier-Stokes solver applying a vector coupled approach was used, where the solver then handled the velocities as vectors directly, and not separately, as *Oasis* does through the segregated approach. When such a coupled solver is applied, one does not need to introduce tricks to handle coupled velocity components, resulting in the Crank-Nicolson approach also being applicable for the coupled eddy viscosity term. For that solver no stability problems were encountered, most likely because of the term having beeng discretised in a semi-implicit way.

To cope with this stability problem a couple of solutions was tested. First a type of mixed Crank-Nicolson/Adams-Bashforth approach was implemented, where a Crank-Nicolson discretisation was applied for the velocity component where $i = j$ (i and j are now vector indexes), and the Adams-Bashforth projection was applied for the two other coupled components where $i \neq j$. This could be done as a result of one of the components in $u_j^{n+1/2}$ always being equal to the velocity component that is solved for, when $i = j$ that is. This approach resulted in better stability, but it was too demanding in terms of required computational time, then as a result of the Crank-Nicolson method being a semi-implicit method, followingly leading to both right and left hand side contributions when $i = j$. The left hand side contribution required assembly of an additional matrix; hence, since this is an extremely expensive operation, it was too demanding in terms of extra required computational time, subsequently leading to this semi Crank-Nicolson/Adams-Bashforth being abandoned.

As a second alternative an approach just as simple, and with the same computational requirements, as the unstable approach, was tested. This method did eventually result in better stability compared to the "original" approach (integration by parts), in addition to it leading to right hand side contributions only, requiring no extra assembly of an additional left hand side matrix. The idea is to rewrite the corresponding term as

$$\frac{\partial}{\partial x_j}\left(\nu_T \frac{\partial \overline{u}_j}{\partial x_i}\right) = \frac{\partial \nu_T}{\partial x_j}\frac{\partial \overline{u}_j}{\partial x_i} + \nu_T \frac{\partial^2 \overline{u}_j}{\partial x_j \partial x_i} = \frac{\partial \nu_T}{\partial x_j}\frac{\partial \overline{u}_j}{\partial x_j}, \tag{4.5}$$

where the last equality follows as a result of interchangeable differentiation and the continuity equation $\frac{\partial \overline{u}_j}{\partial x_j} = 0$. For the last term in Eq. (4.2) this yields

$$\left\langle \frac{\partial}{\partial x_j}\left(\nu_T \frac{\partial \overline{u}_j}{\partial x_i}\right), v \right\rangle^{n+1/2} = \left\langle \frac{\partial \nu_T}{\partial x_j}\frac{\partial \overline{u}_j}{\partial x_i}, v \right\rangle^{n+1/2} = \left\langle \frac{\partial \widetilde{\nu}_T}{\partial x_j}\frac{\partial \overline{u}_{j,AB}^{n+1/2}}{\partial x_i}, v \right\rangle. \tag{4.6}$$

Segregating the last term in Eq. (4.6) and writing out the summations now yield the following right hand side contributions to the tentative velocity equations:

$$b_{\overline{U}^\star,\nu_T} = \left\langle \frac{\partial \widetilde{\nu}_T}{\partial x}\frac{\partial \overline{u}_{AB}^{n+1/2}}{\partial x} + \frac{\partial \widetilde{\nu}_T}{\partial y}\frac{\partial \overline{v}_{AB}^{n+1/2}}{\partial x} + \frac{\partial \widetilde{\nu}_T}{\partial z}\frac{\partial \overline{w}_{AB}^{n+1/2}}{\partial x}, v \right\rangle, \tag{4.7}$$

$$b_{\overline{V}^\star,\nu_T} = \left\langle \frac{\partial \widetilde{\nu}_T}{\partial x}\frac{\partial \overline{u}_{AB}^{n+1/2}}{\partial y} + \frac{\partial \widetilde{\nu}_T}{\partial y}\frac{\partial \overline{v}_{AB}^{n+1/2}}{\partial y} + \frac{\partial \widetilde{\nu}_T}{\partial z}\frac{\partial \overline{w}_{AB}^{n+1/2}}{\partial y}, v \right\rangle, \tag{4.8}$$

$$b_{\overline{W}^\star,\nu_T} = \left\langle \frac{\partial \widetilde{\nu}_T}{\partial x}\frac{\partial \overline{u}_{AB}^{n+1/2}}{\partial z} + \frac{\partial \widetilde{\nu}_T}{\partial y}\frac{\partial \overline{v}_{AB}^{n+1/2}}{\partial z} + \frac{\partial \widetilde{\nu}_T}{\partial z}\frac{\partial \overline{w}_{AB}^{n+1/2}}{\partial z}, v \right\rangle. \tag{4.9}$$

Since these terms are fully explicit they do not need to be computed by a matrix-vector product, but can be assembled as right hand side contributions directly; hence, we will not insert the decomposed velocities here. This method results in better overall stability (though not as good as for the mixed Crank-Nicolson/Adams-Bashforth approach), in addition to it having no extra requirements in terms of additional assembly procedures or similar, hence it has been chosen for this implementation.

When it comes to $\widetilde{\nu}_T$, it needs to be computed each time step such that it is valid at time location $n + 1/2$. Since

$$\nu_T = \nu_T\left(\overline{u}_i\left(t\right)\right), \tag{4.10}$$

for all the LES models to be applied in this thesis (except the one based on the subgrid-scale kinetic energy), we have that

$$\widetilde{\nu}_T = \nu_T\left(\overline{u}_i^{n+1/2}\right). \tag{4.11}$$

Since the Crank-Nicolson scheme is not applicable here ($\overline{u}_i^\star$ is unknown) we instead apply the Adams-Bashforth projection $\overline{u}_{i,AB}^{n+1/2}$ when computing $\widetilde{\nu}_T$, such that it is computed for the correct time location.

4.2 Implementation of the Eddy Viscosity Contribution

As seen through Eq. (2.31) the matrices in front of $\overline{U}_j^\star$ and $\overline{U}_j^n$ (and the two other velocity components) are nearly identical. This fact is exploited by *Oasis* which computes a general matrix A_{ij} that is first applied for the right hand side of the linear system, then it is multiplied by -1 and $\frac{2}{\Delta t}M_{ij}$ is added to it, such that the general left hand side matrix is obtained. As for the addition of the LES terms two things need to be done:

1. Add the common Crank-Nicolson term from Eq. (4.3) to the general system matrix A_{ij}.
2. Add the Adams-Bashforth terms from Eqs. (4.7) to (4.9) to the corresponding right hand sides.

We first start with assembling the required matrix K_{ij,ν_T} from Eq. (4.3), and then add it to the general system matrix A_{ij}. Now assume that we in *Oasis* enter a new step in the time loop, then some functionality is executed which eventually results in the matrix A_{ij} being computed as

$$A_{ij} = \left(\frac{1}{\Delta t} M_{ij} - \frac{1}{2}\nu K_{ij} - \frac{1}{2} X_{ij} \right), \tag{4.12}$$

which is equivalent to the right hand matrix seen in front of $\overline{U}^n$ in Eq. (2.31). The only work left to do is to add the required LES term such that A_{ij} becomes equal to

$$A_{ij} = \left(\frac{1}{\Delta t} M_{ij} - \frac{1}{2}\nu K_{ij} - \frac{1}{2} X_{ij} - \frac{1}{2} K_{ij,\nu_T} \right), \tag{4.13}$$

This is done each time step as

```python
# Assemble K_nut
K_nut = assemble(nut*inner(grad(u),grad(v))*dx)
# Add to Aij
Aij.axpy(-0.5, K_nut)
```

First *K_nut* is assembled by applying the *assemble* function on its UFL form; notice how the notation is very similar to the mathematical of description of this matrix in Eq. (4.4). u and v are here the trial and test functions for the velocity function space, and *nut* is a general FEniCS function which depends on what type of LES model the user has chosen. The operation *axpy* (works both for vectors and matrices) can be translated to

$$Aij.axpy(-0.5, K\_nut) \Leftrightarrow A_{ij} \rightarrow A_{ij} - 0.5 K_{ij,\nu_T}. \tag{4.14}$$

Next we compute the contributions to the right hand sides applying this general matrix A_{ij} as

```python
# Loop over right hand sides
for i,comp in enumerate(['u', 'v', 'w']):
    # Compute general contribution and add to corresponding rhs
    b[comp].axpy(1.0, Aij*u_[i].vector())
```

where b is a Python dictionary containing the right hand sides, and $u\_$ is a list containing the velocities at the previous time step. Now A_{ij} is multiplied by -1, then $\frac{2}{\Delta t} M_{ij}$ is added to it, such that the common left hand side matrix is obtained as

$$A_{ij} = \left(\frac{1}{\Delta t} M_{ij} + \frac{1}{2}\nu K_{ij} + \frac{1}{2} X_{ij} + \frac{1}{2} K_{ij,\nu_T} \right). \tag{4.15}$$

The Crank-Nicolson terms in Eq. (4.3) have now been added both to the matrix A_{ij}, and to the three right hand side vectors in b, hence our first task is done.

The next step is now to add the Adams-Bashforth contributions from Eqs. (4.7) to (4.9) to the components in b, followingly finalising the assembly of the right hand sides. As a result of *nut* being time dependent, these terms need to be computed each time step; however, instead of assembling large matrices and then do matrix-vector products, we assemble the right hand side vector contributions directly as it is a much more efficient solution. Direct assembly of the right hand side contribution-vectors is done as

```
# Loop over velocity components and to right hand sides
for i,comp in enumerate(['u', 'v', 'w']):
    # Assemble rhs contribution for given velocity component
    LES_AB = assemble(inner(inner(grad(nut), u_ab.dx(i)), v)*dx)
    # Add to corresponding right hand side vector
    b[comp].axpy(1.0, LES_AB)
```

The vector *u_ab*, containing all the Adams-Bashforth projected velocities, is provided by the solver framework. To compute the different right hand side contributions, here named *LES_AB*, we start by taking the inner product between the gradient of *nut* and *u_ab.dx(i)*, where *u_ab.dx(i)* here means differentiation of each component in the vector *u_ab* with respect to spatial component i. This expression is then multiplied by the test function v, *LES_AB* is assembled, and followingly added to the corresponding right hand side. To elaborate some more on the $u\_ab.dx(i)$ expression; for example for the first iteration we have $i = 0$ and *comp* = *'u'*, thus the *LES_AB* contribution is a function of the inner product between the two vectors $\frac{\partial \nu_T}{\partial x_j}$ and $\left[\frac{\partial \overline{u}_{AB}^{n+1/2}}{\partial x}, \frac{\partial \overline{v}_{AB}^{n+1/2}}{\partial x}, \frac{\partial \overline{w}_{AB}^{n+1/2}}{\partial x}\right]$, which is consistent with what is presented in Eq. (4.7). Further *LES_AB* is assembled, and then added to the corresponding right hand side vector $b['u']$ for the component $\overline{U}_j^\star$. For the two following iterations $i = 1$ and $i = 2$ we get similar contributions to the right hand sides for components $\overline{V}_j^\star$ and $\overline{W}_j^\star$, where *u_ab* then is differentiated with respect to y and z, respectively.

Now, as a result of both the general system matrix A_{ij} and the specific right hand sides in b having been modified to include the LES contributions, we may pick a wanted LES model, compute the corresponding *nut*, and followingly start to solve for the filtered velocity vector. If no LES model is specified to the solver the eddy viscosity is simply set to be equal to zero, eventually resulting in no turbulence modelling and Direct Numerical Simulations (DNS) instead.

4.3 Implementation of the LES Models

The next step in the process is now to implement the different LES models presented in Ch. 3 into the solver framework. Implementational details for the

Smagorinsky model, the WALE model, the σ model, the Dynamic Smagorinsky model, the Scale-Dependent Dynamic Smagorinsky model, and the Kinetic-Energy SGS model will in the following sections be presented. Simple code snippets will be shown for crucial parts of the implementations, whereas the more general parts of the implementations will be left out. All the details can be found in the source code, available online at [21], where the implementations of the LES models are located in the folder **/solvers/NSfracStep/LES/**.

When it comes to user activation of the models a simple argument is passed through *Oasis*' *NS_parameters* dictionary, where the argument specifies which LES model one wants to apply. Currently on may select the following models

```
les_models = [None, "Smagorinsky", "Wale", "Sigma",
              "DynamicLagrangian", "ScaleDepDynamicLagrangian",
              "KineticEnergySGS"]
```

where one activates a LES model as

```
# Activate LES model, les_model = None by default
NS_parameters.update(les_model = "Smagorinsky")
```

In addition to this the constants applied in some of the models, e.g. C_w for WALE, may be controlled by the user as

```
NS_parameters["Wale"].update(Cw = 0.6)
```

or for the Dynamic Smagorinsky model one has the choice of updating C_s f.ex. each 10th time step as

```
NS_parameters["DynamicSmagorinsky"].update(Cs_comp_step = 10)
```

The implementation has been designed such that it may be simple to activate a LES model, in addition to the user having the ability to easily tune some of the parameters of the selected model.

4.3.1 The Smagorinsky Model

Starting with the Smagorinsky model, the length scale term $\overline{\Delta}$ can be obtained in FEniCS through its UFL function $CellVolume$ as

```
dim = mesh.geometry().dim()
delta = CellVolume(mesh)**(1./dim)
```

where $CellVolume$ now returns the symbolic volume for each cell in the mesh, and *dim* equals the geometrical dimension of the problem. Further the term $|\overline{S}|$ needs to be specified. For computation of the rate of strain tensor we apply the two UFL functions *grad* and *sym* defined as

$$grad\,(u_i) = \frac{\partial u_i}{\partial x_j}, \tag{4.16}$$

for a vector u_i, and

$$sym\left(A\right) = \frac{1}{2}\left(A + A^T\right), \tag{4.17}$$

for some matrix A. When these two functions are combined $\overline{S}_{ij}$ may be defined as

$$\overline{S}_{ij} = sym\left(grad(\overline{u})\right) = sym(\frac{\overline{u}_i}{\overline{x}_j}) = \frac{1}{2}\left(\frac{\overline{u}_i}{\overline{x}_j} + \frac{\overline{u}_j}{\overline{x}_i}\right), \tag{4.18}$$

translated to UFL code in Python as

```
Sij = sym(grad(u_ab))
```

where *u_ab* (as mentioned earlier) is a vector function of rank 1 containing the Adams-Bashforth projected velocity components, and *Sij* a UFL form of rank 2 (second rank tensor). The magnitude of $\overline{S}_{ij}$ can be obtained applying the UFL inner product function *inner* and the square root function *sqrt* as

```
magS = sqrt(2*inner(Sij,Sij))
```

where, due to contraction of both indices in S_{ij}, *magS* now represents a UFL form of rank 0. The final UFL form of rank 0 for *nut* can now be specified by the Python variables *Cs*, *delta* and *magS* as

```
nut = Smagorinsky["Cs"]**2 * delta**2 * magS
```

where C_s is accessed through the Smagorinsky dictionary stored in the solver. All these steps are performed in the preprocessing, before the time stepping starts.

The last step in the procedure is now to actually compute *nut*; as it stands now it is a UFL form only containing no explicit values. Computation of it is done by some fast projection method where a linear system is solved for *nut* each time step, followed by *nut* being sent to the solver framework. In addition to the projection of *nut* a type of bounding procedure is introduced, such that negative values (artefacts) occurring as a result of the linear system procedure are removed.

4.3.2 The WALE Model

The WALE model in Eq. (3.33) is implemented similarly to the Smagorinsky model, where some extra ingredients now are required for the UFL specification of the term D_W. The LES length scale $\overline{\Delta}$ is computed the same way as for the Smagorinsky model, and the constant C_w is by default defined as 0.325 in a *Wale* dictionary stored in the solver. If desired by the user, the constant may be changed to whatever value is wanted.

When defining D_W we first set up the UFL forms for $G_{ij} = \frac{\overline{u}_{i,AB}}{\overline{x}_j}$ and $\overline{S}_{ij}$ as

```
Gij = grad(u_ab)
Sij = sym(Gij)
```

To specify the tensor S_{ij}^d some extra UFL ingredients are required: the function tr which returns the trace of a tensor, e.g. as $tr(\overline{S}_{ij})$; and the function $Identity$ which returns the identity matrix. S_{ij}^d and its UFL form can now be specified as

```
Sd = sym(Gij*Gij) - (1./dim)*Identity(dim)*tr(Gij*Gij)
```

which clearly is consistent with the definition of D_W given in Eq. (3.31). The final UFL form for D_W can now be defined applying the function $pow(a, b)$, which mathematically translates to a^b, and the function $inner(a, b)$. This results in

```
Dw = pow(inner(Sd,Sd), 1.5) / (pow(inner(Sij,Sij), 2.5) \
          + pow(inner(Sd,Sd), 1.25)
```

Now the UFL form for nut can be defined as

```
nut = Wale["Cw"]**2 * delta**2 * Dw
```

where, as with Smagorinsky, the constant C_w also here is accessed through the Wale dictionary.

As with the Smagorinsky model nut is now computed each time step, and followingly passed to the solver.

4.3.3 The σ Model

Compared to the implementations of the Smagorinsky and WALE models, where both of them easily could be coded by just defining the appropriate UFL forms, the σ model requires some more advanced ingredients in its implementation. Since the singular values of the velocity gradient tensor is required, the discrete velocity tensor must be explicitly computed, then one needs some type of loop, which iterates over each node in the mesh, and computes the local gradient matrix and its singular values there. The singular values are then put into the expression for ν_T, where ν_T subsequently is projected and applied by the solver.

First, we start by computing all the components of the velocity-gradient tensor. This involves solving either one large linear system of equations applying FEniCS' TensorFunctionSpace where one may operate with tensors directly, or by solving nine smaller and simpler systems of equations applying some scalar function space. This may be done extremely fast if we apply the Discontinuous Galerkin (DG) function space of order zero, where each cell in the mesh contains one computational node in its midpoint, and the basis function for each node is equal to one for that node, and zero else. Applying this type of function space eventually leads to the FEM mass-matrix $M_{ij} = \langle \phi_i, \phi_j \rangle$ becoming diagonal, which again results in its inverse being trivially computed as $\frac{1}{A_{ij}}$. Applying this space is, however, quite inaccurate, as the number of nodes in the mesh becomes drastically reduced, and the overall accuracy (e.g. in wall regions) becomes bad. To attain the desired accuracy in an efficient way, the usual linear P_1 Lagrange elements will instead be applied.

First the left hand side mass matrix and the right hand side “derivative”-matrices required to solve for the velocity gradients are pre-assembled before the time stepping as

```
# Extract Trial and Test functions for P1 Lagrange space CG1
p,q = TrialFunction(CG1), TestFunction(CG1)
# Assemble left hand side mass matrix
A = assemble(inner(p,q)*dx)
# Assemble right hand side derivative matrices
b_mats = [assemble(inner(u.dx(i),q)*dx for i in range(dim)]
```

The three “derivative”-matrices in *b_mats* are defined as

$$A_{ij,x} = \int_\Omega \frac{\partial \psi_j}{\partial x}\phi_i dx, \tag{4.19}$$

$$A_{ij,y} = \int_\Omega \frac{\partial \psi_j}{\partial y}\phi_i dx, \tag{4.20}$$

$$A_{ij,z} = \int_\Omega \frac{\partial \psi_j}{\partial z}\phi_i dx, \tag{4.21}$$

where ψ_j are the basis functions for the velocity space, and ϕ_i are the basis functions for the linear P_1 space. Now, to obtain e.g. the x-derivative of w we compute the right hand side as $b_{i,wx} = A_{ij,x}w_j$, pass it to the solver together with the left hand side matrix A, eventually obtaining the discrete horizontal derivative of w. Similarly, if we want to compute e.g. $\frac{\partial u}{\partial y}$, we compute the right hand side as $b_{i,uy} = A_{ij,y}u_j$. Similar for all the other seven components of the velocity gradient tensor. By doing this we avoid any type of matrix or vector assembly during time stepping, thus saving valuable computational time. A specific linear algebra solver named *grad_solver* is also defined during the pre-processing, such that maximum performance when solving the linear systems is obtained. During the time stepping the velocity gradients are then obtained as

```
# Loop over each tensor component
for k in range(tensdim):
    # Extract i,j for given tensor component
    i,j = ij_pairs[k]
    # Solve for component at location i,j
    grad_solver.solve(A, g[k], b_mats[j]*u_ab[i].vector())
```

where the list $g[k]$ now contains the velocity gradients as scalar fields. The variables i and j extracted from $ij\_pairs[k]$ tell us which tensor component we are solving for, e.g. (0,0) or (0,1), and so on. *tensdim* equals the number of components in the gradient tensor, equal to 9 in three dimensions.

The next step is now to construct local gradient matrices at each node in the mesh, followed by the Singular Value Decompositions (SVDs) of all these matrices. As a first solution this was done by getting the local number of nodes for each process, then a loop over each node was initiated, where a local gradient matrix G of size 3×3 was computed at each location. Singular values for G

were then computed with the linear algebra package of NumPy [61]. Though this approach worked as intended, a much faster solution was obtained by applying the functions *concatenate*, *reshape* and *transpose* of NumPy, together with the available vectorized functionality of NumPy's SVD function. First, by applying the *concatenate* function all the gradient arrays in g are combined into one huge array, where the first Q entries are those found in $g[0]$, followed by all the R entries found in $g[1]$, and so on. This array is then reshaped and transposed such that we obtain a large array of size M (M is here equal to the local number of nodes for the corresponding process), where we for each entry in this array now find a local gradient matrix of size 3×3, containing all the nine gradients at each specific node in the mesh. The SVD function of NumPy is then called, where this array of gradient matrices is sent directly into NumPy, which then computes the SVD of each matrix in this array, followingly returning an array of size M containing three σs at each index. This can be coded as

```
# Create array of local gradient matrices by fast NumPy
    vectorization
G = NumPy.concatenate(tuple([gij[k].array() for k in
    xrange(tensdim)])).reshape(dim,dim,-1).transpose(2,0,1)
# Solve for Singular values of all matrices in G simultaneously
sigmas = NumPy.linalg.svd(G,compute_uv=False)
```

Notice the flag $compute\_uv = False$ telling the SVD-function of NumPy to only compute the singular values, and not the matrices U and V in the decomposition, followingly saving some computational time. The singular values in *sigmas* are then extracted and added to the corresponding FEniCS functions $sigma1$, $sigma2$ and $sigma3$, where each of these functions will be a scalar field containing the corresponding singular value at each computational node. This solution is extremely fast, avoiding the slow python loop over each node in the mesh, and vectorizing the singular value decomposition of each local gradient matrix.

As a last step the UFL forms for the differential operator and the eddy viscosity is coded as

```
D_sigma = sigma3 * (sigma1 - sigma2) * (sigma2 - sigma3) /
    sigma1**2
nut = Sigma["Cs"]**2 * delta**2 * D_sigma
```

where C_σ (here named C_s) is accessed through the Sigma dictionary. In addition to this a variable *comp_step* is available in the Sigma-dictionary, where the user have the ability to specify how often the singular values are to be updated; e.g. if $comp\_step = 10$ they are updated each 10th time step. This choice does not affect computation of *nut* since it is explicitly computed each time step, then by reusing the previously computed singular values. If wanted one may update D_σ each time step, but this would, nevertheless, lead to a substantial increase in computational time, especially for larger meshes. Thus, some selected value for *comp_step* should be presented by the user.

4.3.4 The Dynamic Smagorinsky Model

Compared to the three previous models, implementation of the Dynamic Smagorinsky model requires some additional ingredients. Where the Smagorinsky and WALE models required a specification of *nut* through some UFL forms only, the dynamic model needs a test filtering operation, explicit computation of L_{ij} and M_{ij}, and Lagrange averaging of $L_{ij}M_{ij}$ and $M_{ij}M_{ij}$, before the UFL form for *nut* finally can be projected and sent to the solver. As this algorithm requires some additional amount of computational time, a lot of effort and time has been spent on optimization, avoiding FEniCS' automated functionality in favour of faster, better and more "manual" solutions.

First we need to specify a filtering operation $\widehat{()}$, which takes the once implicitly filtered velocity vector $\overline{u}_i$ to the test filtered level $\widehat{\overline{u}}_i$. In general specifying such a filter on unstructured meshes is difficult: where one on structured meshes more easily may specify a region of integration for each cell, it is not as easily defined on unstructured meshes where neighbouring cells usually have different size. One natural solution suggested by Volker [62], is to solve an additional set of the Navier-Stokes equations on a coarser grid. While the idea may be good, it is not feasible in terms of computational cost and power compared to simpler solutions as e.g. approximations to the filtering integrals or similar weighting procedures. One idea initially tested for this thesis was to apply a double mesh solution, where two meshes were provided by the user, then some procedure was applied to interpolate $\overline{u}_i$ to the coarser mesh and hence obtain $\widehat{\overline{u}}_i$. This method was also problematic, as the interpolation methods used to transfer the functions between the meshes led to severe numerical effects and artefacts. In addition, it was impractical, as the user had to create and provide two meshes to the solver.

Two different solutions have been found and tested here, one of them suggested by Jansen [63], the other mentioned by [64] as a product of the work of Germano [65]. A derivative based filter applying the Laplace operator defined by [63] was also investigated and implemented, but the obtained results were unsatisfactory.

The Generalized Top Hat Filter

The first method is the so-called generalized top hat filter (GTHF) applied by Jansen [63], defined as

$$\widehat{\overline{u}} = \frac{\int_\Omega G\overline{u}dx}{\int_\Omega Gdx}, \tag{4.22}$$

where G further is defined as a piecewise function constant equal to 1 for a vertex n and for all vertices in elements that contain vertex n, and 0 for all other vertices in the domain. See Fig. 4.1. When one averages or filters the value for the middle node, all the marked nodes are taken into account. For the specific figure

G becomes equal to 1 for all the marked nodes, 0 else, somewhat representing a type of filter kernel that is similar to that of the box filter in Eq. (3.7).

This type of function space is currently unavailable through FEniCS, thus G is in this implementation set to be equal to the piecewise linear basis functions for the P_1 element. These basis functions differs from the definition of G, but they are similar in that the exact same node groups as in Fig. 4.1 are covered, hence, the resulting filtered fields applying these functions would most likely be in the vicinity of those one would have obtained if the original formulation of G had been applied. One problem that arises when applying the P_1 basis functions, is the fact that the overall weighting procedure puts too much emphasis on the middle node compared to the outer nodes, possibly representing the filtering kernel $\overline{G}$ better than it represents $\widehat{G}$. If the piecewise constant function for G had been used, the outer nodes would have been weighted much more, something which would have resulted in heavier filtering overall.

If we as mentioned apply the P_1 basis functions as $G = \phi_i$, filtering of a generic scalar field $c = \sum_{j=0}^{N} C_j \phi_j$ is then done as

$$\widehat{c} = \frac{G_{ij} C_j}{G_i^\star}, \tag{4.23}$$

where

$$G_{ij} = \langle \phi_i, \phi_j \rangle , \tag{4.24}$$

and

$$G_i^\star = \int_\Omega \phi_i dx. \tag{4.25}$$

Now both G_{ij} and $G_i^\star$ can be computed during the pre-processing, and the filtering operation can be carried out as a simple matrix-vector product during the time

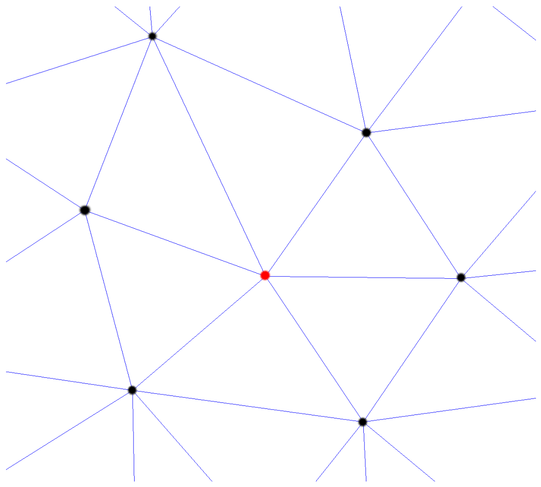


Figure 4.1: Nodes included in the weighting of the middle node with GTHF.

stepping iteration. It is important that the filtering operation is fast since it is to be applied several times in the algorithm for C_s ($\sim$ 20 times in total per algorithm call); this is clearly achieved with this method since pre-processing of the two terms G_{ij} and $G_i^\star$ is possible, where again the filtered values are obtained by fast matrix-vector products only.

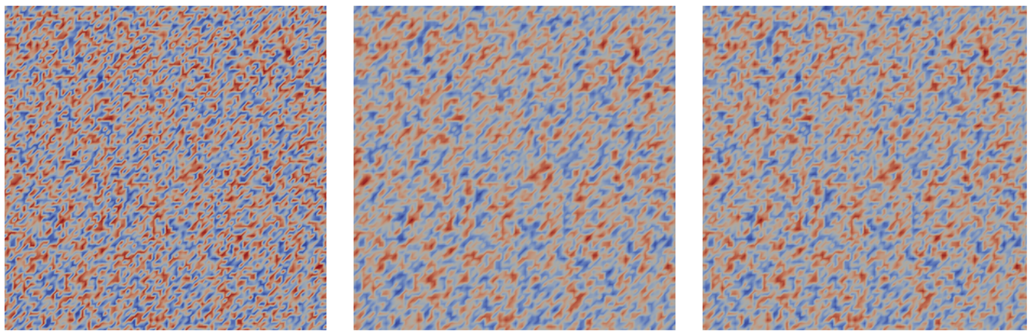
The Inverse Helmholtz Filter

The inverse Helmholtz filter (IHF) presented by Germano [65] and applied by Bull et al. [64], is a differentiable type of filter that may be derived as a result of a special exponential Gaussian filter kernel being applied. Germano [65] shows that a PDE given as

$$\overline{f} = \widehat{\overline{f}} - \nabla \cdot \left(a \nabla \widehat{\overline{f}} \right), \tag{4.26}$$

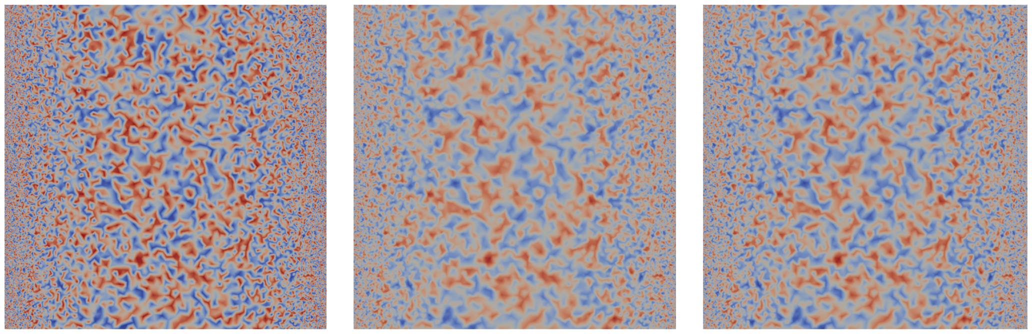
may be solved for the test filtered field $\widehat{\overline{f}}$. a is defined as $a = \frac{\left(\alpha\overline{\Delta}\right)^2}{24}$, where α here is the same α as was seen in M_{ij} for the Dynamic Smagorinsky model.

One drawback of this method is that a PDE, and some subsequent linear system, needs to be solved in order to obtain the filtered field. As the PDE is linear, in addition to it having no coefficients that are time-dependent, the linear system may be assembled during pre-processing, such that the only required operations when filtering are to compute the right hand side matrix-vector product and solve the linear system. Compared to the GTHF the operation there is much cheaper as it consists of a matrix-vector product only. In addition, comparing results shows that the differences between the obtained filtered fields for these two methods are small; hence, the GTHF has been further applied here. See Figs. 4.2 to 4.4 for some results.



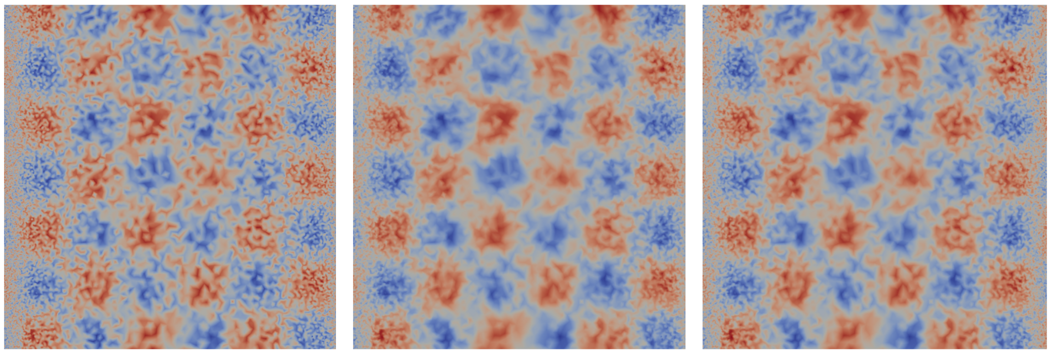
(a) Unfiltered. **(b)** GTHF. **(c)** IHF.

Figure 4.2: Unfiltered vs. filtered random field on a structured mesh.



(a) Unfiltered. **(b)** GTHF. **(c)** IHF.

Figure 4.3: Unfiltered vs. filtered random field on an unstructured mesh.



(a) Unfiltered **(b)** GTHF. **(c)** IHF.

Figure 4.4: Unfiltered vs. filtered random perturbed standing waves on an unstructured mesh.

Computation of L_{ij}

The next step would now be to compute the two symmetric tensors L_{ij} and M_{ij}. Starting with L_{ij}, it consists of velocity outer products only, a mathematical operation accessible through FEniCS' UFL function *outer*. However, since any UFL form needs to be projected (solving a linear system) before its values can be used, this method has been abandoned in favour of the more efficient, and simpler, route where all the terms in L_{ij} are computed manually by simple vector multiplication operations.

Assume now that we have two velocity vector functions $u = \overline{u}_{AB}$ and $uf = \widehat{\overline{u}}_{AB}$ containing the unfiltered and filtered Adams-Bashforth velocities, respectively, together with a Python list Lij, which contains scalar functions for each tensor component. Also assume that we have a Python function named $filter$, which filters the function given as argument, and then returns the filtered field. This results in L_{ij} being computed as

```
# Loop over each component in Lij
for component in range(tensdim):
    # Extract i, j indexes
    i, j = velocity_pair[component]
    # Add Filter(uiuj) to Lij
    Lij[component].axpy(1.0, filter(u[i]*u[j]))
    # Subtract Filter(ui)Filter(uj) from Lij
    Lij[component].axpy(-1.0, uf[i]*uf[j])
```

The variable *velocity_pair* is pre-defined in the constructor for 2D and 3D cases, so e.g. in 3D the third component of L_{ij} is for indexes $i = 0,\ j = 2$, hence velocity_pair[2] = (0,2). Ordering of the tensor components is done from the top left then right and downwards, such that a general symmetric tensor A_{ij} has component numbering

$$A_{ij,3D} = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 3 & 4 \\ 2 & 4 & 5 \end{pmatrix}, \quad A_{ij,2D} = \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix}. \tag{4.27}$$

The variable *tensdim* equals the number of tensor components, 6 in 3D and 3 in 2D as seen from (4.27). Notice also the operation axpy (applied in section 4.2) that for vectors $\mathbf{a}, \mathbf{b}$ and a constant c mathematically translates to

$$\mathbf{a}.axpy(c, \mathbf{b}) \equiv \mathbf{a} + c \cdot \mathbf{b}. \tag{4.28}$$

L_{ij} is now computed and ready for use.

Computation of M_{ij}

Moving on to the tensor M_{ij}, there are as we can see from Eq. (3.44) some more advanced ingredients required here compared to the computation of L_{ij}. Where we avoided the solution of linear systems for L_{ij}, this cannot be avoided for M_{ij}, since the derivatives of $\overline{u}_{AB}$ are explicitly required in the computation of the tensor. FEniCS again includes functionality for working with tensors and solving huge linear systems for tensors directly; however, it has been tested that solving for each component of $\overline{S}_{ij}$ by smaller linear systems, is a much better and faster solution. For M_{ij} most of the computations are done manually through either vector or array operations, only solving linear systems for the components of $\overline{S}_{ij}$. Followingly, in order for us to obtain $\widehat{\overline{S}}_{ij}$, all the components of $\overline{S}_{ij}$ are filtered by the GTHF operation. It is possible to solve linear systems for the components in $\widehat{\overline{S}}_{ij}$ by applying the filtered velocity $\widehat{\overline{u}}_i$; nevertheless, is has been tested, and it seems to be a better solution to simply filter all the components in the original rate of strain tensor. It is a faster solution as a result of the filtering operation being much quicker than solving a linear system, but it is in general not correct as a result of commutation with derivatives being lost for inhomogeneous filters (which the GTHF filter on an unstructured mesh absolutely is). Testing have, however, shown that it is a better solution to filter $\overline{S}_{ij}$ to obtain $\widehat{\overline{S}}_{ij}$, hence this method has been chosen.

Computing $\overline{S}_{ij}$ and $\widehat{\overline{S}}_{ij}$ is now done as

```
for component in range(tensdim):
    # Solve for components of Sij
    Sij_sol.solve(A, Sij[component], b[component])
    # Filter components of Sij to obtain F(Sij)
    Sijf[component] = filter(Sij[component])
```

FEniCS' *solve* function takes three arguments: the left hand side matrix A, a coefficient vector, which in this case is located in $Sij[component]$, and the right hand side vectors contained in the list b. The right hand sides in b are created by matrix-vector products between pre-assembled "derivative" matrices and the velocity vector, similar to what was done for the σ model in section 4.3.3. The same "derivative"-matrices as applied there may also be used to obtain the components of the rate of strain tensor, since these components are simple linear combinations of the different velocity gradients. Notice that we apply $Sij\_sol.solve(..)$, where Sij_sol here is a pre-defined linear algebra solver, similar to what was done for the σ model, where some tweaks has been done to maximize speed. We could have applied FEniCS' *solve* function directly, but making a pre-defined solver results in a speed up.

When the two rate of strain tensors have been computed their magnitudes are computed through the homemade Python function *mag* as

```
magS = mag(Sij)
magSf = mag(Sijf)
```

Leaving out the details the function *mag* takes a Python list tensor as input, then it by FEniCS vector operations first computes the tensor contraction $\overline{S} = \overline{S}_{ij}\overline{S}_{ij}$, followed by its magnitude as $|\overline{S}| = 2\sqrt{\overline{S}}$ by vectorized NumPy array operations. It should be noted here that one cannot simply filter $|\overline{S}|$ to obtain $|\widehat{\overline{S}}|$, since all the components in these tensors are constructed of nonlinear combinations of the rate of strain tensor components. Therefore, we compute the magnitudes from the both the filtered and the test filtered rate of strain tensors directly. Subsequently, a loop is now made over all the components of the Python list Mij, where the final computations are done as

```
# Loop over all components of Mij
for component in range(tensdim):
    # Add Filter(magS*Sij[component]) to Mij
    Mij[component].axpy(1.0, filter(magS*Sij[component]))
    # Subtract alpha**2*Filter(magS)*Filter(Sij[component])
        from Mij
    Mij[component].axpy(-1.0, alpha**2*magSf*Sijf[component])
    # Multiply Mij by 2*delta**2
    Mij[component] *= 2*delta**2
```

When this loop is done M_{ij} has been computed and is ready for further use.

Averaging the Tensor Contractions

The contractions $L_{ij}M_{ij}$ and $M_{ij}M_{ij}$ are now computed by a Python function applying vector multiplication operations. As for the Lagrange averaging of these fields the same implicit approach as applied by Meneveau [36] has been chosen, such that one just updates the new values for J_{LM}^{n+1} and J_{MM}^{n+1} from the old values J_{LM}^{n} and J_{MM}^{n}. More precisely the scheme is defined as

$$J_{LM}^{n+1} = H\left\{\epsilon L_{ij}^{n+1}M_{ij}^{n+1} + (1-\epsilon)J_{LM}^{n}(x_i - \Delta t\overline{u}_i)\right\}, \tag{4.29}$$

$$J_{MM}^{n+1} = \epsilon M_{ij}^{n+1}M_{ij}^{n+1} + (1-\epsilon)J_{MM}^{n}(x_i - \Delta t\overline{u}_i), \tag{4.30}$$

for $\epsilon = \frac{\Delta t/T}{1+\Delta t/T}$, and the ramp function $H\{x\}$ defined as

$$H\{x\} = \begin{cases} x, & x > 0 \\ 10^{-32}, & x \leq 0 \end{cases}, \tag{4.31}$$

which has been introduced to remove negative values from J_{LM} (J_{MM} is always positive). The Lagrangian PDEs are then updated by vector and vectorized array operations, where subsequently the Smagorinsky constant is computed from Eq. (3.49) applying NumPy array division. As a last step the UFL form for *nut* is defined as

```
nut = C_s * delta**2 * magS
```

The Lagrangian averaging approach has been very problematic (mildly speaking), and initial tests where the Lagrangian PDEs were solved by a Crank-Nicolson scheme in time and the standard linear system procedure, resulted in extremely unstable and erroneous results. As a result of this an implicit method, quite similar to the one used by Meneveau et al. [36], has been chosen. This is also a very efficient solution as one may simply update the PDEs by vector or array operations.

As seen from Eqs. (4.29) and (4.30) the implicit procedure requires upstream values of J_{LM} and J_{MM} at location $x_i - \Delta t\overline{u}_i$. These values were by [36] and others obtained by linear interpolation; nonetheless, it has here not been found a good way of interpolating these terms to the upstream locations. A solution that was tested was to apply the Taylor approximation, e.g. for J_{LM} as

$$J_{LM}^{n}\left(x_i - \Delta t\overline{u}_i\right) \simeq J_{LM}^{n}\left(x_i\right) - \Delta t\overline{u}_i\frac{\partial J_{LM}^{n}}{\partial x_i} + \mathcal{O}\left(\Delta t^2\right), \tag{4.32}$$

and then keep terms up to order Δt. This unfortunately did increase time usage, and introduced new stability problems into the implicit equations. The implementation as it is done now approximates this term to $J^n_{LM}(x_i)$ only, eventually reducing computational accuracy. This can, however, be defended by the fact that there already are many factors regarding this dynamic procedure that are approximations; hence, high accuracy when solving these transport PDEs are not a requirement (e.g. the matrices L_{ij} and M_{ij} are not computed for time location $n+1$, but rather for location $n+1/2$). It may though be so that this type of approximation is a little too rough.

Investigation also shows that the initial conditions for J_{LM} and J_{MM} are critical, as they must be of the same order as the tensor contractions, such that the source contributions actually are accounted for when the PDEs are moved to the next time step. On the other hand, the initial condition for J_{MM} must not be too small as that may result in near zero-division for Eq. (3.49). The original authors [36] propose that the initial conditions should be taken as

$$J_{LM}(x_i, t=0) = C_s^2 M_{ij} M_{ij}, \tag{4.33}$$

$$J_{MM}(x_i, t=0) = M_{ij} M_{ij}, \tag{4.34}$$

which results in J_{LM} and J_{MM} having the correct magnitude at start up. In addition, one could specify C_s at $t=0$ depending on the initial flow type; e.g. for turbulent start up $C_s^2 = 0.15$ could be a good guess, whereas it for a laminar start up could be initialized to some smaller value. This has been tested and is used in the current implementation, as it provides the best results, combined with the fact that the user does not have to specify initial values for J_{LM} and J_{MM}.

A Note on Updating C_s

Running through this algorithm to update C_s does, in despite of a lot of optimizations and shortcuts, require a small amount of extra computational time compared to either Smagorinsky, WALE, or the σ model. To reduce the extra requirement it is possible to update C_s e.g. either each 10th, 20th, or even 50th time step, depending on how rapidly the flow changes per time step Δt.

C_s is then updated at time step n to obtain C_s^n, then the same C_s^n is applied when updating ν_T the next k time steps, until C_s again is updated to C_s^{n+k} at time step $n+k$. This would eventually reduce the overall computational time by some preferable amount; it should, however, be investigated how often C_s needs to be updated in order for the results to be good.

4.3.5 The Scale-Dependent Dynamic Smagorinsky Model

Both from a mathematical and a numerical perspective, the Scale-Dependent Dynamic Smagorinsky model is a simple extension of the Dynamic Smagorinsky model, as a result of all of its required ingredients being very similar to those of the traditional dynamic approach. The only extras needed are the computations of Q_{ij} and N_{ij}, Lagrange averaging of these two tensor contractions, followed by computation of β and C_s. In addition, a second test filtering operation denoted $\widetilde{()}$ is required, defined at the level $\widetilde{\overline{\Delta}} = \alpha\widehat{\overline{\Delta}} = \alpha^2\overline{\Delta}$, with the optimal value $\alpha = 2$ leading to the second test filter $\widetilde{\overline{\Delta}}$ representing a cutoff width of $4\overline{\Delta}$.

Details regarding the implementation of this method will not be presented here, the main reason being that the code and algorithm is much similar to that of the traditional dynamic model. There has neither been found a successful, or good, filtering operation for the second test filter at the α^2-level. As for now this has been approximated by applying the GTHF twice: a solution that works, and which returns some interesting results similar to those obtained by the original authors [39]. However, applying the GTHF twice is wrong in several ways, the most severe problem being the erroneous cutoff scale, which not represents that of $4\overline{\Delta}$. An idea could be to use an extended version of the GTHF where the number of averaging nodes is increased, such that the weighting becomes a function of a higher number of values. The function $\widetilde{G}$ would then be similar to G of the GTHF, where e.g. the function $\widetilde{G}$ for the marked black (middle) node in Fig. 4.5 then would be equal to one for all the marked nodes, and zero else. This is, nevertheless, problematic as this function space no longer comes packed with some important similarities to that of a P_1 function space, compared to what the

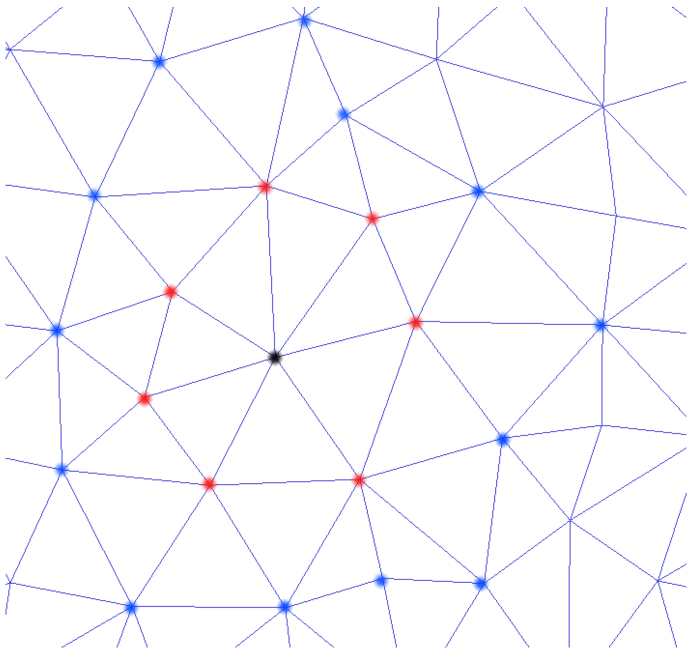


Figure 4.5: Area of averaging for the black (surrounded) node. All nodes must be taken into account in the extended GTHF.

GTHF space did.

A different solution would be to apply the IHF with the value $\alpha^2 = 4$ to obtain the filtered values at the $\widetilde{G}$-level. However, as briefly discussed when presenting the Helmholtz filter, the time requirement of the filtering operation is too high for this method. In addition, some simple testing showed that applying the IHF with $\alpha^2 = 4$ or the GTHF twice, actually yielded solutions that were quite similar.

As a result of these problems, the method has been implemented for testing purposes only. The general behaviour of the model is (in despite of the wrong filtering operation) good, most likely as a result of it being nearly identical to the dynamic model. This model will not be emphasised in any of the succeeding sections, nor be validated or tested with any of the simulations in Ch. 6.

4.3.6 The Kinetic-Energy SGS Model

The Kinetic-Energy SGS model involves one extra PDE for the subgrid-scale kinetic energy k_{SGS}, which must be solved. The equation given in Eq. (3.67) is nonlinear in k_{SGS}^{n+1} if e.g. a Crank-Nicolson implementation is applied; hence, to avoid the requirement of iteration, we rewrite the nonlinear terms as $k_{SGS}^{3/2} = k_{SGS}k_{SGS}^{1/2}$. The Crank-Nicolson scheme in time is then used for terms that are linear in k_{SGS}, combined with explicit evaluation of the terms that are nonlinear in k_{SGS}. This yields for Eq. (3.67)

$$\frac{k_{SGS}^{n+1} - k_{SGS}^{n}}{\Delta t} + \frac{1}{2}\overline{u}_j^{n+1/2}\frac{\partial k_{SGS}^{n+1} + k_{SGS}^{n}}{\partial x_j} = 2C_k \left(k_{SGS}^{n}\right)^{1/2} \overline{\Delta} \overline{S}_{ij}^{n+1/2}\frac{\partial \overline{u}_i^{n+1/2}}{\partial x_j}$$
$$- \frac{1}{2\overline{\Delta}}C_\epsilon \left(k_{SGS}^{n+1} + k_{SGS}^{n}\right)\left(k_{SGS}^{n}\right)^{1/2} + \frac{\partial}{\partial x_j}\left(\frac{C_k \left(k_{SGS}^{n}\right)^{1/2}}{2\sigma}\frac{\partial \overline{k}_{SGS}^{n+1} + k_{SGS}^{n}}{\partial x_j}\right). \quad (4.35)$$

Now both $\overline{u}_j^{n+1/2}$ and $\overline{S}_{ij}^{n+1/2}$ can be computed applying the Adams-Bashforth projection $\overline{u}_{i,AB}$. Additionally, we now see that the equation is linear in k_{SGS}^{n+1}; thus, we can invoke the same type of discretisation as done for the tentative velocity equation in section 2.2.3. Since some k_{SGS} terms are evaluated at time step n the convergence rate of Δt^2 originally obtained with the Crank-Nicolson scheme is lost; however, as a result of us applying the semi-implicit scheme as the general time discretisation the equation becomes much more stable when being solved. We also avoid the need for iteration since the equation becomes linear in k_{SGS}^{n+1}. Notice also how most terms involve nonlinear k_{SGS} terms, or the products between velocities and k_{SGS}. This results in most parts of the equation requiring assembly each time step, increasing the amount of computational time by some amount when this model is used. Avoiding these assemblies is unfortunately not possible.

The full implementation of this equation is quite heavy, thus we will include some portions of it only. We will also skip the FEM discretisation of the PDE, as this procedure is similar to what was done for the tentative velocity equation in section 2.2.3. Moving on to the implementation we start by assembling the term that only contains $(k_{SGS}^n)^{1/2}$, as this will contribute to the right hand side b_i only. This is done as

```
# Extract Ck and Ce from the KineticEnergySGS dict.
Ck, Ce = KineticEnergySGS["Ck"], KineticEnergySGS["Ce"]
# Assemble rhs contribution
bi = assemble(dt*2*Ck*delta*sqrt(ksgs)\
        *inner(Sij,grad(u_ab))*q*dx)
```

We also here have a dictionary named $KineticEnergySGS$ containing the values C_k and C_e, where the user then may specify these values if wanted. Now all the Crank-Nicolson terms are assembled into one large system matrix A_{ij} as

```
# Assemble general system matrix
Aij = assemble(0.5*dt*inner(dot(u_ab, grad(p)), q)*dx
            + inner((0.5*dt*Ce*sqrt(ksgs)/delta)*p, q)*dx
            + inner(0.5*dt*Ck*sqrt(ksgs)*delta*grad(p),
               grad(q))*dx)
```

where p and q here are the trial and test functions for the P_1 function space for k_{SGS}, respectively. We then add all the contributions in A_{ij} to the right hand side as

```
# Add right hand side contribution from Aij
bi.axpy(-1.0, Aij*ksgs.vector())
```

in addition to adding the mass contribution (the only portion of this equation that one may pre-assemble) from the first term on the left hand side to both A_{ij} and b_i as

```
# Add mass contribution to Aij and bi
Aij.axpy(1.0, M)
bi.axpy(1.0, M*ksgs.vector())
```

The boundary conditions are then applied to A_{ij} and b_i, followed by solving the linear system by a pre-defined linear algebra solver. Then the UFL form for ν_T is defined as

```
nut = Ck * delta * sqrt(ksgs)
```

where it followingly is projected and sent to the solver after k_{SGS} has been computed.

This model will not be tested for any of the simulations in Ch. 6; nonetheless, it has been implemented and tested for some simple cases where some good results have been obtained. In terms of computational time the requirements of this model is quite high, mainly as a result of the required matrix assembly procedures. As done with the Dynamic Smagorinsky model and the σ model, one could also here evaluate how often k_{SGS} needs to be updated to save valuable computational time.

4.4 Conclusions

In this chapter discretisaitonal and implementational details for the eddy viscosity formulation have been thoroughly discussed, in addition to implementational details for the Smagorinsky, WALE, σ, Dynamic Smagorinsky, Scale-Dependent Dynamic Smagorinsky, and the Kinetic-Energy SGS models having been presented and justified. In general the LES models that require no explicit computation of any quantities, are of a much simpler nature as they may be expressed through some UFL expressions only, compared to models that either require additional filtering operations, extra PDEs to be solved, or explicit quantities as e.g. the discrete velocity gradients.

Some ideas and solutions for the test filtering operation required in the Dynamic Smagorinsky model was discussed in detail, as this has been one of the more challenging parts of this work. What became clear when research was done for this test filter is that there exists several solutions to this problem, where none is perfect or well defined, or necessarily easy to implement. The simplest and most efficient solution is clearly the GTHF as presented by Jansen [63]. It combines FEM functionality in several ways to produce a fast and good solution, which is both easy to implement and to apply. Currently such a function space G as originally defined by [63] is unavailable in FEniCS; hence, the choice has been made of applying the P_1 basis functions instead. Much likely this choice results in less filtering than if the original formulation for G had been applied, but in general the results should be good, and of the same type as if the originally proposed G had been used.

The implementations of the LES models have been done such that new models, or refined versions of existing models, may be implemented with ease. E.g., most of the functionality used by the Dynamic Smagorinsky model is implemented in a private module-file. This module file contains the filtering operation, a completely general implementation of the Lagrangian averaging approach, general functions for the computation of M_{ij} and L_{ij}, and more. This means that extending models is simple, as general functionality can be imported from a common module. The global dynamic σ model, discussed in section 3.4.2, has actually been implemented as a part of this work. Extending the σ model was done by applying tweaked versions of the already implemented functionality required for the σ and the Dynamic Smagorinsky models, which eventually, after a mere one and a half hours of coding, resulted in the global dynamic σ model. Extending the σ model to the dynamic procedure was particularly easy, then as a result of the differential operator D_σ already having been explicitly computed. Implementational details will not be shown here, but the model has been tested for the FDA case in section 6.2.

The implementations have been refined many times, and in many ways, and a lot of time have been spent on optimizations and so on. That does not mean they are perfect, or maximized in terms of optimization, but they are hopefully good and robust. Nonetheless, continuing to improve upon the work done here should, and will hopefully, be done in the future. It should also be noted that the code snippets presented here are small fractures of the full implementations; e.g., for the Smagorinsky and WALE models, which requires specification of UFL forms only, a mere 30 to 40 lines of code was required. On the other hand, the Dynamic Smagorinsky model is constructed of between 500 and 600 lines of code (where only 20 lines have been included here).

Chapter 5

Verification and Assessment

Naturally, the overall implementation has to be tested and verified such that we are assured that everything has been done correctly. One of the more powerful tools available for verifying implementations of partial differential equations (PDEs) and ordinary differential equations (ODEs), is the so-called Method of Manufactured solutions (MMS). In the case of the Navier-Stokes equations, some analytical fields are initiated for both the pressure and the velocity components, where neither of these in general satisfy the equation set. These expressions are then inserted into the Navier-Stokes equations, where followingly a right hand side residual, or source term, f is computed. The approach of MMS is now to include this residual as a source term in the original equations, such that when the velocity and pressure is inserted into the equation set, the arising residual and the source term will cancel each other, resulting in both the manufactured velocity and pressure being solutions of the Navier-Stokes equations.

For this case, we will employ a combined method, where an analytical solution, which satisfies the Navier-Stokes equation set, is applied. This solution does not satisfy the filtered equation set; hence, the contributions emerging from the eddy viscosity terms will produce residuals. These will then be added as a source term into the equations, resulting in the same analytical solution also being a solution to the filtered equations. The MMS will be done in 2D only, something that can be justified by the fact that the implementations of both the eddy viscosity formulation and the Large Eddy Simulation (LES) models have been done in a completely general way; thus, if they work in 2D, they work in 3D as well.

In addition to convergence tests through MMS, the LES models will be tested for a handful of simple test cases to see how they behave for some selected flow situations. These types of phenomena are present in nearly all types of flows; hence, they are good for illustrational and testing purposes, showing both pros and cons for the different models. Doing this will both present us with some of their advantages and shortcomings, in addition to giving us indications on whether the implementations have been done correctly or not.

5.1 The Method of Manufactured Solutions

Assume that we have an analytical velocity vector as

$$u_{i,A} = [u_A, v_A, w_A], \tag{5.1}$$

which satisfies continuity but is not a solution to the Navier-Stokes equations. We also have the analytical pressure denoted p_A. Applying MMS these analytical expressions are now inserted into the equations, where then residuals (source terms) for each velocity component are computed. These source terms may be inserted back into the equations, such that when they are included and one then inserts the corresponding analytical pressure and velocity, a solution to the equations will be obtained.

MMS is easy to do applying the SymPy package [66], where these residual terms then may be obtained by exact symbolic expressions computed by this framework in Python. These expressions may then be converted into code applicable with FEniCS and $dolfin$, and then be sent directly to *Oasis*. Convergence rates in time will be computed for different cases of ν_T, and the computed eddy viscosities for Smagorinsky and WALE will also be compared to their analytical counterparts to assess the accuracy of the computation.

Applying MMS in a semi-explicit way, we for the analytical solution choose the two-dimensional Taylor-Green vortex, which is an exact solution to the Navier-Stokes equations, with velocities and pressure given as

$$\begin{aligned} u_{TG}(x,y,t) &= -\sin(\pi y)\cos(\pi x)\exp\left(-2\pi^2 \nu t\right), \\ v_{TG}(x,y,t) &= \sin(\pi x)\cos(\pi y)\exp(-2\pi^2 \nu t), \\ p_{TG}(x,y,t) &= -(\cos(2\pi x) + \cos(2\pi y))\exp(-4\pi^2 \nu t)/4, \end{aligned}$$

for a quadratic domain $(x, y) = [0, 2] \times [0, 2]$. Boundary conditions are set to periodic in each direction. To start the solver the analytical correct solution for both the velocities and pressure are initiated at the correct time steps.

For computing convergence rates it is assumed that the error is on the form

$$E_i = C\Delta t_i^r, \tag{5.2}$$

for some constant C and convergence rate r. Since the Incremental Pressure Correction scheme (IPCS) is discretised by the Crank-Nicolson scheme in time we expect second order rates $r = 2$. To obtain r we define the error for two different Δt as

$$E_{k-1} = C\Delta t_{k-1}^r, \tag{5.3}$$

$$E_k = C\Delta t_k^r. \tag{5.4}$$

Each equation is solved for C, then we equate them and solve for r to obtain

$$r = \frac{\ln\left(E_{k-1}/E_k\right)}{\ln\left(\Delta t_{k-1}/\Delta t_k\right)}. \tag{5.5}$$

The relative errors E_k are computed by the L^2 norm, i.e. $||u_i - u_{i,TG}||_{L^2}$.

5.1.1 No Eddy Viscosity

The first test case is for zero eddy viscosity $\nu_T = 0$. Convergence tests for the solver itself has been done by Mortensen [18], however, for the sake of formality, such a test is included here too. The case has been run from $t = 0$ to $t = 10$, and then the final error has been computed.

Table 5.1: Convergence rates for $\nu_T = 0$

Δt	$\|\|u_i - u_{i,TG}\|\|_{L^2}$	r
5.00E-01	6.77E-01	-
1.25E-01	3.73E-02	2.08
1.00E-01	2.39E-02	2.00
6.25E-02	9.34E-03	1.99
3.125E-02	2.34E-03	1.99

As we see from Table 5.1 the convergence rates are as expected around $r = 2$ for the velocity, showing that the general IPCS implementation is good, and the expected Δt^2 error for the Crank-Nicolson scheme is correct.

5.1.2 Constant Eddy Viscosity

First we test for $\nu_T = C$, for some constant C of any size. It should be so that the source term added through f, and the eddy viscosity contribution added through the discretisation, will cancel each other out. Thus, to maximize the effect of the test, we choose some high value as e.g. $\nu_T = 2.2E6$.

Table 5.2: Convergence rates for $\nu_T = 2.2E6$.

Δt	$\|\|u_i - u_{i,TG}\|\|_{L^2}$	r
5.00E-01	7.53E-03	-
1.25E-01	4.70E-04	2.00
1.00E-01	3.01E-04	2.00
6.25E-02	1.17E-04	2.00
3.125E-02	3.08E-05	1.93

It is evident from Table 5.2 that the implementation works good for ν_T equal to a constant, returning the correct convergence rate $r \simeq 2$. This case is, nonetheless, drastically simplified, as the constant eddy viscosity will lead to terms like $\frac{\partial \nu_T}{\partial x_j}$ being equal to zero, eventually removing the Adams-Bashforth contribution to the right hand sides (Eqs. (4.7) to (4.9)).

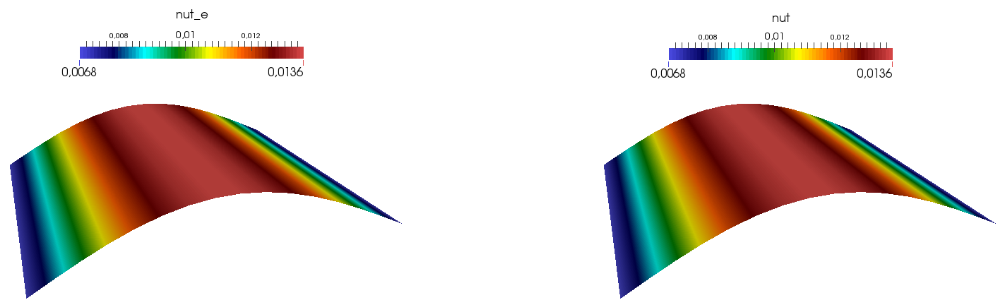
5.1.3 A Fabricated Eddy Viscosity

As a second test case we fabricate a solution for ν_T that is dependent on both time and space, but not on any of the velocity components. We choose a simple analytical expression as

$$\nu_{T,e} = \nu\left(1+\sin\left(\frac{1}{2}\pi x\right)\right)(1+\sin(t))\,. \tag{5.6}$$

Some things that should be noted here: first, a sine function in space has here been chosen, such that the periodic boundary conditions needed for the Taylor-Green solutions also are fulfilled for ν_T. Secondly, as seen in Eq. (5.6), ν_T is multiplied by ν to keep the eddy viscosity term in the vicinity of ν. It has been tested, and severe stability problems emerged if ν_T was much larger than ν. Therefore, the eddy viscosity is here kept in the same range as the kinematic viscosity itself. Most likely, when large values for ν_T are present, one gets larger differences between the source and the eddy viscosity term as a result of discretisational differences that eventually lead to stability problems. In addition, as discussed thoroughly in section 4.1, stability problems are in general a problem as a result of the Adams-Bashforth LES contribution having being discretised fully explicit. Subsequently, if large differences between these factors are present, or if the explicit Adams-Bashforth term is dominating over the Crank-Nicolson viscosity contributions, stability problems may arise.

Moving on to the computations we first check how the analytical eddy viscosity compares to the one computed applying FEniCS. For $\nu = 0.01$ and $t = 10$ we obtain the exact same solution for both expressions, where again the difference computes to be equal to zero. The eddy viscosity is here interpolated to a function space consisting of polynomials of high order (P_4 or P_5), such that spatial errors will become small, followingly eliminating most of the differences between the analytical and the discrete solution. In addition, since the eddy viscosity here is a function of x and t only, there are no errors present, which possibly could have



(a) ν_T as computed by FEniCS. **(b)** $\nu_{T,e}$ as computed by SymPy.

Figure 5.1: Discrete ν_T vs. exact $\nu_{T,e}$ at $t = 10$.

Table 5.3: Convergence rates for $\nu_T = \nu\left(1+\sin\left(\frac{1}{2}\pi x\right)\right)(1+\sin(t))$.

Δt	$\lvert\lvert u_i - u_{i,TG}\rvert\rvert_{L^2}$	r
5.00E-01	8.35E-02	-
1.25E-01	3.33E-03	2.32
1.00E-01	2.10E-03	2.05
6.25E-02	8.11E-04	2.03
3.125E-02	2.01E-04	2.01
1.00E-02	2.07E-05	1.99

emerged as a result of it also being a function of the discretely computed Adams-Bashforth projected velocity field. For the Smagorinsky and WALE models, or for a fabricated ν_T dependent on $\overline{u}_i$, differences would naturally have become larger.

Table 5.3 shows us that the obtained results are as expected, yielding second order convergence also for this case, something which again certifies the general implementation of the eddy viscosity formulation into the solver framework. The convergence rate r is varying slightly more compared to the results obtained for constant ν_T, something that most likely is a result of small errors or differences between the discrete and the exact solution.

5.1.4 The Smagorinsky Model

The next step is now to test the implementation for a selection of eddy viscosity models. As applying SymPy for models that require additional test filtering or extra PDEs to be solved is difficult, MMS will here be done for the implementations of the Smagorinsky and WALE models only.

Starting with the Smagorinsky model ν_T is modelled as in Eq. (3.27). For this case we choose the Smagorinsky constant to have the value $C_s = 0.25$, combined with a uniform mesh constructed of cells with the same size, such that $\overline{\Delta}$ becomes equal to a constant in the whole domain. SymPy is again used for computing the magnitude of the rate of strain tensor from Eq. (3.26).

Compared to the previous fabricated solutions, some additional errors are introduced as a result of ν_T being computed by applying the Adams-Bashforth projected velocity. This approximated velocity is good, and of good order (Δt^2), but it is still a source of errors, which may result in differences between the exact and the discrete eddy viscosities that possibly will perturb in time. In addition, ν_T is by implementational reasons projected to a space of P_1 elements by default, as P_1P_1 and P_2P_1 element pairs most likely are applied for the velocity and pressure, hence, applying a P_1 space for ν_T is accurate enough in those cases.

Table 5.4: Convergence rates for Smagorinsky; ν_T in P_1 space.

Δt	$\|\|u_i - u_{i,TG}\|\|_{L^2}$	r
5.00E-01	5.56E-01	-
1.25E-01	3.00E-02	2.10
1.00E-01	1.92E-02	1.99
6.25E-02	7,66E-03	1.96
3.125E-02	2.16E-03	1.82
1.00E-02	7.46E-04	1.27

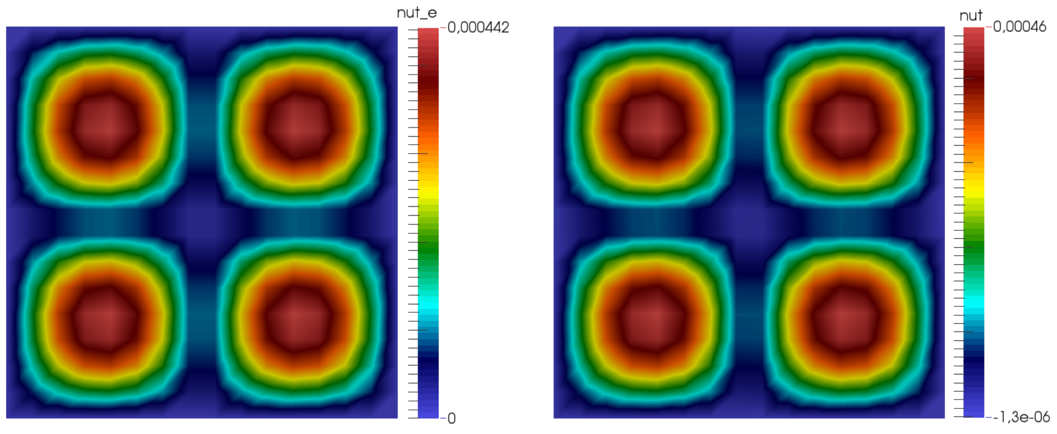
It is clear from the results in Table 5.4 that the convergence rate has a hard time staying around $r = 2$, wandering off as Δt is decreased. This may be a result of $\overline{u}_{j,AB}^{n+1/2}$ being applied in the computation of ν_T, together with ν_T now being projected to a function space of low order linear P_1 elements. Since both the velocity and the pressure are in the P_4 and P_3 function spaces, respectively, for this specific case, the mesh has been kept coarse such that calculations are fast. This again results in bad approximations for ν_T, which further results in large errors that are allowed to grow even more when the time step is decreased.

As a remedy we modify the solver such that ν_T is projected to the same space as the velocity components, eliminating most of the discretisational errors arising as a result of the coarse mesh applied. As we see from Table 5.5 the results are better and more consistent compared to those obtained for the previous solution, especially for the finer time steps where r stays around 2. In addition to this test, it was tried to apply the analytical velocity components when computing ν_T in the P_5 space. This, however, did lead to little or no improvements compared to applying the Adams-Bashforth approximation.

Comparing ν_T at $t = 10$ when a time step of $\Delta t = 0.125$ has been applied, shows us that the analytically computed ν_T, and the one computed by FEniCS

Table 5.5: Convergence rates for Smagorinsky; ν_T in P_5 space.

Δt	$\|\|u_i - u_{i,TG}\|\|_{L^2}$	r
5.00E-01	5.44E-01	-
1.25E-01	2.78E-02	2.14
1.00E-01	1.75E-02	2.08
6.25E-02	6.53E-03	2.09
3.125E-02	1.51E-03	2.11
2.50E-02	9.68E-04	1.98



(a) Analytical $\nu_{T,e}$. **(b)** Discrete ν_T.

Figure 5.2: The Smagorinsky model; analytical $\nu_{T,e}$ vs. discrete ν_T at $t = 10$.

applying the Adams-Bashforth velocity, are nearly identical. There are some differences, e.g. notice from Fig. 5.2 how the one computed by FEniCS actually obtains negative values some places, that in despite of the expression for ν_T being strictly positive. This is a result of the projection method applied to compute ν_T at each time step, where, as a part of the procedure, a linear system has to be solved for all the coefficients of ν_T, something which possibly may lead to small artificial negative values at some locations. A convergence test was done where ν_T was bounded such that minimum values were cut to zero, leading to no difference in the convergence rates compared to those in Table 5.5. The negative values should, however, be removed, as they for other cases may lead to stability problems and errors in the computations. Hence, in the implementation of all the eddy viscosity models a bounding procedure is included such that negative values are removed.

5.1.5 The WALE Model

To see how the WALE model performs the same recipe as used for the Smagorinsky model is applied. C_w is now fixed at $C_w = 0.5$, together with us applying a mesh with constant cell size, i.e. $\overline{\Delta}$ is equal to a constant in the whole domain. The Adams-Bashforth projected velocity is again applied to compute ν_T at time location $n + 1/2$. In addition, based on the results obtained in the last section, we also here apply high order P_5 elements for the eddy viscosity. Investigation of this exact case shows that the analytical expression for WALE actually computes to zero. This is negative in that the solver does not get to reproduce a more advanced expression; on the other hand, it is positive since, if the solver manages to compute the WALE model to zero, the implementation is most likely good,

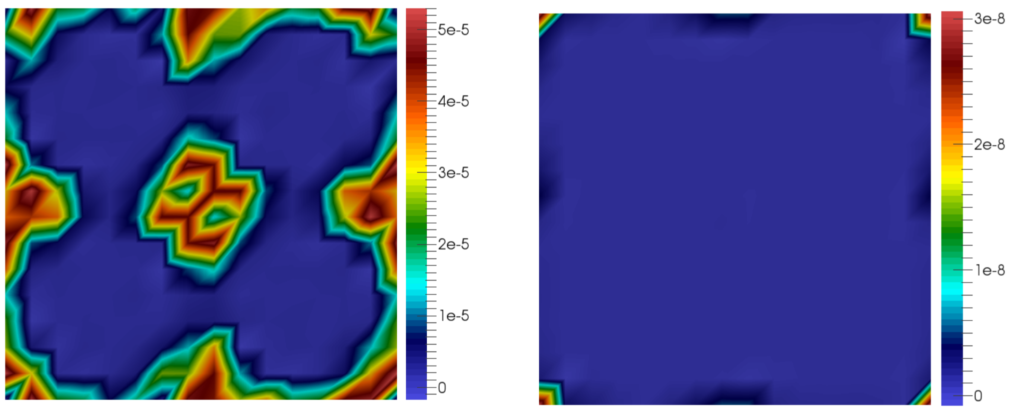
Table 5.6: Convergence rates for WALE; ν_T in P_5 space.

Δt	$\|\|u_i - u_{i,TG}\|\|_{L^2}$	r
5.00E-01	6.77E-01	-
1.25E-01	3.73E-02	2.09
1.00E-01	2.39E-02	2.00
6.25E-02	9.34E-03	1.99
3.125E-02	2.34E-03	1.99
2.50E-02	1.50E-03	1.99

and correct.

The results in Table 5.6 clearly reflects the solver's ability to reproduce the analytical expression for WALE to some extent, where the obtained convergence rates are hovering around $r \simeq 2$ for all Δt. If we look at the eddy viscosity at $t = 10$ obtained with $\Delta t = 0.5$ and $\Delta t = 0.01$, we from Fig. 5.3 clearly see how the exact solution is approximated much better for the lower time step. That is clearly a result of the overall error in the velocity at end time being much lower for the smallest time step, compared to larger errors for the higher time step. For larger time steps the error contribution is, in despite of being nonzero, quite small, thus it does not have any effect on the solving process overall. Nevertheless, for more challenging cases this could have been problematic.

A test where the WALE model would have returned nonzero eddy viscosity could have been done. However, as a result of the good approximations obtained



(a) ν_T at $t = 10$; $\Delta t = 0.5$. **(b)** ν_T at $t = 10$; $\Delta t = 0.01$.

Figure 5.3: WALE; discrete ν_T obtained with $\Delta t = 0.5$ **(a)** and $\Delta t = 0.01$ **(b)**, at $t = 10$.

here together with the results obtained in the next section, the implementation of the WALE model is most likely good and correct.

5.2 Test Cases

5.2.1 Simple Shear Flow

The first test case is laminar flow of simple shear between two plates that are infinitely long in the z and x-directions. The gradients of the velocity vector is for this case given as e.g.

$$\frac{\partial \overline{u}_i}{\partial x_j} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{5.7}$$

Applying the Navier-Stokes equations one arrives at a general solution on the form

$$\overline{u}_i = [y, 0, 0], \tag{5.8}$$

for shear flow between plates located at $y = 0$ and $y = 1$. We have no-slip condition at the bottom, whereas we at $y = 1$ drags the plate with a constant velocity in the vertical direction, equal to $u(1) = 1$.

The velocity field can be plotted in three dimensions as seen in Fig. 5.4. This case is extremely simple, though it does rather clearly show how the LES models return different, both correct and wrong, results for such a trivial case. Since the flow is laminar, no eddy viscosity should be added anywhere in the domain.

If we compute the resulting values for the differential operators applying the Smagorinsky and WALE models, we get the results seen in Fig. 5.5 **(a)** and **(b)**. The first thing to notice from the left plot **(a)**, is how the differential

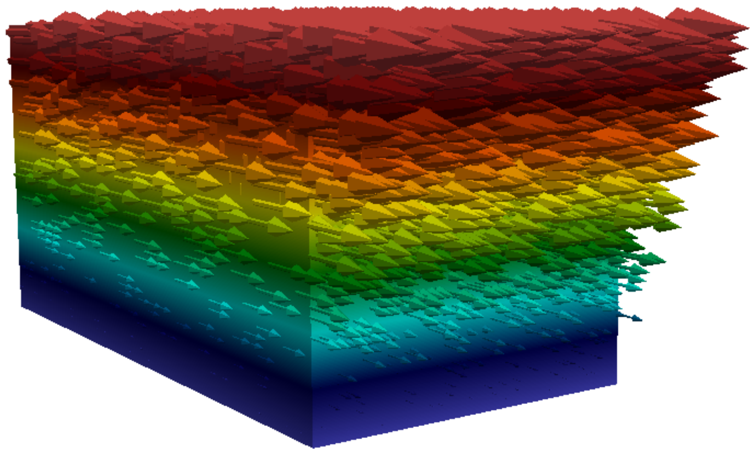
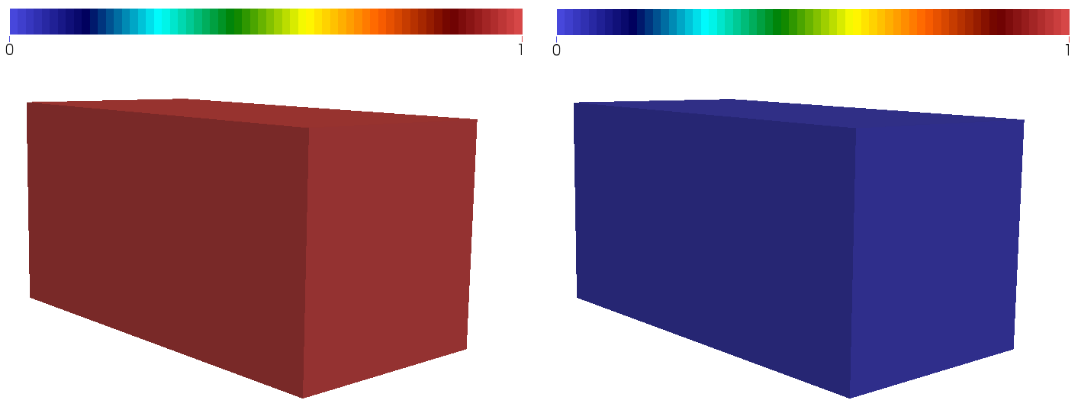


Figure 5.4: Simple shear flow.



(a) Smagorinsky; $|\overline{S}| = 1$.

(b) Wale; $D_W = 0$.

Figure 5.5: Differential operators for Smagorinsky $|\overline{S}|$ **(a)**, and WALE D_W **(b)**.

operator $|\overline{S}|$ for Smagorinsky in this case becomes equal to 1 in the whole domain, subsequently leading to a constant eddy viscosity contribution at all locations. This behaviour is wrong, and actually quite severe, as this type of shear flow is the base flow in boundary layers, and boundary regions of e.g. stable jets or similar. Perturbations, which are present in such regions, may be damped and killed completely by the Smagorinsky models erroneous behaviour. This is a well-known problem with this model, as briefly discussed in section 3.3.1, where damping functions or the dynamic procedure are possible solutions. Another solution is to apply eddy viscosity models that fix this problem through their differential operator; thus, in comparison, the WALE model in Fig. 5.5 **(b)** does, as expected, return $D_W = 0$ in the whole domain. As a result of this, boundary layer instabilities and perturbations are allowed to grow, instead of being damped and killed by an erroneous contribution of eddy viscosity, as the Smagorinsky model does. The σ model does also, as expected, return the same results as for the WALE mode, as D_σ is computed to be equal to zero in the whole domain.

The results obtained with the Dynamic Smagorinsky model for both local and Lagrangian averaging were for this case equal to that obtained with the WALE model. No eddy viscosity was added as a result of C_s being computed to $C_s \approx 0$ in the whole domain, eliminating the problem of $|\overline{S}|$ being nonzero. For this simple test case, the Lagrangian averaging approach is somewhat overpowered, as it was constructed specifically to cope with complex geometries combined with complex, turbulent flows. Hence, as a result of this, the locally averaged (by the GTHF) dynamic model was tested, where it returned results that were just as good as its more advanced counterpart. It was also tested to apply two different initial conditions for the Lagrangian PDEs, namely $C_s = 0.16$ and $C_s = 10^{-5}$. Setting C_s to 0.16 resulted in the Lagrangian PDEs being pushed towards zero as they were updated. The second initialization value resulted in C_s constantly floating around 10^{-10}.

5.2.2 Solid Body Rotation

The second test case is that of laminar solid body rotation, defined in three dimensions but with a z-directed velocity component $w = 0$. The gradients of the velocity vector are now given as

$$\frac{\partial \overline{u}_i}{\partial x_j} = \begin{bmatrix} 0 & -\alpha & 0 \\ \alpha & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{5.9}$$

for rotation in the anti clockwise direction. There are other possible expressions for the gradient tensor that would have led to the same results, however, this specific case has been chosen here. For simplicity the values of the gradients will here be set to $\alpha = 1$. Applying the Navier-Stokes equations together with integration, results in the analytical solutions for the velocity vector as

$$\overline{u}_i = [-y + C, x + D, 0] \tag{5.10}$$

We here choose the centre of the rotation to be located at origo, leading to $C = D = 0$. For the velocity field see Fig. 5.6.

One interesting thing to notice here is that the rate of strain tensor becomes equal to zero as

$$\overline{S}_{ij} = \frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right) = 0. \tag{5.11}$$

This result leads to the Smagorinsky model behaving correctly in this case, since its differential operator $|\overline{S}|$ will become equal zero. Certainly it is clear from Fig. 5.6 that the flow is laminar and the eddy viscosity should subsequently be zero in the whole field, therefore, the correct behaviour is obtained through the appliance of the differential operator $|\overline{S}|$. See Fig. 5.7 **(a)** for the results obtained with the Smagorinsky model, the eddy viscosity is as expected, and correctly, computed to be equal to zero.

Where the rate of strain tensor became zero, the antisymmetric rate of rotation tensor does for this case become nonzero as

$$\overline{\Omega}_{ij} = \frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j} - \frac{\partial \overline{u}_j}{\partial x_i}\right) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{5.12}$$

It can actually be shown that the differential operator D_W of WALE, can be directly decomposed into terms containing the rate of strain tensor, terms containing the rate-of-rotation tensor, and terms containing a combination of both.

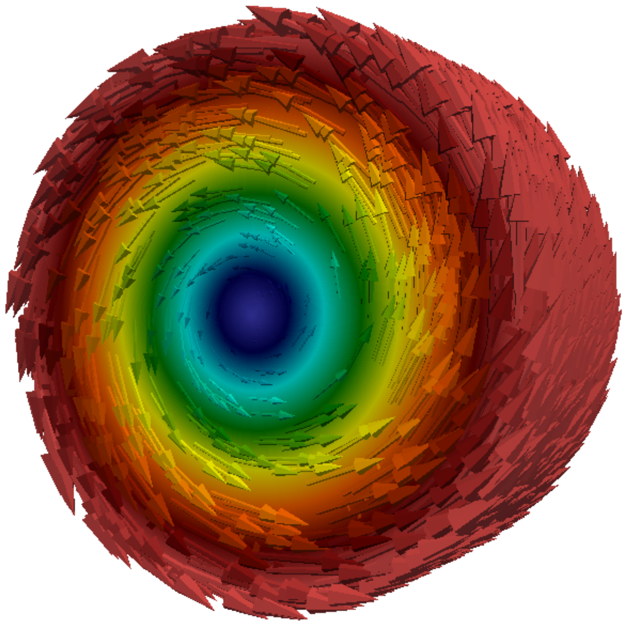
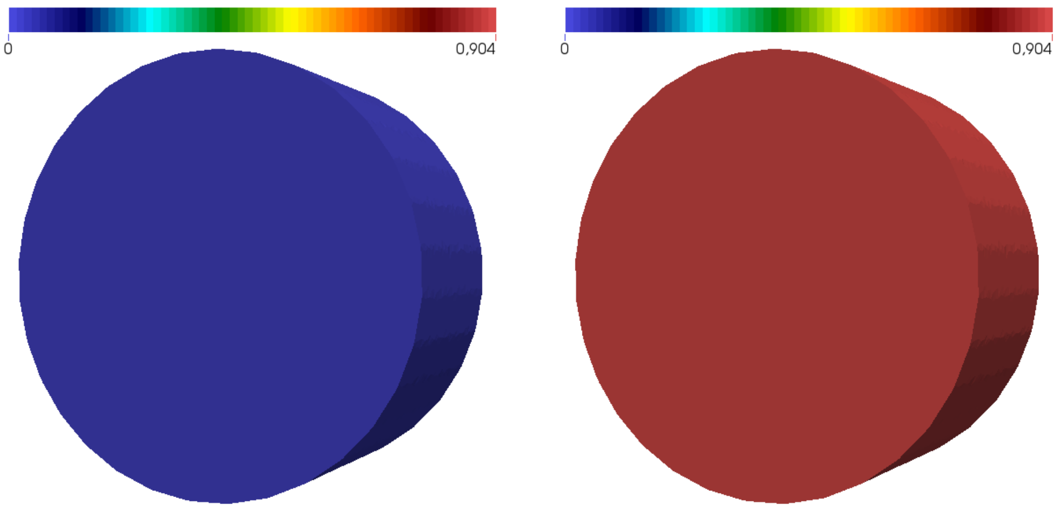
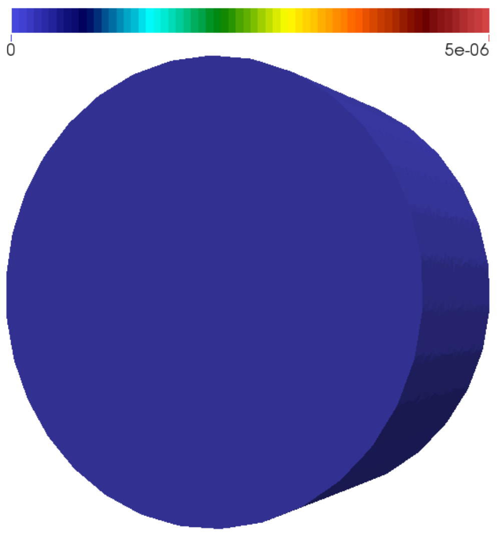


Figure 5.6: Velocity field for the case of solid body rotation.

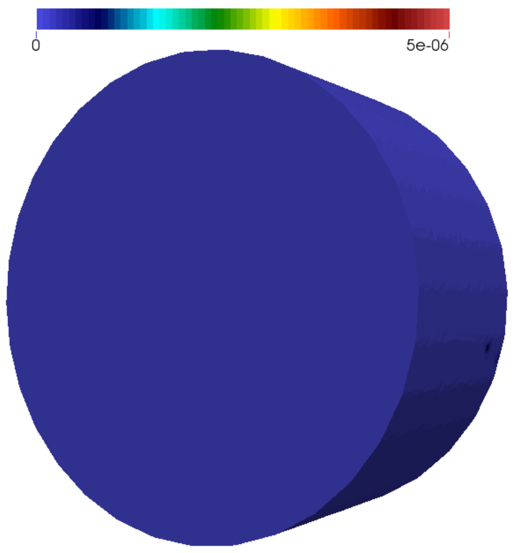


(a) Smagorinsky; $|\overline{S}| = 0$. **(b)** WALE; $D_W \approx 0.9$.

Figure 5.7: Solid body rotation, differential operator values for the Smagorinsky **(a)** and the WALE model **(b)**.



(a) Local averaging.



(b) Lagrangian averaging.

Figure 5.8: Solid body rotation; results obtained for C_s with the Dynamic Smagorinsky model, local averaging **(a)** and Lagrangian averaging **(b)**.

Hence, since $\overline{\Omega}_{ij}$ here becomes nonzero, the differential operator D_W also becomes nonzero where it for this case take the value of $D_W \approx 0.9$ at all grid locations, as seen in Fig. 5.7 **(b)**. Thus, for this specific flow type, the WALE model actually fails to predict zero eddy viscosity, whereas the Smagorinsky model quite surprisingly tackles this correctly. No plot for the σ model is included here, but the model does, as expected, correctly compute D_σ to be equal to 0 also for this case.

The Dynamic Smagorinsky model does also here compute the Smagorinsky constant to be equal to zero overall, then for both Lagrangian and local averaging. The correct behaviour is here secured in two ways: $|\overline{S}|$ is equal to zero, and the Smagorinsky constant C_s is computed to be equal to be near zero everywhere. For the results for C_s for both averaging techniques see Fig. 5.8.

5.3 Conclusions

In this chapter convergence tests have been done for the general residual stress tensor implementation by applying several fabricated ν_T fields. Convergence tests was also done for the Smagorinsky and WALE models. Good values for the convergence rate r were obtained for all cases, pointing towards good implementations for both the general eddy viscosity terms, and for the Smagorinsky and the

WALE models. Some stability problems when the fabricated ν_T in Eq. (5.6) was applied were encountered if $\nu_T >> \nu$. These problems are most likely connected to the stability problems discussed in section 4.1, hence they most likely do not point towards general errors in the implementations. The discrete eddy viscosities for Smagorinsky and WALE were compared to their analytical counterparts, in general showing good coherency.

When it comes to the test cases, the results obtained are good and as expected. The Smagorinsky model fails for simple shear, whereas the WALE model, the σ model, and the Dynamic Smagorinsky model all tackle this case perfectly. For the case of solid body rotation the WALE model actually fails, returning nonzero eddy viscosity for a simple case where the eddy viscosity should have been equal to zero everywhere. Both the Smagorinsky and the Dynamic Smagorinsky model had no problems coping with this specific case, both of them through $|\overline{S}| = 0$, where the dynamic model in addition secured this through C_s being computed to be equal to 0 in the whole domain. Again the σ model proved its strengths, and correctly returned zero eddy viscosity for this case as well.

As mentioned the problems with the Smagorinsky model seen for the shear flow case, is especially severe for flows involving boundary layers, where the problem becomes more and more drastic as a function of the thickness of the boundary layer δ. If δ is small, the gradients near the wall are high, thus leading to high values for the differential operator $|\overline{S}|$. In addition to boundary layers, simple shear regions of some type are present in almost all types of flows. E.g. for a stable, laminar jet the phenomenon may be found all along the boundary of the jet. As this is a region where instabilities may start to grow, eventually leading to a breakdown of the jet, the erroneous eddy viscosity contribution of the Smagorinsky model may lead to damping of all perturbations, and subsequently no transition from laminar to turbulent flow. Some of these problems will be further addressed and analysed for the FDA case in section 6.2.

It should also be noted, again, that in despite of the MMS tests having been done for the two-dimensional case only, all the implementations have been done in a completely general way such that if good results are obtained in 2D, results just as good would most likely have been obtained in 3D. The two presented test cases were performed for the three-dimensional case; thus, when e.g. the differential operator D_W for WALE was computed, the input for the solid body rotation case was a velocity vector where the w component was equal to zero. The implementations, both for WALE but also for the other models, clearly managed to compute the correct eddy viscosities for both test cases. Therefore, when the MMS and test results are combined, all arrows are pointing towards the implementations being correct and good.

Chapter 6

Results and Discussion

Testing through the Method of Manufactured Solutions (MMS) and verification of the implementation by some simple test cases are, in despite of their simplicity, important ingredients. Through the MMS the general implementation has been shown to be good, both in terms of stability and of the obtained convergence rates of Δt^2. The simple test cases are applied merely to check that some expected results are obtained under very special circumstances, additionally showing that the results returned by the differential operators for some of the models are either correct or wrong, and equal to some expected value.

In despite of the previous tests being positive, the implementation still needs to be verified and validated by more conventional and advanced methods, to show that the work indeed is good. For this purpose, two cases have been chosen: the first one being the traditional setup of fully developed turbulent channel flow in a channel, periodic in the x and z directions. Good Direct Numerical Simulation (DNS) data obtained by higher order methods are available for a selection of Reynolds numbers, the case is easy to set up, and is in general one of the classic benchmarking tools in the Computational Fluid Dynamic (CFD) community. Secondly the U.S. Food and Drug Administration (USFDA/FDA)'s computational round robin #1 has been chosen. The case consists of blood flow in a pipe, followed by a nozzle, a narrow pipe, and an expansion into a larger pipe. The expansion will result in a jet that transitions into turbulent flow, and completely dissipates after some time. Controlling the breakdown of the jet (and hence also the transition to turbulence) is one of the challenges with this simulation. Where under-resolved DNS in general predicts the jet to break too early, the Large Eddy Simulations (LES) will hopefully present us with better results.

6.1 Turbulent Channel Flow

The case of fully developed turbulent channel flow is a traditional benchmarking method, often used to test the performance of CFD codes. DNS data is available online at [67], where the DNS simulations have been done by Moser, Kim and Mansour (MKM) [22], and by Jimenez et al. [23]. Their DNS data was obtained by higher order spectral methods such that the correct rate of dissipation of turbulent kinetic energy was captured. E.g. MKM [22] used a Chebyshev-tau formulation in the wall-normal y direction, and Fourier representations in the x and z directions.

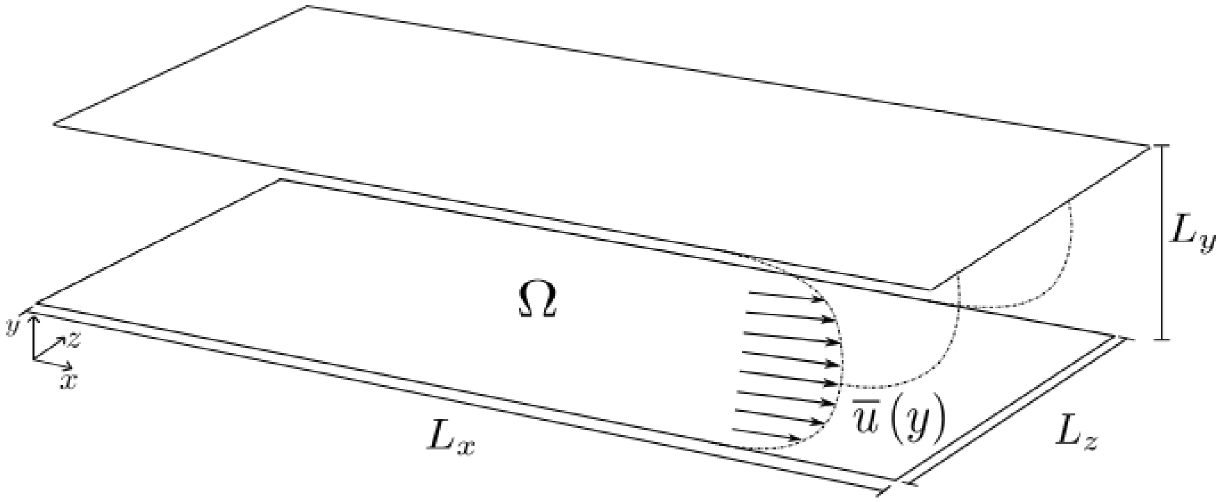


Figure 6.1: Domain for the channel flow case.

The channel is a computational box of dimensions L_x, L_y and L_z, where $L_y = 2\delta$, and L_x and L_z are problem specific in terms of the Reynolds number. For the simulations to be done here we have chosen the channel half width $\delta = 1$, in addition to the top wall and the bottom wall of the channel being located at $y = \pm 1$. The two vertices defining the domain are then located at $[0, -L_y/2, -L_z/2]$ and $[L_x, L_y/2, L_z/2]$. No-slip boundary conditions are applied at the walls, in addition to periodic boundary conditions being applied in the x and z directions. As discretisation in space we apply a coarse mesh of 64 hexahedra in each direction, where each hexahedra again is divided into 6 tetrahedra, resulting in a total of $6 \cdot 64^3$ cells. Element types will be P_1 elements for both velocity and pressure, but for $Re_\tau = [180, 395]$ we will also present some results applying P_2 Lagrange elements for the velocity and P_1 Lagrange elements for the pressure, then applying a coarser mesh of $6 \cdot 32^3$ cells. The velocity space will then have the same number of degrees of freedom as when $6 \cdot 64^3$ cells is applied, whereas both the pressure and ν_T will be in a lower ordered P_1 space. Some problem specific parameters are the shear velocity defined as

$$u_\tau = \sqrt{\nu \partial u / \partial y_{wall}} \tag{6.1}$$

Table 6.1: Parameters for the selected cases of fully developed turbulent channel flow.

$Re_{\tau,\text{Nom.}}$	$Re_{\tau,\text{Act.}}$	L_x	L_z	Q_{Re_τ}
180	178.13	$4\pi\delta$	$\frac{4\pi}{3}\delta$	0.467919
395	392.24	$2\pi\delta$	$\pi\delta$	0.864785
590	587.19	$2\pi\delta$	$\pi\delta$	1.376444
950	944.00	$8\pi\delta$	$3\pi\delta$	6.981046

and the turbulent Reynolds number defined as

$$Re_\tau = \frac{\delta u_\tau}{\nu}. \tag{6.2}$$

We have here chosen a kinematic viscosity of $\nu = 2 \cdot 10^{-5}$. Further, the normalization parameters applied when plotting the mean data are defined as

$$U^+ = \frac{u}{u_\tau} \tag{6.3}$$

$$y^+ = \frac{y u_\tau}{\nu}. \tag{6.4}$$

Since the domain is periodic in both the x and z directions, some forcing that consequently drives the flow through the domain needs to be activated. Analytical work shows that the pressure gradient $\frac{\partial p}{\partial x}$ for this specific case actually is computed to

$$-\frac{\partial p}{\partial x} = u_\tau^2. \tag{6.5}$$

This forcing can be activated by adding a source term to the right hand side of the Navier-Stokes equations as

$$\overline{f}_{b,S} = \left[u_\tau^2, 0, 0\right]. \tag{6.6}$$

Now two different approaches exists, the second one being the most popular approach (used by [22], [23], and many more), but both of them applicable.

1. **The Constant Gradient (CG) method.** Both Re_τ and ν are specified, hence compute u_τ from Eq. (6.2). Apply u_τ as a constant forcing term into the Navier-Stokes equations. Compute until the flux stabilizes, then start to sample mean data. This method is slower compared to the next one, as one is required to wait until the flux has stabilized before sampling of data can be initiated. It is, however, more "natural" as the analytically correct forcing is applied. The CG method: correct shear velocity forcing, wrong flux.

2. **The Flux Adjustment (FA) method.** Instead of applying the constant forcing the correct mass flux is specified, where thus the forcing is adjusted at each time step such that the correct flux is obtained. This requires two extra ingredients: (a) the correct flux for a given Re_τ, and (b) a method of updating the forcing term. When these ingredients have been specified, the forcing term can be updated each time step. Compared to the CG method this one is more of a "cheating" approach, as near perfect results will be obtained since the flux is directly adjusted to the correct one. The FA method: wrong shear velocity forcing, correct flux.

What procedure one applies is up for choosing; however, because of the first method being the most correct one, it has been selected here. In addition, activating a LES model with the CG method is more of a natural choice, as the mean velocities then will, as a result of unresolved quantities, be overpredicted, where followingly the LES model's task is to add dissipation, and hopefully alter the mean velocities towards the DNS data. The FA method has also been tested to see how the results stack up compared to the CG method, hence, some details regarding flux computation and the forcing update should be given.

The volume fluxes for each Re_τ are obtained by numerically integrating the dimensionalised half-velocity mean profile as

$$Q_{Re_\tau} = \int_{-L_z/2}^{L_z/2} \int_0^{2\delta} u_{Re_\tau} dydz. \tag{6.7}$$

Since the profile is symmetric as $u_{Re_\tau}(y) = u_{Re_\tau}(2\delta - y)$ and independent of z, the integral simplifies to

$$Q_{Re_\tau} = 2L_z \int_0^{\delta} u_{Re_\tau} dy. \tag{6.8}$$

The integrals are calculated applying the data from [22] and [23] (which come as non-dimensionalised values, hence the u_τ and ν applied here is used to dimensionalise them), together with NumPy's trapezoidal method to perform the numerical calculation. The values for Q_{Re_τ} can be found in the last column in Table 6.1. As for the forcing update a simple procedure is used as

$$\overline{f}_{b,S} = \left[\frac{Q_{Re_\tau} - |Q|}{A\Delta t}, 0, 0 \right], \tag{6.9}$$

where Q is the volume flux computed at the outlet/inlet of the domain, and $A = L_y L_z$ is the area of the corresponding surface. Clearly, if Q is larger than Q_{Re_τ}, the source becomes negative, leading to a reduction in volume flux. Vice-versa, if Q is smaller than Q_{Re_τ}, the source becomes positive and one will see an increase in volume flux. Testing resulted in convergence to the correct flux just after a few time steps, compared to the other solution that needed from 10k to 50k time steps to stabilize at a higher flux.

As for the initial condition, a randomly perturbed base flow is applied such that the flow is in a turbulent state immediately at start-up. The base flow is a slightly modified version of the traditional log-law equation, where the x-directed velocity is given as

$$\overline{u} = 1.25\left(\frac{u_\tau}{\kappa}\ln\left(\frac{u_\tau}{\nu}\right) + 5u_\tau\right). \tag{6.10}$$

Perturbation is done for this base flow, and for the y- and z-directed velocity components, by a random stream function created by applying Pythons random-library. The values of this stream function was found to be important: the perturbations needed to be high enough at start-up such that the viscosity did not immediately damp them. On the other hand, too high values would eventually lead to problems, as the magnitude of the perturbations possibly would be too high. Achieving good magnitudes for the perturbations was done by multiplying the randomized values by a constant equal to 0.0025, resulting in nice turbulent start-up conditions.

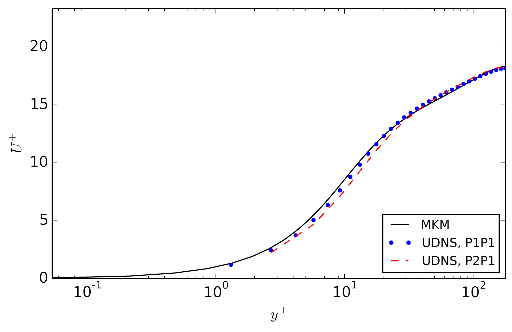
For all cases mean velocity data for resolved DNS, under-resolved DNS, and results obtained with the LES models activated, will be compared. Both forcing methods will be shown, but the focus will be on applying the CG method, as this is the most natural and correct way of handling this case. As for the mean data, the velocities are first averaged over a long time period, and then the time-averaged data sets are averaged over the whole channel, eventually resulting in the plotted mean profiles. Some mean Reynolds stress data will also be shown for $Re_\tau = 395$ and $Re_\tau = 590$. The notation is a little confusing; thus, to clarify, e.g. for the Reynolds stress $\overline{u'u'}$, it is computed as

$$\overline{u'u'} = \overline{\overline{u}\,\overline{u}} - \overline{\overline{u}}\,\overline{\overline{u}}, \tag{6.11}$$

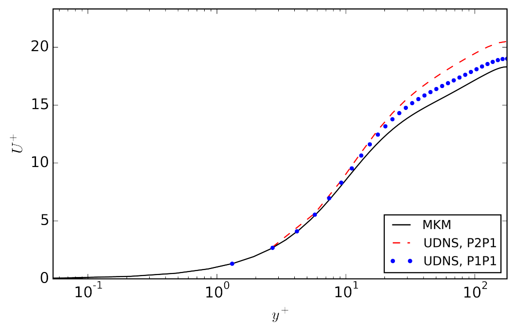
where the upper overline now denotes the general time and domain average and not the LES filter, whereas $\overline{u}$ here denotes the velocity obtained from the LES simulations.

6.1.1 $Re_\tau = 180$

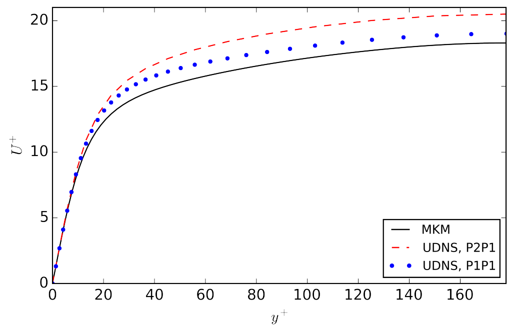
First we take a look at plots of the obtained mean under-resolved DNS data for both P_2P_1 and P_1P_1 element pairs, and compare with the data provided by [22]. The first thing to notice is how the FA method gives very good mean velocity profiles, compared to that of resolved DNS (Fig. 6.2 **(a)**; MKM is DNS data, UDNS is the under-resolved DNS data), where the results are slightly better with the P_1P_1 combination of elements. Moving on to the CG method we in Fig. 6.2 **(b)** and 6.2 **(c)** see how the flux balances out at a higher value compared to the DNS profile. This is expected behaviour for all Re_τ since the coarse mesh in general will result in a huge range of turbulent effects being left out of



(a) P_1P_1 vs P_2P_1, FA.



(b) P_1P_1 vs. P_2P_1, CG.



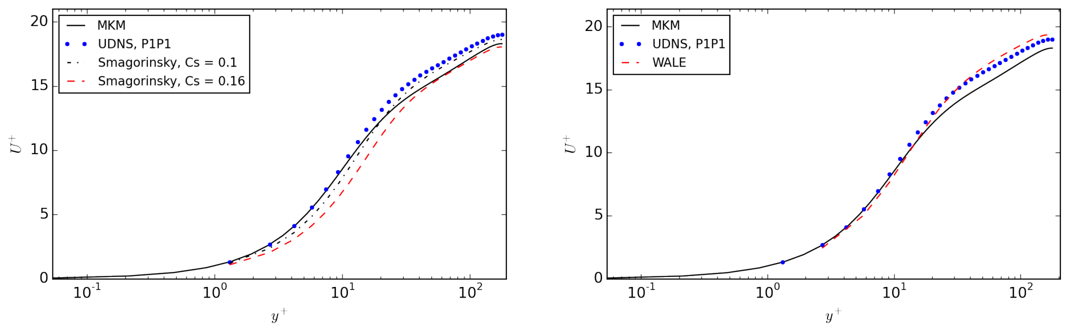
(c) P_1P_1 vs. P_2P_1, CG, non-log.

Figure 6.2: Channel flow, $Re_\tau = 180$; mean velocities for under-resolved DNS.

the simulations. Again we see that the P_2P_1 mean data is worse compared to the one obtained with P_1P_1, resulting in slightly higher mean velocities. Notice how the solution in the inner part of the boundary layer is well represented, as the matching between the DNS data and the mean data is good for $y^+ \in [0, 5]$ approximately.

If we move on to the LES simulations we in Fig. 6.3 **(a)** see how the Smagorinsky model behaves for the two values of $C_s = 0.16$ and $C_s = 0.1$. It is clear that the eddy viscosity contribution has a positive effect, as the mean velocity is adjusted towards the DNS data as expected. Especially in the outer region the results obtained for $C_s = 0.16$ is in good agreement with the DNS data. It most likely is just a matter of adjusting C_s such that the best possible results are obtained. In the inner wall-region, where the under-resolved DNS solution actually represents resolved DNS data quite well, the Smagorinsky model is clearly over-dissipative, something which eventually leads to the mean data being altered a little too much there. For $C_s = 0.1$ we see that the mean data places itself in between under-resolved DNS and resolved DNS data, where the mean velocities in the boundary layer are less affected compared to what was seen for $C_s = 0.16$.

The results obtained for the WALE model (Fig. 6.3 **(b)**) with $C_w = 0.325$ are



(a) Smagorinsky, $C_s = [0.1, 0.16]$, CG. **(b)** WALE, CG.

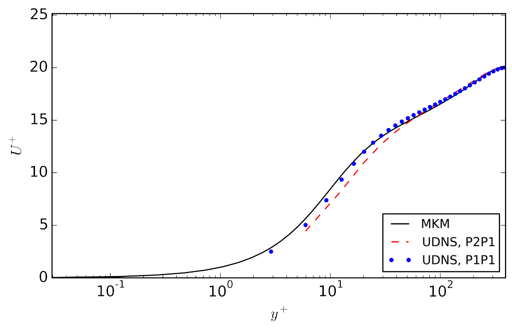
Figure 6.3: Channel flow, $Re_\tau = 180$; mean velocities for Smagorinsky and WALE, CG method.

clearly wrong, and unexpected, as the overall mean velocities becomes slightly higher compared to the ones obtained with under-resolved DNS. When starting the computations after activating the WALE model, the flux increased and stabilized at a higher value, quite opposite to what was seen for the Smagorinsky model. As we see there is a slightly positive effect closer to the wall, however, at approximately $y^+ = 40$ the mean velocities becomes higher than those seen for under-resolved DNS. This exact same result was obtained for the σ model and the Dynamic Smagorinsky model, as both of them returned higher mean velocities compared to those obtained when no model was activated. This behaviour does indeed point towards implementational errors, either in the general eddy viscosity term, or in any of the LES models. Nonetheless, since the Smagorinsky model returns good results, combined with this strange behaviour of the WALE, σ, and the Dynamic Lagrangian model, some arrows are pointing towards this being more of a universal problem for this specific case. The discussion regarding this phenomenon is quite extensive, some results and discussion can be found in section 6.1.5.

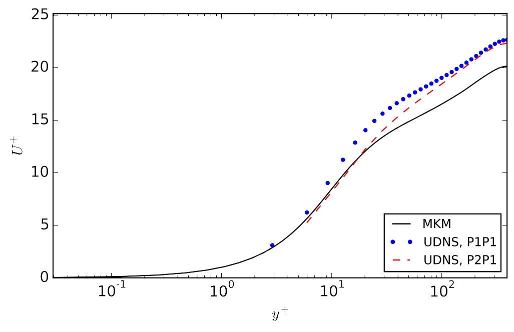
6.1.2 $Re_\tau = 395$

The mean velocities for this case, seen in Fig. 6.4, show the same tendencies as the ones obtained for the $Re_\tau = 180$ case. Differences here are a larger distance between the under-resolved DNS and the resolved DNS data, in addition to the P_2P_1 simulation actually returning a slightly better solution closer to the wall for the CG method. Again we see how the FA method produces results that are in very good agreement with those of resolved DNS, where, as with the previous case, the CG method also here produces too high mean velocities.

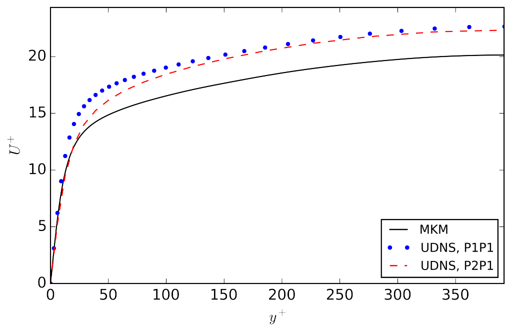
Simulations applying LES models have here been done for the Smagorinsky model with $C_s = 0.1$, and the σ model with $C_\sigma = 1.5$. The Smagorinsky model also here has a generally positive effect on the simulation, returning a mean velocity profile closer, though slightly higher, than the DNS data. The value of $C_s = 0.1$ is too conservative here, where small adjustment of the Smagorinsky constant most likely would have returned better results. The σ model returns results that are similar to those seen for the WALE model for the previous case, where mean velocities in general are higher compared to under-resolved DNS data,



(a) P_1P_1 vs P_2P_1, FA.



(b) P_1P_1 vs. P_2P_1, CG.



(c) P_1P_1 vs. P_2P_1, CG, non-log.

Figure 6.4: Channel flow, $Re_\tau = 395$; mean velocities for under-resolved DNS.

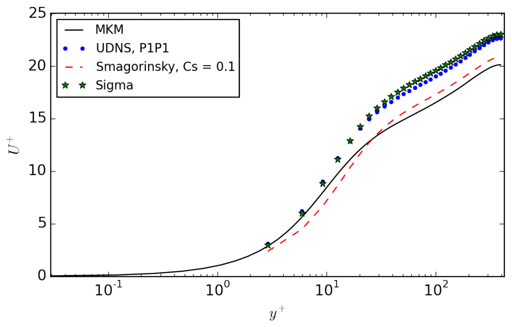
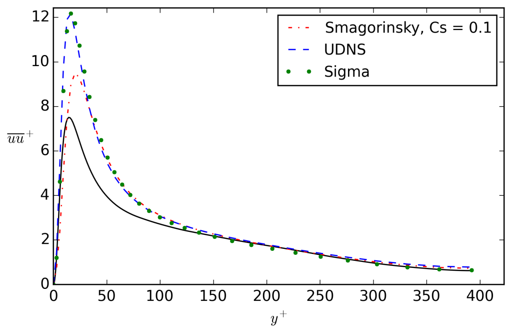
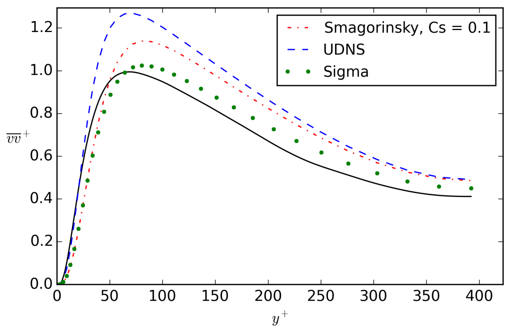


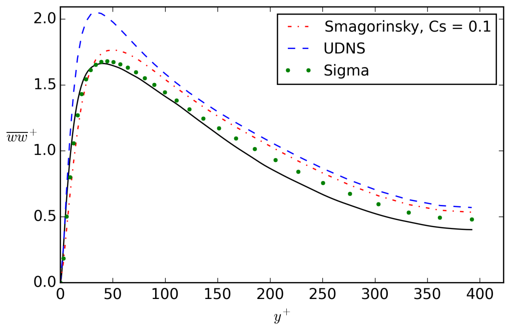
Figure 6.5: Channel flow, $Re_\tau = 395$; mean velocities for Smagorinsky and σ, CG.



(a) Mean Reynolds stress $\overline{u'u'}$ for Smagorinsky, σ and UDNS.



(b) Mean Reynolds stress $\overline{v'v'}$ for Smagorinsky, σ and UDNS.



(c) Mean Reynolds stress $\overline{w'w'}$ for Smagorinsky, σ and UDNS.

Figure 6.6: Channel flow, $Re_\tau = 395$; Reynolds stresses for UDNS, Smagorinsky and σ.

especially outside of the boundary layer region. A positive, though extremely weak effect is seen closer to the wall, where the σ model returns slightly lowered mean velocity data; however, we also here see how the model returns higher mean velocities as we come closer to the middle of the channel.

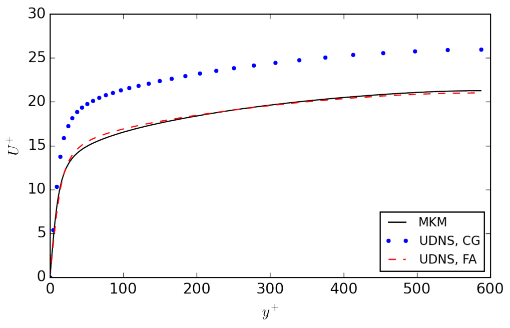
Plots for the Reynolds stresses $\overline{u'u'}$, $\overline{v'v'}$ and $\overline{w'w'}$ can be seen in Fig. 6.6. All three simulations overpredict the Reynolds stresses $\overline{u'u'}$ as seen in Fig. 6.6 **(a)**, where the results are highest for the σ model and the UDNS simulations, but slightly lower for the Smagorinsky model. For the two other components in Figs. 6.6 **(b)** and **(c)** both LES models results in better mean data compared to that of UDNS, where it is the σ model that quite surprisingly produces the best mean profiles. The σ model clearly dampens the two latter Reynolds stress components, whereas the former stays unchanged. Since the mean velocity profile for the σ model is nearly identical to the one of UDNS, the results obtained for $\overline{u'u'}$ is as expected; on the other hand, it looks like the two other components are damped independently of the u' fluctuations, where clearly the mean velocity results are unaffected by the improved results for $\overline{v'v'}$ and $\overline{w'w'}$.

6.1.3 $Re_\tau = 590$

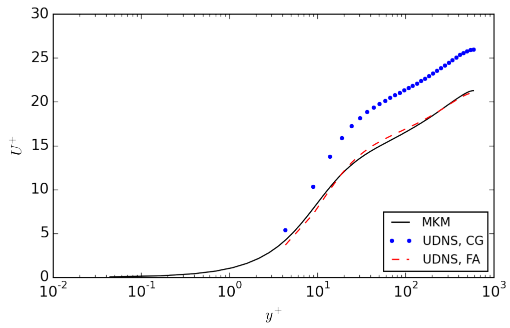
Moving on the third case of $Re_\tau = 590$ we again see that the results shows the same behaviour as with the two previous Re_τ. If we compare the CG and the FA methods, it is clear that the latter returns very nice results, whereas the former returns mean velocities that are drastically higher than DNS data. The P_2P_1 element pair has not been tested here as a result of stability problems.

As for the LES simulations in Fig. 6.8 we again see how the Smagorinsky model returns good results, where it is clear that $C_s = 0.1$ also here is a bit small. The Dynamic Smagorinsky model with Lagrangian averaging is similar to WALE and σ for the two previous Re_τ, almost no differences can be seen compared to the UDNS-profile, but where WALE and σ returned higher mean velocities, this is not the case for the dynamic model. In the middle of the channel, the returned mean data is almost identical to that of under-resolved DNS, whereas there clearly are some effects closer to the wall where the returned mean data is slightly improved. The results are, however, disappointing.

In Fig. 6.9 the Reynolds stresses $\overline{u'u'}$, $\overline{v'v'}$ and $\overline{w'w'}$ are plotted for DNS,



(a) CG vs. FA, P_1P_1.



(b) CG vs. FA, P_1P_1, non-log.

Figure 6.7: Channel flow, $Re_\tau = 590$; mean velocities for under-resolved DNS.

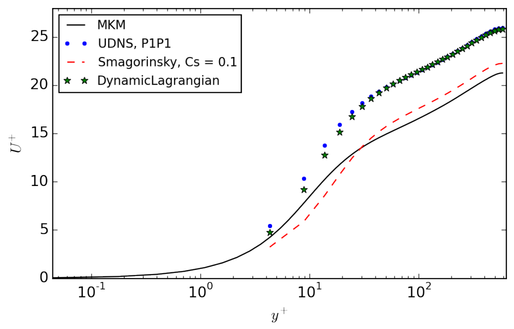
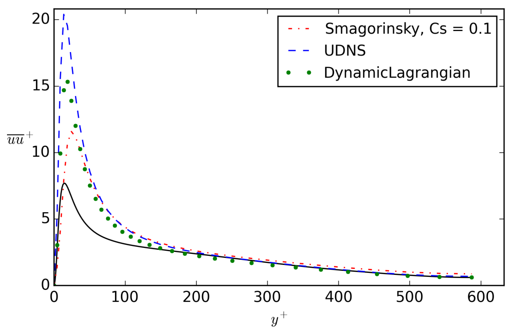
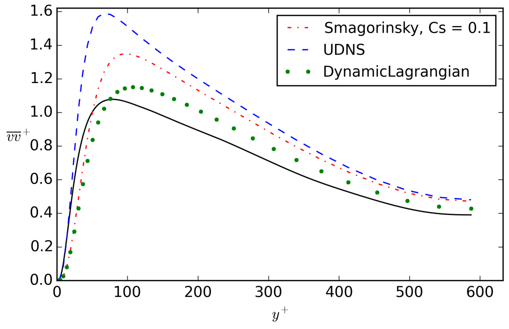


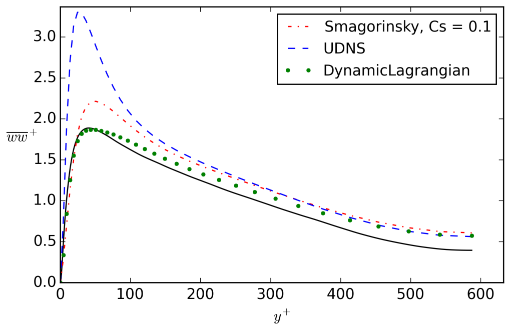
Figure 6.8: Channel flow, $Re_\tau = 590$; mean velocities for Smagorinsky and Dynamic Lagrangian.



(a) Mean Reynolds stress $\overline{u'u'}$ for UDNS, Smagorinsky and Dynamic Lagrangian.



(b) Mean Reynolds stress $\overline{v'v'}$ for UDNS, Smagorinsky and Dynamic Lagrangian.



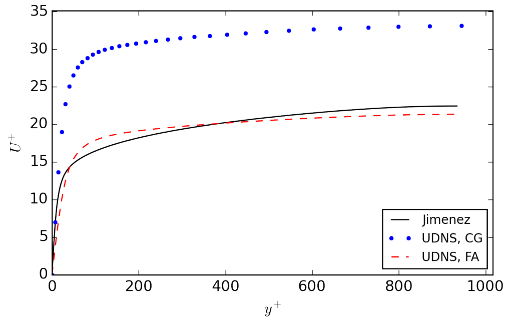
(c) Mean Reynolds stress $\overline{v'v'}$ for UDNS, Smagorinsky and Dynamic Lagrangian.

Figure 6.9: Channel flow, $Re_\tau = 590$; Reynolds stresses for UDNS, Smagorinsky and Dynamic Lagrangian.

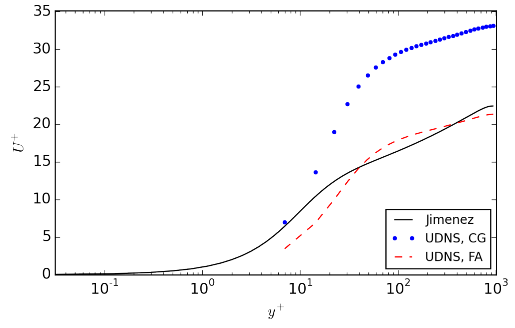
under-resolved DNS, Smagorinsky, and the Dynamic Lagrangian method. The results are very similar to those obtained for $Re_\tau = 395$: the Smagorinsky model produces better profiles compared to UDNS, where, quite surprisingly, the Dynamic Lagrangian model produces the best mean profiles for $\overline{v'v'}$ and $\overline{w'w'}$. The same model does also produce lower values for $\overline{u'u'}$ compared to the σ model for $Re_\tau = 395$, where the results here places themselves between the UDNS and Smagorinsky profiles. These lowered values may be connected to the improved mean velocity data seen for this model in the same region.

6.1.4 $Re_\tau = 950$

Moving on to $Re_\tau = 950$ the results are very similar to earlier ones: the data from the under-resolved DNS simulation positions itself at a higher mean velocity for the CG method, whereas the FA method yields good results right out of the box as seen in Fig. 6.10. As for the LES simulations in Fig. 6.11 it is clear that the Smagorinsky model now works to well in that the flow rate, and followingly the mean velocities, is pushed to mush downwards. For such a high turbulent



(a) CG vs. FA, P_1P_1.



(b) CG vs. FA, P_1P_1, non-log.

Figure 6.10: Channel flow, $Re_\tau = 950$; mean velocities for under-resolved DNS.

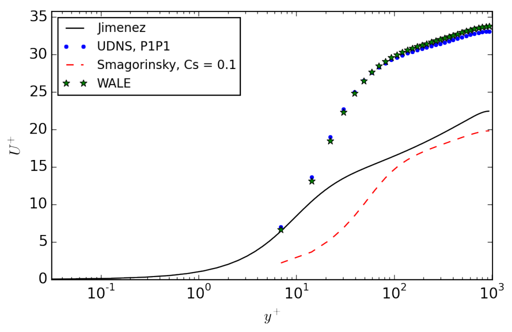
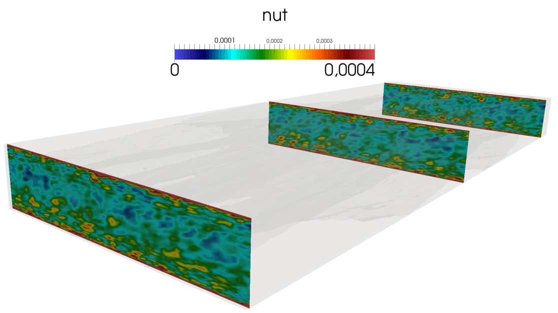


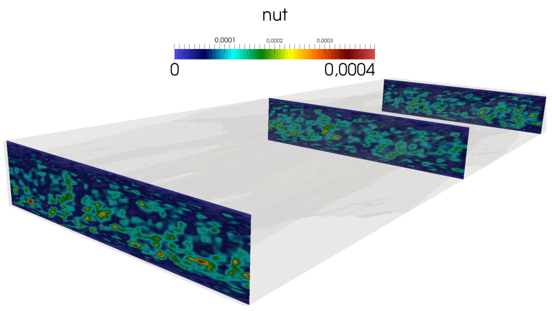
Figure 6.11: Channel flow, $Re_\tau = 950$; mean velocities for Smagorinsky and WALE.

Reynolds number it is most likely so that, in despite of a Smagorinsky constant value of $C_s = 0.1$, the models differential operator $|\overline{S}|$ returns high values near the boundary as a result of the large velocity gradients in this region. Thus, the eddy viscosity contribution becomes large, and a more noticeable reduction is seen in the mean velocity data for this case, compared to what was seen for the three previous Re_τ. As for the WALE model we get the same results as seen for the $Re_\tau = 180$ case, where the mean velocity profile also here places itself above that of under-resolved DNS.

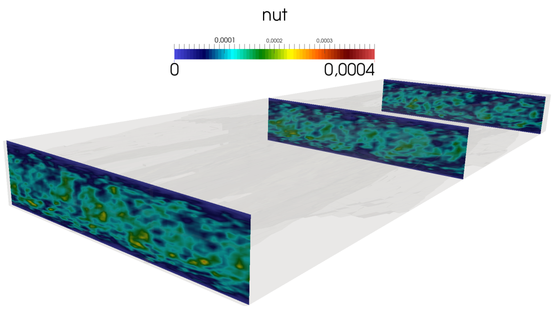
In Figs. 6.12 **(a)** to **(e)** the eddy viscosities for Smagorinsky, WALE, Sigma and Dynamic Lagrangian have been plotted for a certain time step at three different slices in the domain. Notice how the near wall-contribution for the Smagorinsky model is quite large, whereas it for the three other models is nearly non-existent; it seems very likely that the obtained mean velocities has a connection to the models different wall behaviour. In the middle of the channel it is evident that the σ and the Smagorinsky model are more similar in their contribution, whereas WALE is the more conservative of these three overall. The



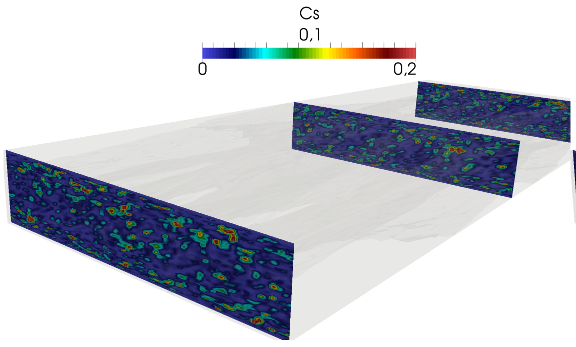
(a) Smagorinsky, $C_s = 0.16$.



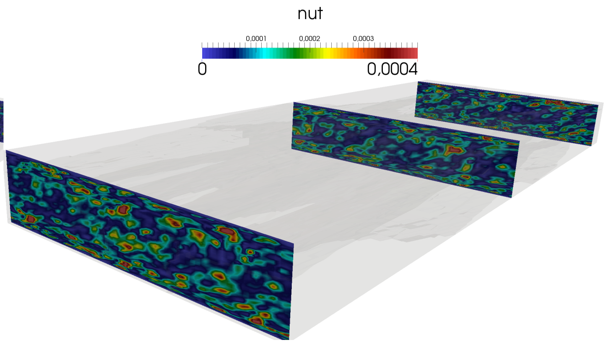
(b) WALE, $C_w = 0.5$.



(c) σ, $C_\sigma = 1.5$.



(d) Dynamic Lagrangian, C_s.

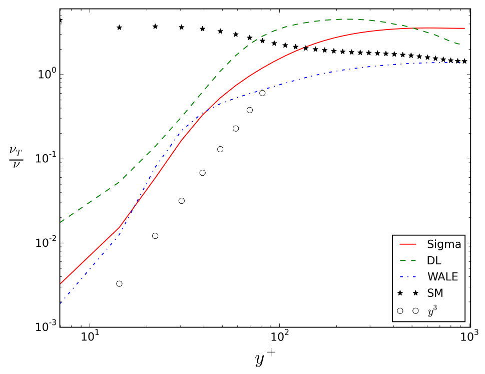


(e) Dynamic Lagrangian, ν_T.

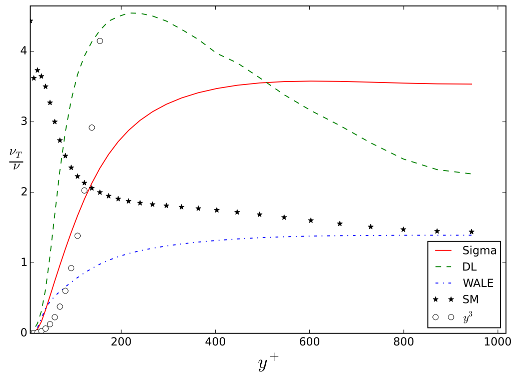
Figure 6.12: Channel flow, $Re_\tau = 950$; slice plots for ν_T at three different locations. Smagorinsky **(a)**, WALE **(b)**, Sigma (σ; c), Dynamic Lagrangian (d; C_s) and **(e)** .

Dynamic Smagorinsky model with Lagrangian averaging shows good results for C_s and ν_T, both being of correct order and in the same vicinity as the other models. Maximum value for C_s varied a bit during the simulation, ranging from everything between 0.2 and around 1. Higher values are not seen on as artefacts, but rather as results of the computation and the dynamic method's ability to compute values of C_s, both large and small. However, it may be so that values of $C_s \simeq 1$ are too high and should be clipped, e.g. to a maximum of 0.4 or 0.5.

In Fig. 6.13 **(a)** and **(b)** the normalized mean value ν_T/ν has been plotted for all the four models for $Re_\tau = 950$. First notice how the wall behaviour of



(a) Means of ν_T, log-log plot.



(b) Means of ν_T, regular (non-log) plot.

Figure 6.13: Channel flow, $Re_\tau = 950$; means of ν_T for Smagorinsky (SM), WALE, Sigma, and Dynamic Lagrangian (DL). The profile for y^3 is also included.

all of them is indeed differing a lot: Smagorinsky (SM) does in general have a wall scaling of y^0, whereas the others seem to follow the scaling of y^3 somewhat correctly. WALE is quite conservative overall, merely adding a mean value of one times ν in the middle of the channel. σ is about three times higher than WALE in the middle of the domain, whereas Dynamic Smagorinsky (DL, here C_s is updated each 10th time step) has the highest peak at around $y^+ = 200$, but decreases as one moves towards the centre of the channel. It is interesting to see how SM yields good mean velocity results, whereas both DL and σ, which have a higher eddy viscosity contribution in the middle of the channel, both yield erroneous results for the mean velocity profiles. The only exception is the Dynamic Lagrangian model for $Re_\tau = 590$ that in general produced a small, but noticeable, positive effect; from Fig. 6.13 we see that it is the dynamic model that actually has the highest contribution closer to the wall. Most likely the addition of eddy viscosity in the near wall region is critical for the simulation results, whereas the amount added in the outer region is somewhat irrelevant (as long as nothing is added near enough to the wall). This is at least true for the mean velocities and the Reynolds stress $\overline{u'u'}$; as seen from Figs. 6.6 and 6.9 the σ and Dynamic Lagrangian models did indeed produce better mean profiles for the two Reynolds stresses $\overline{v'v'}$ and $\overline{w'w'}$, independent of the little or no improved mean velocity profiles.

6.1.5 Discussion

The Smagorinsky model returned good results for all turbulent Reynolds numbers, in general producing a lower valued mean velocity profile closer to the DNS data. Good is here said in the sense that the model behaves as expected without applying damping-functions for ν_T. Its problems are, however, quite clear, not only for its differential operator $|\overline{S}|$, but also in terms of the general constant C_s. Where mean velocity for $Re_\tau = 950$ was altered too much for $C_s = 0.1$, the opposite happened for the three other Re_τ. In addition, the boundary layer problems are evident, as too much eddy viscosity is added in the already well-represented near-wall regions. On the other hand, the more conservative methods like WALE, σ, and the Dynamic Smagorinsky model, lead to overall higher valued mean velocity profiles for all Re_τ.

A lot of time has been spent on finding the "error" and an explanation to this quite counter intuitive phenomenon. At first (and as mentioned in the companion article to this thesis, [6]) when simulations only had been done for $Re_\tau = 180$, it was assumed that this could have something to do with the mentioned Re_τ being too low, eventually requiring only small amounts of modelling from the eddy viscosity contributions. This assumption was rapidly eliminated as the exact same results for these models were obtained for $Re_\tau = 395$, $Re_\tau = 590$ and $Re_\tau = 950$. The next natural assumption was errors either in the implementations of the general eddy viscosity expression, the models, or in the simulation setup. However, the good results obtained in the previous chapter, together with those obtained here for under-resolved DNS and the Smagorinsky model quickly debunks all of those assumptions. In addition, it is clear that this behaviour is common for the WALE, σ and Dynamic Smagorinsky models, thus it is most likely not an implementational error in those models. For testing purposes some different implementations for WALE were tested, where slightly expressions for the differential operator was tried out. This did, however, lead to the exact same results as originally obtained.

As a result of these investigations, all arrows pointed towards a certain phenomenon occurring for this exact case when LES models with a certain type of behaviour was applied. The Kinetic-Energy SGS model was also tested, where the obtained results were very similar to those of the Smagorinsky model. Both these models includes the mentioned wall problems, whereas the WALE, σ and Dynamic Smagorinsky models are all behaving properly near the wall. Therefore, the problem was postulated to have a connection to the wall behaviour of the applied LES model.

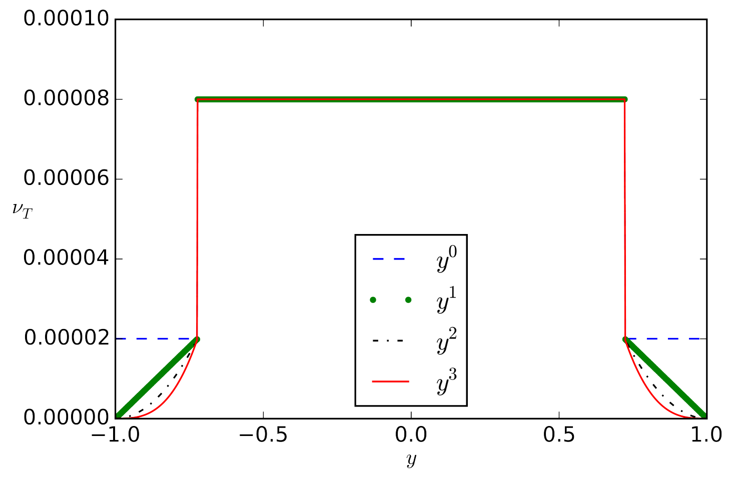
After a lot of searching around the literature, it was eventually discovered that some people had experienced similar problems. Schmidt et al. [68] experienced too high mean velocities when a specific version of the Dynamic Smagorinsky model was applied with a spectral element solver, and Cottet et al. [69] obtained too high mean velocities for the Dynamic Smagorinsky model applied with a fourth-order finite difference method. On the other hand, Menevau et al. [36] did in their paper obtain good results for $Re_\tau = 650$, then applying the same spectral code as was done by MKM [22].

The most interesting results has been found in the 2014 PHD thesis of Lampitella [70], situated at the Polytechnic University of Milan. He does turbulent channel flow applying a Finite Volume code, together with the σ model and Dynamic

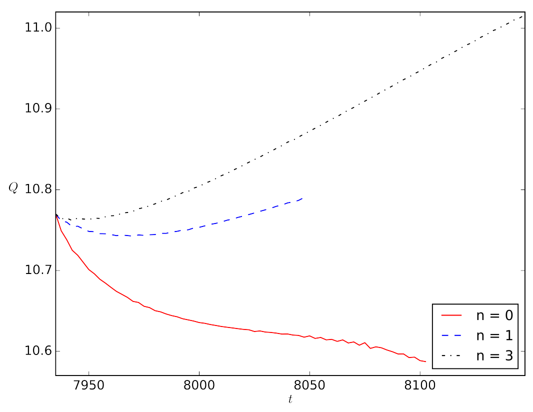
Smagorinsky model among others. The results there show that both these models yield higher mean velocity data, compared to that of under-resolved DNS. He gives a brief explanation in that the required behaviour of y^3 scaling for ν_T near walls (Chapman and Kuhn, [38]) appears overrated, and may be entirely wrong, as it possibly depends on the solver, on the mesh, and on the flow itself. The problem lies in the fact that for this flow not all scales are reproduced near the wall on a coarse mesh/lower order solver; hence, no eddy viscosity is added there, as there are unresolved fluctuations closer to the wall. In addition, all models which return bad results follow the y^3 scaling near the wall, something which eventually results in unwanted behaviour. Lampitella [70] shows that the models that includes a net contribution of eddy viscosity closer to the wall results in positive solutions, thus it is evident that the y^3 behaviour of WALE, σ, the Dynamic Smagorinsky model and so on is erroneous when scales are unresolved near the wall. This also explains why the Smagorinsky model (scales as y^0), and the Vreman model ([50], scales as y^1, simulations done by [70]) in general yields solutions that are more correct. In addition, the positive channel flow results obtained by Meneveau et al. [36] (Lagrangian Dynamic model), are most likely a product of them applying a computational code of spectral accuracy, leading to the y^3 scaling being the correct rate of decay for ν_T since most fluctuations near the wall are resolved.

As briefly discussed in section 3.4.1, LES comes packed with some quite demanding factors in terms of wall resolution and so on. It may be that what is experienced here actually is directly connected to the mentioned wall problems, and that, for this case, if scales near the walls are left unresolved, the eddy viscosity models with correct behaviour is unable to correctly control the simulation overall. For the higher turbulent Reynolds numbers, like 590 and 950, this could be the case as the thickness of the boundary layer δ is small (the first computational node is located at $y^+ \simeq 4.3$ for the former, and at $y^+ \simeq 14.3$ for the latter); however, for $Re_\tau = 180$ where the boundary layer is thicker, it is not as clear why this happens compared to the two other cases (the first computational node is, however, still located outside the crucial region, at $y^+ \simeq 1.31$). Some good results were obtained with the WALE model for a P_2P_1 $Re_\tau = 180$ simulation on a coarse mesh of 32^3 cells, where a slightly positive effect was seen for the mean velocity profile. Nevertheless, this was most likely a result of the P_1 Lagrange elements being applied for ν_T for the coarse mesh, thus its eddy viscosity contribution became more inaccurate, and a certain amount was added closer to the wall, therefore resulting in improved mean data.

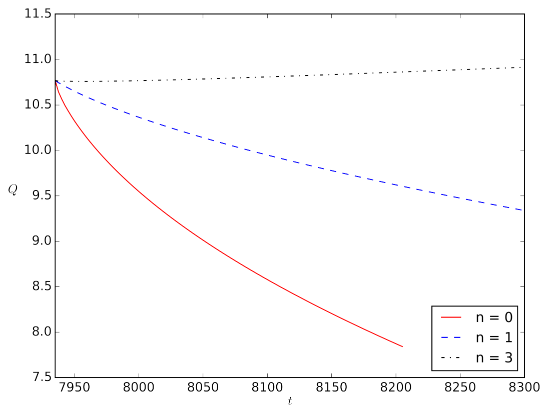
To test if the problem with increased flux and higher mean velocities for some models is reproducible, an expression for ν_T is fabricated that includes decaying as y^n in the viscous sub- and buffer-layers, for $y^+ < 40$ approximately. Outside this layer the eddy viscosity is set equal to $\beta\nu$, whereas it for the inner layer has the constant value κ as coefficient. Both these constant values somewhat depends on Re_τ, but for testing purposes they were here chosen to be equal to $\beta = 4$ and



(a) Decaying ν_T; Eq. (6.12).



(b) Dev. of Q for ν_T as in **(a)**.



(c) Dev. of Q for ν_T in modified **(a)**.

Figure 6.14: **(a)**: Fabricated ν_T from Eq. (6.12). **(b)**: Initial flux development for Eq. (6.12) with $\kappa = 1$, $\beta = 4$ and $Re_\tau = 950$. **(c)**: Initial flux development for modified Eq. (6.12) with $\kappa = 10$, $\beta = 4$, $Re_\tau = 950$, and decaying zone for $y^+ \in [0, 60]$.

$\kappa = 1$. More precisely the function is defined as

$$\nu_T = \begin{cases} \beta\nu \text{ if } y^+ \geq 40 \\ \kappa\nu \left(y/\left(y^+ Re_\tau\right)\right)^n \text{ if } y^+ < 40) \end{cases}, \tag{6.12}$$

where n is equal to the wanted rate of decay. The function is plotted in Fig. 6.14 **(a)** for $n = 0, 1, 2, 3$, $\beta = 4, \kappa = 1$ and $Re_\tau = 180$. For these values of β and κ it is assumed that the eddy viscosity has an approximate mean value of $\beta\nu$ in the middle of the channel, where the decaying starts from $\kappa\nu$ at $y^+ = 40$.

Results for $Re_\tau = 950$ in Fig. 6.14 **(b)** shows that the flux does indeed for $n = 0$ and $n = 1$ decrease at start-up. For $n = 3$ the flux increases rapidly, clearly illustrating the problem with the wall decaying behaviour of y^3 for ν_T. We also after some time see an increase in the flux for $n = 1$, whereas for $n = 0$ it keeps going downwards.

Modifying Eq. (6.12) such that $\kappa = 10$ (which results in the decaying starting at 10ν, a much higher value compared to those seen in Figs. 6.13 **(a)** and **(b)**, this is for illustrational purposes only), combined with altering Eq. (6.12) such that the decaying zone instead is for the region $y^+ \in [0, 60)$ (justified from the mean profiles in Figs. 6.13 **(a)** and 6.13 **(b)**), yields the results showed in Fig. 6.14 **(c)**. Again we see how the flux rapidly decreases for $n = 0$ and $n = 1$, whereas it slightly increases for $n = 3$. Clearly the eddy viscosity added in the wall region heavily controls the quality of the simulations, justifying the mean data obtained for WALE, σ and Dynamic Smagorinsky. In some sense, you can add as much eddy viscosity in the middle of the channel as you want, it will eventually not lead to better results for the mean velocity profiles. For that to happen a net contribution of eddy viscosity close enough to the wall is necessary; that is, not as close to the wall as the Smagorinsky model, but closer to the wall than what is seen for the WALE, σ and Dynamic Smagorinsky models.

A second test where eddy viscosity was added either only in the middle of the channel, or only in the boundary layers, did eventually return results that further strengthen this explanation. If a high amount of eddy viscosity was added only the middle of the channel the only effect that was seen was increased mean velocities and increased flux. On the other hand, if eddy viscosity was added only in the boundary layer, the flux, and followingly the mean velocities, decreased immediately at start-up, after some time stabilising at a lower flow rate.

6.2 The FDA Case

The U.S. Food and Drug Administration (USFDA/FDA)'s computational round robin #1 available at [71], was an international round robin initiative by FDA and the National Cancer Institute put forward in 2008-2009. The case consists of blood flowing in a generic medical device, formed as a pipe with a simple inlet at the left hand side, followed by a conical nozzle, a narrow pipe, and then an expansion to a larger pipe. Boundary conditions are set to no-slip at the pipe walls, zero pressure outlet at the right hand side, and a fixed velocity inlet at the left hand side. Inside the domain, the fluid is at rest at start-up.

All the 28 groups that submitted results in the original round robin used Reynolds-Averaged Navier-Stokes (RANS) methods, where none of the models accurately predicted the breakdown point of the jet for $Re = 3500$ [72]. After the round robin, near perfect results have since been obtained using DNS, LES and hybrid RANS-LES for the same Reynolds number [73, 74, 75]. After the sudden

Table 6.2: Physical and computational parameters for the FDA case.

Re	3500
μ	0.0035 $kg/(s \cdot m)$
ρ	1056 kg/m^3
ν	$3.3 \cdot 10^{-6}$ m^2/s
Inlet flow rate	$3.63 \cdot 10^{-5}$ m^3/s
Time step Δt	10^{-4} s
Simulation time	$\simeq 16 \cdot 10^4 \cdot \Delta t$ s

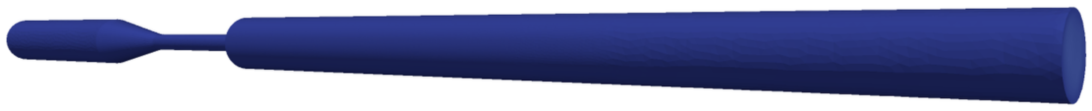


Figure 6.15: Domain for the FDA case.

expansion, a jet will be formed, and a breakdown of the jet will occur at some point. In general, modelling the breakdown of the jet is the hardest part of the simulation, as it is heavily dependent on the transitional process from the laminar flow in the narrow pipe, to the turbulent flow in the jet, eventually leading to a breakdown process. When coarser meshes are applied there are effects in the nozzle and the jet that are unaccounted for, hence, dissipation becomes too low and perturbations are starting to grow too early, resulting in the jet breaking too early. Since the Reynolds number actually is quite low, the main challenge is here to model the transition to turbulence correctly such that the perturbations, which trigger the breakdown of the jet, are initiated at the appropriate time.

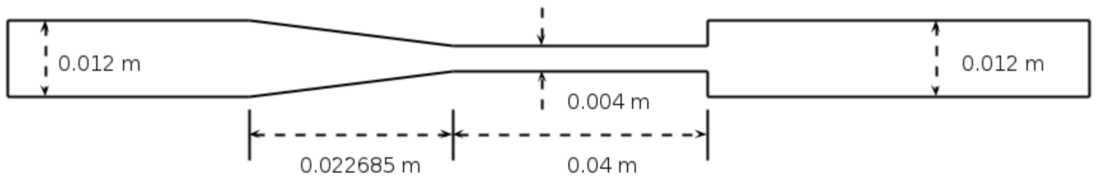


Figure 6.16: Length parameters for the domain.

Experimental data obtained applying Particle Image Velocimetry (PIV) is available through [76]. Here mean velocity data obtained by both under-resolved DNS and LES simulations will be compared to that obtained with PIV, where measurements have been done for all slices between $z = -0.008$ to $z = 0.080$ (see Fig. 6.18). Two types of meshes will be applied; non-uniform meshes of two and three million cells, and uniform meshes of two and three million cells. Applying a P_2P_1 element pair together with a coarse mesh was also tested, but because of stability problems no results will be included here.

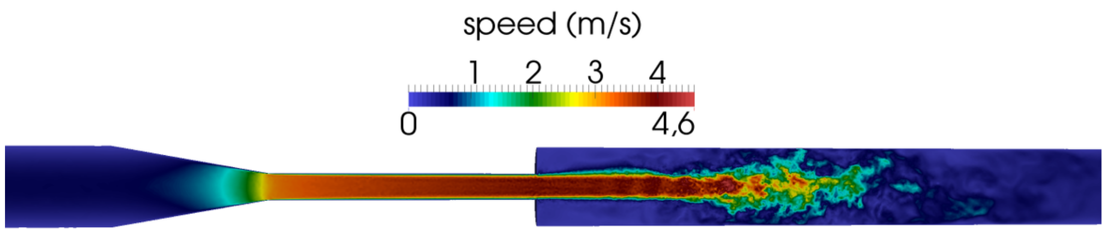


Figure 6.17: Magnitude of the velocity at a certain time step.

As a function of the measured mean velocities, a validation metric value E will be computed and compared for all simulations. Its value determines how close the obtained solutions are to the actual experimental data in total, where a value of $E = 0.0$ reflects a perfect match between experimental and numerical data. Computation of this metric is done applying the relative errors through the formula

$$E = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\overline{u}_i - u_i}{u_i}\right|, \tag{6.13}$$

where n equals the number of discrete points, $\overline{u}_i$ is the mean data from the numerical experiments at location i, and u_i is the experimental mean data at location i. It is evident from the formula that if we have a perfect match between experimental and CFD data $E = 0$, where $E > 0$ for all other cases, and its magnitude then determines how close the CFD data is to the experimental PIV results. In addition to this validation metric a percentage metric P will also be computed as

$$P = \frac{E_{\text{UDNS}} - E_{\text{LES}}}{E_{\text{UDNS}}}, \tag{6.14}$$

such that the overall improvement is quantified through a percentage value. Here E_{UDNS} is the validation metric obtained for an under-resolved DNS simulation on a given mesh, and E_{LES} is the value obtained for the same mesh with some LES model activated.

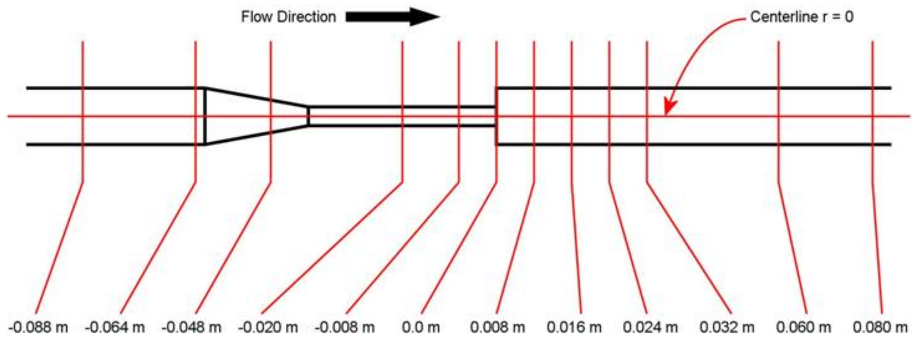


Figure 6.18: Cross section locations for mean data measurements.

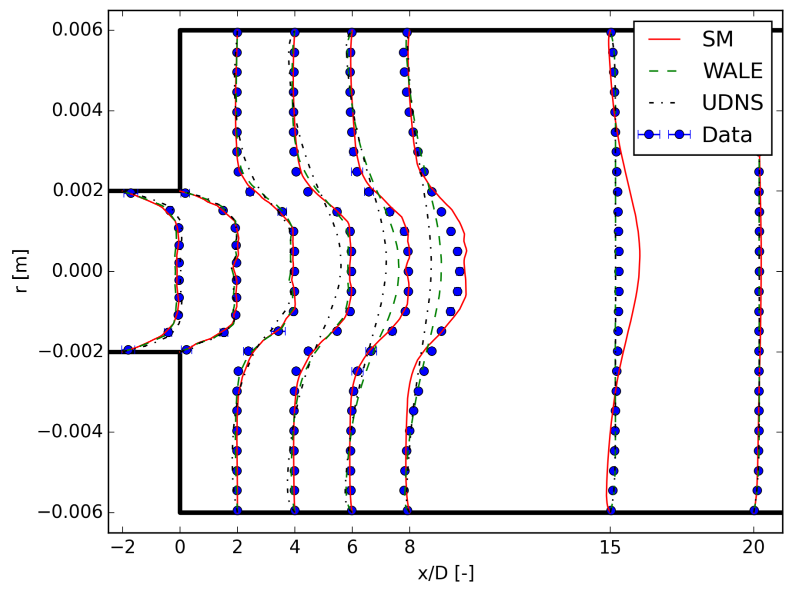
6.2.1 Non-uniform Meshes

First two non-uniform meshes of two and three million cells have been tested. The refinement has primarily been done from $x = 0.0$ and right towards the outlet, with cell size being the smallest closer to the expansion. It is expected that this region is crucial for the simulation as this is the location of the jet, and hence also the region for the breakdown and a lot of turbulent behaviour. Some refinement has also been done in the nozzle, in the narrow pipe, and along the boundary, as these are regions of high velocities and high velocity gradients as well.

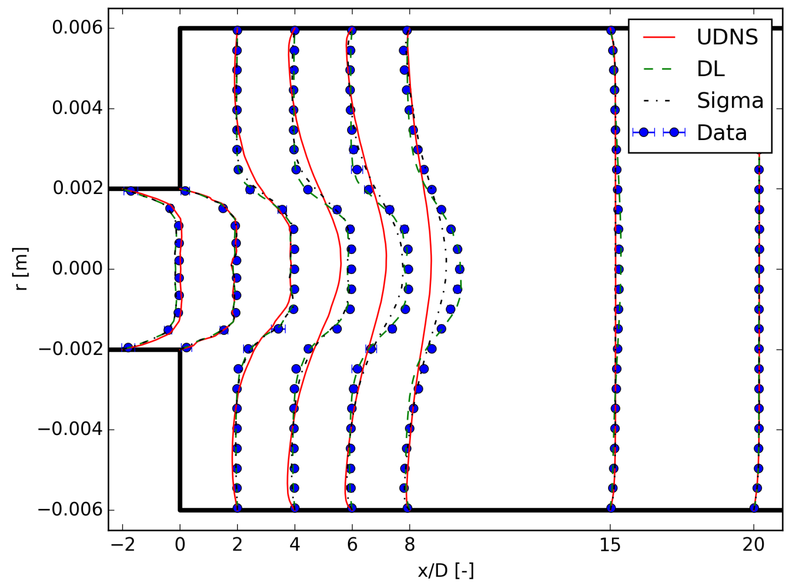
The results from the simulations on the 2M mesh can be seen in Figs. 6.19 **(a)** and **(b)**, and in Table 6.3. First notice how the UDNS case lags behind, breaking the jet too early, hence producing lower mean velocities from slice number four. The Smagorinsky model (SM) is generally over-dissipative, something which for this case results in a better solution compared to UDNS; however, the breaking happens a little too late as seen at the 15th slice. This is most likely a result of the model being overly dissipative (as discussed in Ch. 5), both along the boundary of the jet, and in the boundary regions, as perturbations here to some extent are damped and killed as a result of the over-contribution of ν_T. Moving on to the WALE model the results are not as good as expected, as the improvement over the UDNS profile is evident, but small (only 55% better). Most likely it is a matter of adjustment of C_w such that better results had been obtained, e.g. to $C_w = 0.5$ or even $C_w = 0.6$. Based on the formula given in Eq. (3.29), if one assumes that $C_s = 0.16$ the WALE constant becomes equal to $C_w = 0.5216$, a value which eventually would have returned better results here. Clearly, the problem with the constants in the static models is evident, as they are not completely general, but rather varying as functions of time, space, and maybe also the mesh. For this case the best results are obtained with the Dynamic Smagorinsky model with Lagrangian averaging, as it returns a validation metric of $E = 0.50$, and thus a percentage improvement of $P = 0.804$. Compared to the WALE model, also σ does here result in a total eddy viscosity contribution that is too low, in general producing profiles that are better compared to UDNS and WALE, but which still

Table 6.3: Validation metrics E and P for non-uniform mesh, **two** million cells.

LES Model	E	P
None (UDNS)	2.56	-
Smagorinsky, $C_s = 0.1$	0.82	0.671
WALE, $C_w = 0.325$	1.13	0.558
Sigma (σ), $C_\sigma = 1.6$	0.92	0.64
Dynamic Smagorinsky (Lagrangian)	0.50	0.804



(a) Mean velocity at slices; Underresolved DNS (UDNS) vs. Smagorinsky (SM) vs. WALE vs. Experimental (Data).



(b) Mean velocity at slices; Underresolved DNS (UDNS) vs. Sigma vs. Dynamic Smagorinsky (DL) vs. Experimental (Data).

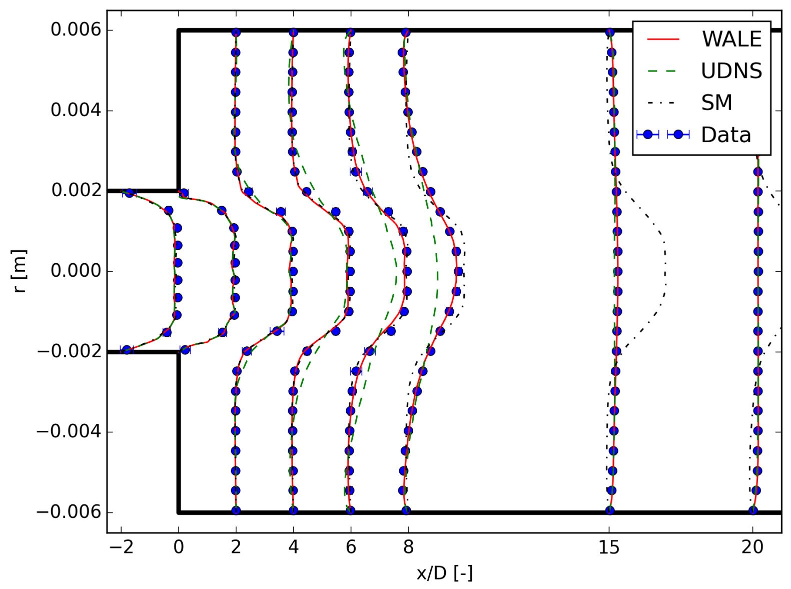
Figure 6.19: FDA case, slice plot of mean velocity data for a non-uniform mesh of two million cells.

position themselves somewhat behind the experimental data. Again it seems like a matter of adjusting C_σ to obtain better results: the constant was here equal to $C_\sigma = 1.6$, increasing it somewhat to 1.7 or 1.8 would possible have returned better data.

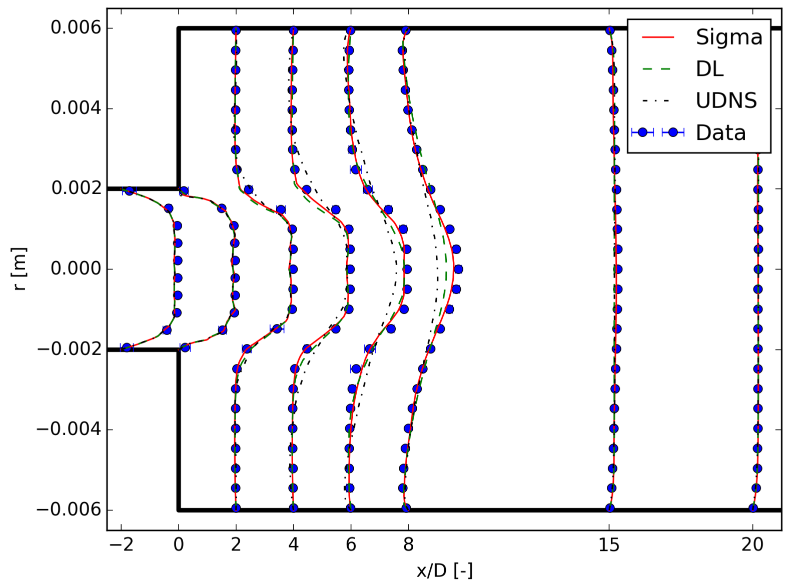
Moving on to the mesh of three million cells some very interesting results are obtained, see Fig. 6.20 **(a)** and **(b)**, and Table 6.4. Again, as expected, the UDNS data is positioned behind the experimental profiles, where the results are better than that of UDNS on the 2M mesh. It is interesting to see how the Smagorinsky model fails drastically for this mesh, damping all perturbations resulting in the jet not being broken at all. Clearly, for the 2M mesh the perturbations were able to partially stay alive in despite of the Smagorinsky model giving them a hard time; nonetheless, for this mesh, the model's ν_T contribution manages to completely damp them, subsequently killing all perturbations, leading to no transition to turbulence at all. WALE returns velocity profiles that are excellent, nearly overlapping the experimental data perfectly. Returning $E = 0.35$, which is the best overall validation metric obtained in this work, the model clearly shows its efficiency and strengths compared to the Smagorinsky model. Zooming in on the mean profiles one may see that they still are somewhat behind those of the experimental data, hence slightly increasing the value of C_w would also for this case most likely have returned near to perfect results overall. Both the σ and the Dynamic Smagorinsky models do here return good results, where the σ model in general shows the same behaviour as the WALE model does. It returns better results, $C_\sigma = 1.6$ is still too low, a small increment would have increased the effects, and thus returned better results. The Dynamic Smagorinsky model positions itself somewhat behind the experimental data, still producing a good validation metric of $E = 0.53$, but a lower percentage metric compared to what was seen for the 2M mesh is obtained here.

Table 6.4: Validation metrics E and P for non-uniform mesh, **three** million cells.

LES Model	E	P
None (UDNS)	1.36	-
Smagorinsky, $C_s = 0.1$	-	-
WALE, $C_w = 0.35$	0.35	0.742
Sigma (σ), $C_\sigma = 1.6$	0.52	0.617
Dynamic Smagorinsky (Lagrangian)	0.53	0.610



(a) Mean velocity at slices; Underresolved DNS (UDNS) vs. Smagorinsky (SM) vs. WALE vs. Experimental (Data).



(b) Mean velocity at slices; Underresolved DNS (UDNS) vs. Sigma vs. Dynamic Smagorinsky (DL) vs. Experimental (Data).

Figure 6.20: FDA case, slice plot of mean velocity data for a non-uniform mesh of three million cells.

6.2.2 Uniform Meshes

Secondly, two meshes of two and three million cells where all cells are of uniform size have been tested. For this case, we omit the Smagorinsky model, as the results most likely would have been quite similar to those obtained in the previous section. The focus will be on WALE, σ, and the Dynamic Smagorinsky models only. In general, the LES models should have a harder time modelling the breakdown location, then as a result of the under-resolved DNS data being much farther behind the experimental data compared to the two refined meshes. Expectations in terms of the LES results have to be adjusted accordingly.

For the uniform mesh of two million cells validation metrics can be found in Table 6.5, mean velocity at the slices can be seen in Fig. 6.21. Compared to the

Table 6.5: Validation metrics E and P for uniform mesh, **two** million cells.

LES Model	E	P
None (UDNS)	4.59	-
WALE ($C_w = 0.5$)	1.22	0.734
Sigma (σ, $C_\sigma = 1.6$)	1.27	0.726
Dynamic Smagorinsky (Lagrangian)	0.97	0.788

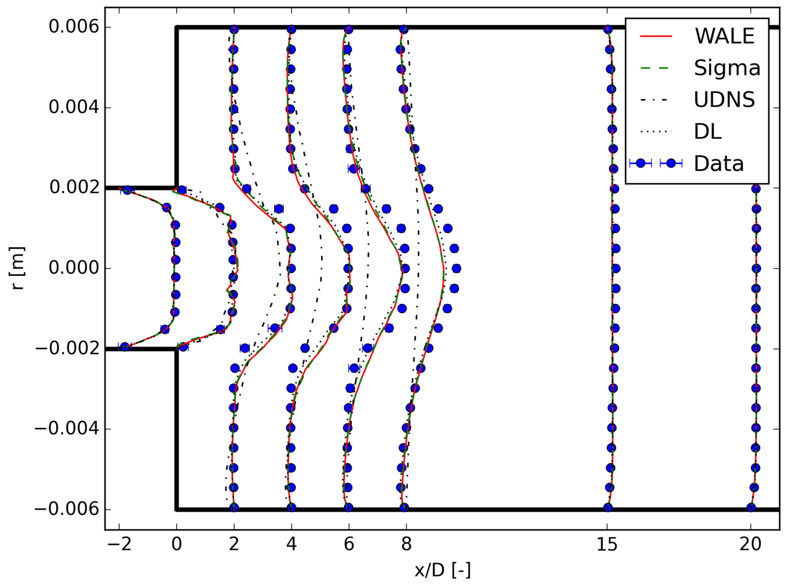


Figure 6.21: Mean velocity at slices; 2M Uniform; Underresolved DNS (UDNS) vs. WALE vs. Sigma vs. Dynamic Smagorinsky (DL) vs. Experimental (Data)

non-uniform meshes the under-resolved DNS solution clearly suffers from the cell size in the region following the expansion, where the validation metric clocks in at $E_{\text{DNS}} = 4.59$, about 80% worse than that of the non-uniform mesh of two million cells. This is expected behaviour as the cell size both in the nozzle, and near the breakdown location, is much larger compared to the non-uniform mesh. Notice from Fig. 6.21 how UDNS predicts that the breakdown process starts already at slice number two, leading to the jet having been almost completely dissipated at slice number six. Compared to the data obtained for the refined meshes it is quite clear that the UDNS simulations here are substantially worse compared to those obtained for non-uniform meshes. As for the LES models it is clear that all of them under-predict the mean position for the jet breakdown. As for the static models, adjustment of the constants would also here most likely have

Table 6.6: Validation metrics E and P for uniform mesh, **three** million cells.

LES Model	E	P
None (UDNS)	3.80	-
WALE ($C_w = 0.5$)	0.946	0.749
Sigma ($\sigma,\ C_\sigma = 1.6$)	1.16	0.690
Dynamic Smagorinsky (Lagrangian)	1.16	0.690

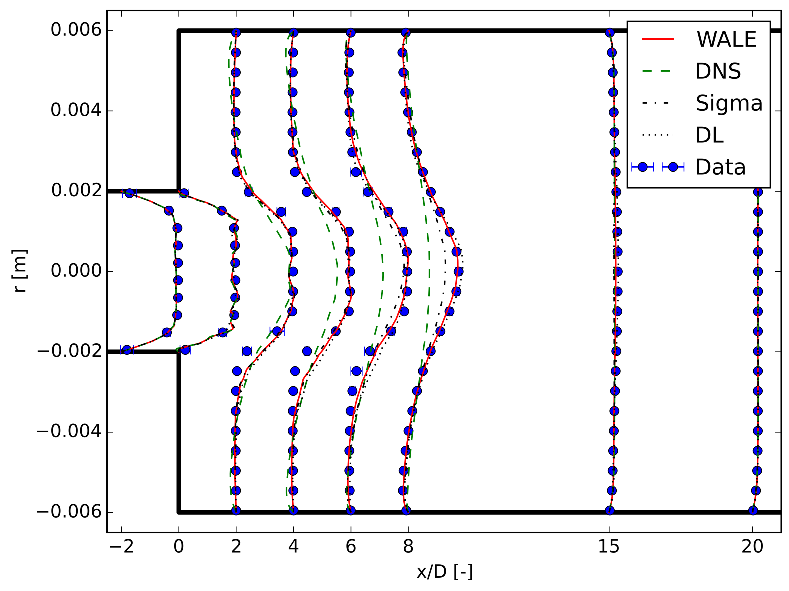


Figure 6.22: Mean velocity at slices; 3M Uniform; Underresolved DNS (UDNS) vs. WALE vs. Sigma vs. Dynamic Smagorinsky (DL) vs. Experimental (Data)

returned better results. On the other hand, the Dynamic Smagorinsky model again presents us with the best solution, resulting in $E = 0.97$, and a percentage improvement of $P = 0.788$. In total all models yields over 70% improvement from the under-resolved data, a number which is quite high, and good.

Moving on to the uniform mesh of three million cells we first from Table 6.6 see that the under-resolved DNS data is somewhat improved from the previous case, but not as drastically as it was for the non-uniform meshes. As for the LES models, it is actually the WALE model that obtains the lowest validation metric, and highest percentage improvement, here. From the plot in Fig. 6.22 it is evident that the breaking of the jet happens a little too early for the σ model, a little too late for the Dynamic Smagorinsky model, but nearly perfectly for the WALE model. The errors in the computations are clearly seen at slice number two, four and six, where in the regions $\pm[0.002, 0.004]$ all the models return profiles very similar to that of under-resolved DNS. This is without a doubt the main source of the errors for all the three models here, where, as seen from the plots, the WALE model manages to capture the position of the location of the breakdown better than the two other models. Compared to the results obtained for the uniform two million mesh the percentage improvements are worse, but the overall results are better in that the breakdown location is recovered more correctly.

6.2.3 Discussion

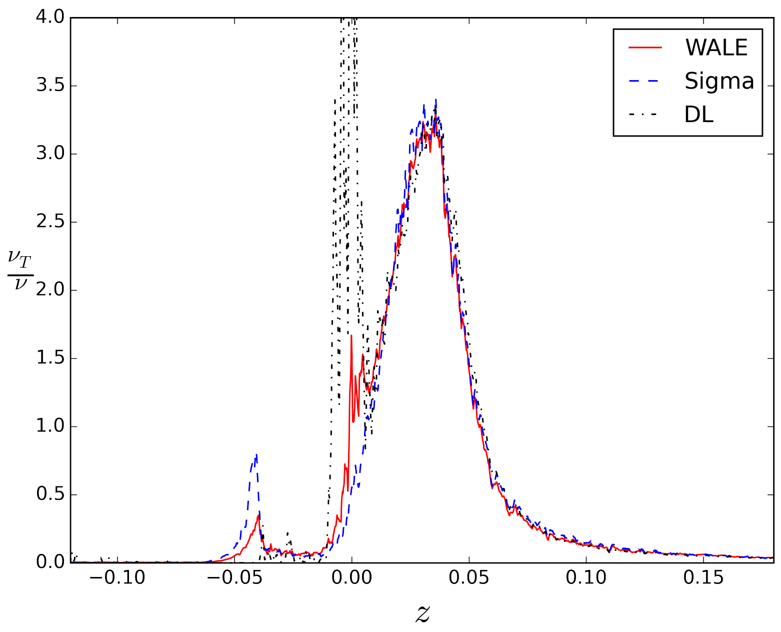
In general, as is clear from the validation metrics, all the LES models are able to recover a quite high amount of the left out dissipation not captured in the under-resolved DNS simulations. The results are varying in the parameter P, where the improvement in the results range between 50 and 80 percent. The worst results are seen for the WALE model on the non-uniform mesh of 2M cells where $P = 0.531$, where on the other hand the best results are seen with the Dynamic Smagorinsky model with $P = 0.804$ on the uniform mesh of 2M cells. Inconsistency is a problem that is evident here, especially for the Dynamic Smagorinsky model; e.g., in the non-uniform case, good results was obtained for the 2M mesh, whereas they were slightly worse for the 3M mesh. The same tendencies was seen for the uniform simulations. As for the σ and WALE models the results for the coarser meshes are unsatisfying, whereas they improve a lot when a finer mesh is applied, i.e. the models do not have optimal behaviour in terms of e.g. mesh independence and so on.

When it comes to the two different types of meshes applied here, there is no doubt that the non-uniform refined meshes in general returns better solutions, compared to their uniform counterparts. Coupled with the inconsistency problem it is clear that mesh dependency is quite clear, especially in terms of the validation metric E. If one rather focus on the percentage metric P, it is clear that the LES models do in general come closer to the experimental data for the two uniform meshes, then with a mean value of $\overline{P}_u = 0.729$, compared to the mean value for the non-uniform meshes $\overline{P}_{nu} = 0.661$. These values are clearly connected to the validation metrics obtained for the UDNS simulations, as their values were much lower for the refined meshes, therefore, since the UDNS simulations on the uniform meshes are much worse, the LES models are able to restore a larger amount of the left out dissipation.

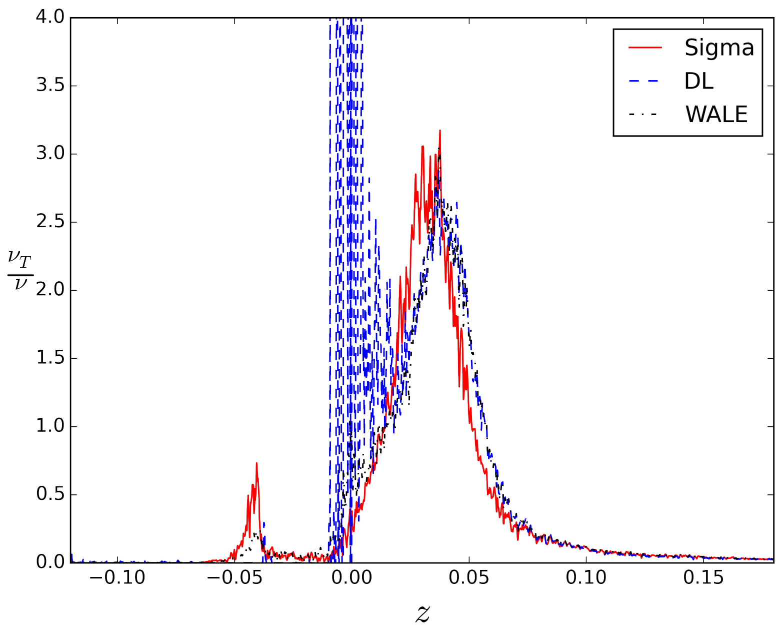
Mean measurements of ν_T along the centreline and at the slices were done for the uniform 2M and 3M simulations, such that some information was obtained regarding the different models eddy viscosity magnitudes and their general contributions. Centreline plots of ν_T can be found in Figs. 6.23 **(a)** and 6.23 **(b)**. The first thing to notice is how the behaviour of the models are quite similar, for both cases, both in terms of overall magnitudes and contribution at all locations. For the 2M mesh the normalized values of ν_T/ν in the jet is around 3, whereas they for the 3M mesh are slightly lower, hovering around 2.5. Both to the left and to the right of the breakdown the overall contribution is near zero, only disturbed by a few spikes here and there.

The only behaviour that differs is the huge spike seen in the Dynamic Lagrangian model (DL) for both meshes at $x = 0.0$, where it is substantially worse for the 3M mesh. There is also a spike at the exact same location for WALE, though it is small compared to the DL model, which there produces a ν_T value over 20 times higher than ν. Most likely it is a type of artefact from the Lagrangian equations and their clipping procedure, which, if J_{MM} is small but J_{LM} is high, may produce extreme values (often removed by clipping C_s, this is not done here). This should absolutely be investigated, as such values may cause problems with the simulations if they are present at locations that heavily controls the overall results. From the results for the 2M simulations it does not look like it has had a huge impact on the simulations in total; however, the results obtained for the 3M mesh may have been somewhat biased by this instability, as the DL model produces the smallest amount of eddy viscosity in the jet, but still results in a later breakdown compared to the other models.

In Fig. 6.24 the instantaneous velocity field and the corresponding ν_T fields for Smagorinsky, WALE, σ, and Dynamic Lagrangian models have been plotted for the non-uniform mesh of two million cells. The velocity field is obtained from a simulation applying the σ model, where then the ν_T plots have been produced applying this velocity field at a certain time step. If we start with the Smagorinsky model (plot number two from the left), it is clear that the eddy viscosity contributions in the near-wall region in the narrow pipe and along the boundary of the jet, are causing this model to return poor results. For this case the results were not as bad as for the non-uniform 3M mesh, where these erroneous contributions there led to no breakdown at all. The contributions from the WALE and the σ models (plots number three and four) are very similar, the σ model seemingly having a slightly higher contribution overall. In the narrow pipe, the WALE model adds some small amount of eddy viscosity in the middle, but nothing near the wall as expected. σ has little or no contributions in the



(a) Mean ν_T along centreline; 2M uniform mesh.



(b) Mean ν_T along centreline; 3M uniform mesh.

Figure 6.23: FDA case, mean plot of ν_T along centreline of the pipe. Uniform meshes of two million cells **(a)** and three million cells **(b)**.

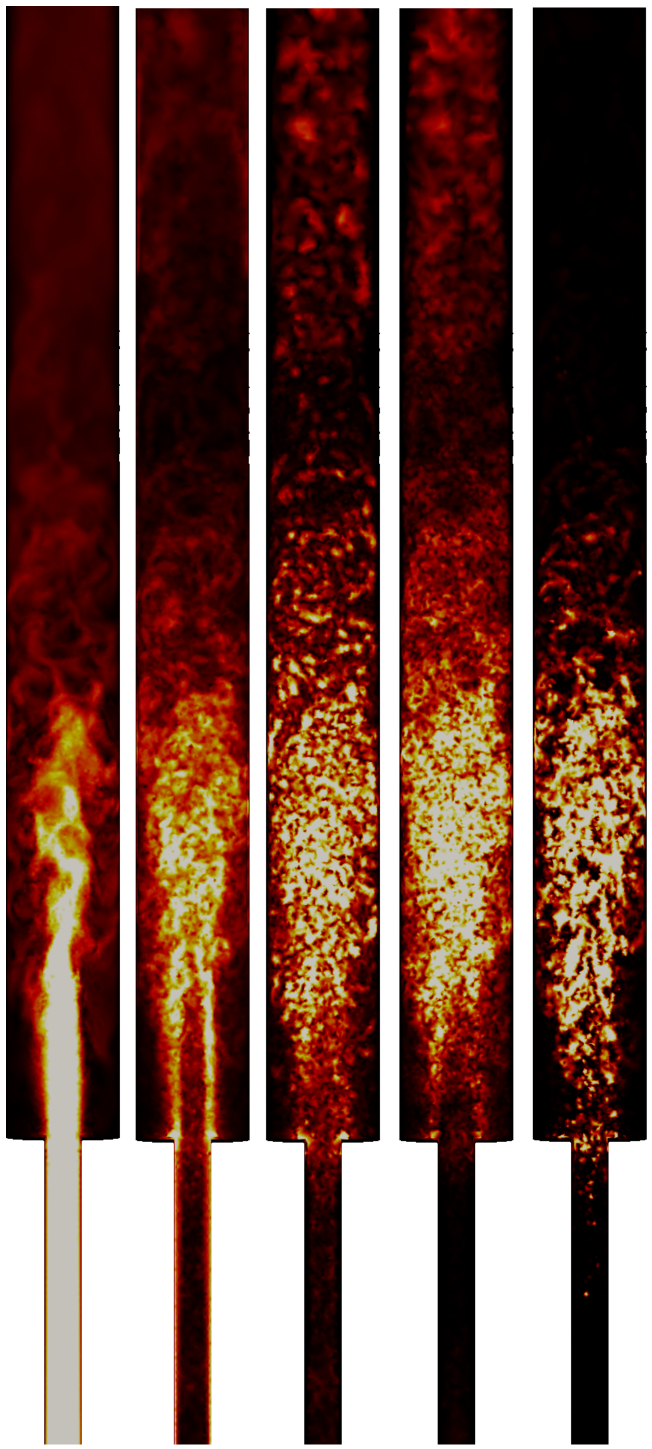
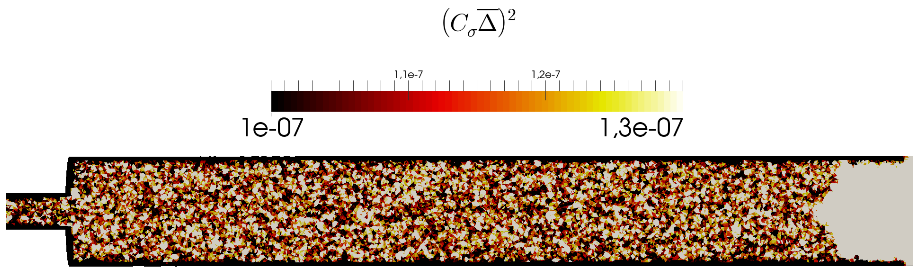


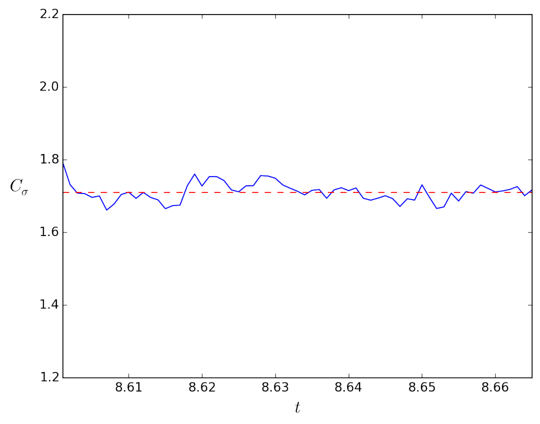
Figure 6.24: FDA case; instantaneous plots of (from left to right) the velocity field, followed by the ν_T fields for Smagorinsky, WALE, σ, and Dynamic Lagrangian model. ν_T is here ranged from 0 to $5 \cdot 10^{-6}$ for all cases.

narrow part. The Dynamic Lagrangian model is the most conservative model overall, where the contribution in the jet itself is similar to the other models, but little or no eddy viscosity contribution is seen in the narrow pipe or closer to the outlet. Overall the models do return good results and show nice tendencies, not only in terms of the values for ν_T and general behaviour, but also for specific and expected behaviour as near-wall contributions and similar. Identifying which effects that specifically control the simulations in total is easy for the Smagorinsky model, but not as easy or clear for the three other models. The addition of eddy viscosity in the jet has without a doubt a major impact on the results: however, why e.g. the σ model consequently returns worse results than the WALE model, remains somewhat unclear.

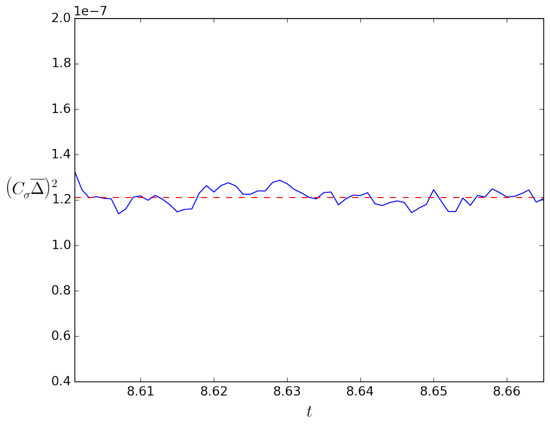
As already mentioned, there is in general a problem with the static models,



(a) Plot of $\left(C_\sigma\overline{\Delta}\right)^2$for $C_\sigma = 1.66$ and $\overline{\Delta}$ as in Eq. (3.25).



(b) Plot of dynamically computed C_σ as a function of time.



(c) Plot of dynamically computed $\left(C_\sigma\overline{\Delta}\right)^2$ as a function of time.

Figure 6.25: The global dynamic σ model applied together with the uniform mesh of two million cells. **(a)**: Plot of constant $C_\sigma = 1.66$ and $\overline{\Delta}$ given in Eq. (3.25), **(b)**: Plot of dynamically computed C_σ with a mean value of $\overline{C}_\sigma = 1.72$, and **(c)**: Plot of dynamically computed $\left(C_\sigma\overline{\Delta}\right)^2$ with a mean value of $\overline{\left(C_\sigma\overline{\Delta}\right)^2} = 1.21 \cdot 10^{-7}$.

in that these models constants are not constant in time, nor in the domain (not strictly true, it depends on the differential operator for the given model). A global dynamic approach was discussed in section 3.4.2, where, if the general dynamic procedure is coupled with the e.g. the σ model, one gets a global dynamic sigma model if the tensor contractions are averaged over the entire computational domain. This model can then be used to dynamically calculate either C_σ, or $\left(C_\sigma\overline{\Delta}\right)^2$. Whatever is best needs to be verified, but it is in general positive that the length scale does not need to be explicitly specified.

To test how this model performed, it was applied together with the non-uniform mesh of two million cells, where again tests were performed for solving for C_σ only, and for solving for the term $\left(C_\sigma\overline{\Delta}\right)^2$ directly. Results can be seen in Fig. 6.25: in **(a)** the field $\left(C_\sigma\overline{\Delta}\right)^2$ has been plotted for the values $C_\sigma = 1.66$ and $\overline{\Delta}$ as in Eq. (3.25). In 6.25 **(b)** and **(c)** we have the plots for dynamically computed C_σ and $\left(C_\sigma\overline{\Delta}\right)^2$, respectively. The first thing to notice is how this global dynamic procedure returns nice values for C_σ, where they hoover around the mean value $\overline{C}_\sigma \simeq 1.72$, with a maximum of $C_{\sigma,max} \sim 1.8$, and a minimum of $C_{\sigma.min} \sim 1.66$. These values are in good agreement with the value of $C_\sigma = 1.5$ presented in the original paper [33]; however, for this case they become slightly higher, something which was expected as the mean velocity results obtained for this case with $C_\sigma = 1.6$ all point towards this being a too conservative value. As for solving for $\left(C_\sigma\overline{\Delta}\right)^2$ directly the results, as seen in Fig. 6.25 **(c)**, are both surprisingly good, and somewhat disappointing. They are good in that, if compared to the plotted field in Fig. 6.25 **(a)**, the results are very similar, showing that there is a good connection between the explicitly defined $\overline{\Delta}$, and the value provided by the dynamic procedure. In addition, it is clear from both plot **(b)** and **(c)** that the two dynamic computations behave almost identically. On the other hand, the results are somewhat disappointing in that the values obtained are very similar to those applied for the static model. If we compare the dynamically computed $\left(C_\sigma\overline{\Delta}\right)^2$ in **(c)** to the static field in **(a)**, we see that there are some differences, but the values are in general close to each other. None of these global dynamic σ models were tested for a full simulation; hence, no mean velocity data is included for either of them.

One interesting question to ask is how well suited this global approach is for this case? Clearly the computations are more stable, but most likely the values of C_σ and $\left(C_\sigma\overline{\Delta}\right)^2$ would have become locally higher if a local averaging approach had been used; thus, the dissipational effect in the jet would possibly have been more hard hitting, possibly leading to better mean data. This has not been tested, but could be investigated in the future. One, however, then encounters the problem of the dynamic procedure being applied to a model that already satisfies the y^3 wall-decaying behaviour, as discussed by [37].

6.3 Conclusions

This chapter acts as the second part in the process of verifying and validating the general eddy viscosity implementation, and the implementations of the different LES models. The Smagorinsky model, the WALE model, the σ model (plus a dynamic variant), and the Dynamic Smagorinsky model with Lagrangian averaging have all been thoroughly tested for two cases: a traditional benchmarking case, and a not-so-traditional, but still demanding, case.

The general conclusions one may draw from the results in this chapter is that the eddy viscosity implementation has been done correctly, in addition to the Smagorinsky, WALE, σ and Dynamic Smagorinsky models having been implemented correctly. Some results obtained for the channel flow case, like heightened mean velocity profiles for WALE, σ and Dynamic Smagorinsky, do not, if presented alone, point to the implementations being good. However, these erroneous results are clearly coupled to the wall behaviour of these models, where again the mean plots for ν_T clearly shows that the decaying rate of y^3 is fulfilled for all of them. The Smagorinsky model does for this case return good results overall, in general producing mean velocity profiles that are much closer to the DNS data. Nevertheless, its over contribution of ν_T near the wall is not only noticeable in the mean velocity profiles, but also for $Re_\tau = 950$, where it is clear that the near-wall problems lead to mean velocity data that is altered too much. All the results obtained for this case do in general verify the eddy viscosity implementation, where also the good data obtained for the mean ν_T plots and some selected Reynolds stresses, all point towards the LES models most likely having been correctly implemented.

The FDA case does, in despite of some varying results, help the process of verification and validation, strengthening the conclusions overall. One may from the results obtained here deduce that, for cases containing transitional effects, the Smagorinsky model without damping functions is not a good approach. Mildly speaking. On the other hand, the three other models correctly tackles the transitional process, allowing perturbations to grow, eventually leading to the transition from laminar to turbulent flow, and a breakdown of the jet. The overall conclusion is that all the tested LES models lead to effects that are positive and as expected, in that all the obtained mean profiles are between 50 to 80 percent improved over the UDNS results. The results are a bit inconsistent, as e.g. the Dynamic Smagorinsky model returns data that is worse for the finer meshes. There is still a way to go in terms of mesh independence, model consistency, and similar effects; nevertheless, this is not only an implementational problem, but also, or rather, a problem with the eddy viscosity models themselves.

Chapter 7

Concluding Remarks

In this work the turbulence modelling method of Large Eddy Simulation (LES) has been implemented into *Oasis*, a Computational Fluid Dynamics (CFD) solver for incompressible flows based on the Finite Element method (FEM), developed in-house at the University of Oslo (UiO). The focus when it comes to the LES models has been on so-called eddy viscosity models, which attempt to model the left-out dissipation, originally caused by the subgrid-scale velocities, through a fictional eddy viscosity term. Two types of models have been presented: static models, which include a general constant, and so-called dynamic models, which aim towards computing these constants dynamically as functions of the flow field.

Implementational details for the general eddy viscosity term, and for the Smagorinsky, WALE, σ, Dynamic Smagorinsky, Scale-Dependent Dynamic, and the Kinetic Energy SGS models, were presented in Ch. 4. Starting with the general LES contribution, the residual stress tensor was discretised with FEM, where some subsequent problems were thoroughly discussed, before its implementation into the *Oasis* framework was presented. As for the LES models selected details from the implementations were shown, where discussion regarding certain problems and challenges was done here as well. In general, implementing models that do not require explicit computations of any sort is an easy task, since most expressions may be coded without much effort when FEniCS' Unified Form Language (UFL) is applied. On the other hand, implementation of models that require additional operations as e.g. test filtering, explicit computation of discrete quantities, or additional Partial Differential Equations (PDEs) to be solved, is a much more demanding work, as one then needs to have a bigger focus on general optimization and implementation of good solutions.

Verification and assessment was first performed applying the Method of Manufactured Solutions (MMS), then to test the implementation of the general residual stress tensor, and the Smagorinsky and WALE models. The convergence rate of Δt^2 was correctly recovered for all cases, showing that the general eddy viscosity implementation, and the Smagorinsky and WALE models, most likely have been correctly implemented. In addition, the eddy viscosity returned by the discrete

Smagorinsky and WALE models were compared to the analytical ones, where good results also here were obtained for both implementations. Further, some simple test cases were applied to assess the models, and show some of their shortcomings and limitations. The Dynamic Smagorinsky model and the σ model did as expected tackle both cases well, whereas Smagorinsky failed for simple shear, and WALE failed for solid body rotation. These test cases do, in despite of being extremely simple, both contribute to further verification of the implementations, as the eddy viscosities were correctly computed by all models for both cases.

Further verification was done through the classic case of turbulent channel flow in an $x-z$ periodic channel, where DNS data from MKM [22] and Jimenez [23] was compared to mean velocity measurements obtained for the different LES models. In general, the Smagorinsky model, which produces too much eddy viscosity in the near-wall regions, actually showed good tendencies for all Re_τ, producing mean velocity data that was much closer to that of resolved DNS compared to that of under-resolved DNS. On the other hand, the WALE, σ and the Dynamic Smagorinsky models all failed for this case, returning over estimated mean velocities for all Re_τ. Investigation showed that this has nothing to do with errors in the implementations, but is rather an effect of these three models being able to correctly reproduce the wall decaying behaviour of y^3 for ν_T. Some slightly positive effects were seen for two cases: the Dynamic Smagorinsky model for $Re_\tau = 590$, and the WALE model for a P_2P_1 simulation for $Re_\tau = 180$. For the former one may from the mean plots for ν_T in Fig. 6.13 see that the Dynamic Smagorinsky model has a larger contribution closer to the wall compared to the σ and the WALE model, most likely leading to the more positive results. As for the latter this was a result of the P_2P_1 simulation, and the fact that ν_T was in a lower ordered P_1 space, resulting in a larger eddy viscosity contribution also from the WALE model closer to the wall. A fabricated ν_T, which included selectable wall decaying rate, clearly showed that the net-contribution of eddy viscosity in the wall region is controlling the quality of the simulation in total.

Validation was done through the U.S. Food and Drug Administration (USFDA/FDA)'s computational round robin #1, where simulations for both uniform and non-uniform meshes of two and three million cells were performed. The general results were good, as all LES models managed to produce better mean velocity profiles compared to the results obtained with no LES model activated (between 50 to 80 percent improvement). Concluding remarks are that the Dynamic Smagorinsky model works best for the simulations done here, as its mean percentage improvement is the highest in total, whereas the static models are more problematic. Both the Smagorinsky, WALE, and the σ model clearly suffered from the fact that general constants were applied (also seen through the channel case), where it was clear that small adjustments of the constants would have produced better profiles. A global dynamic σ model was also tested, it returning good values for both C_σ and $\left(C_\sigma\overline{\Delta}\right)^2$. These values were slightly higher compared to the value of $C_\sigma = 1.5$ presented by [33]; however, this does indeed fit well with the fact that $C_\sigma = 1.6$, as used for the simulations in this work, was too conservative. It is expected that the dynamically computed value of $C_{\sigma,dyn} \simeq 1.72$ would have returned better results.

7.1 Limitations

In general, there are little or no limitations in the work done here. Everything from the eddy viscosity formulation, to the LES models, have been implemented in a completely general way, thus rendering the *Oasis* framework able to handle LES for all types of turbulent flows. However, if one looks for limitations, some may be found for the Dynamic Smagorinsky model, where e.g. no "perfect“ solution for the test filtering operation was implemented. A good approach was discovered in the GTHF, but as a result of the originally proposed GTHF function space not being available in FEniCS, a shortcut through the linear P_1 Lagrange elements and their corresponding basis functions was taken. In addition to this, the Lagrangian averaging approach chosen here has been quite problematic, where some shortcuts, like e.g. the avoidance of interpolation to obtain the upstream-located values, have been taken to arrive at a working solution.

As seen through both the channel case and the FDA-case, the LES models that contain a universal constant of any type are problematic. Determining the correct value of these constants is somewhat impossible from a general point of view, since they both vary in time and space, and between experiments. For the industry running cases where DNS or experimental data is unavailable, the need for models that are completely general is extremely high, such that correct and believable data may be obtained. This limitation is more of a general modelling problem than an implementational one; however, it is clear that the Dynamic Smagorinsky model (and other dynamic models) have a huge advantage in that they require little or no user input, in addition to them being completely general procedures.

In addition to this the solver may still be said to be limited to simulations of moderate Reynolds numbers, then as a result of the near wall problems with LES as discussed briefly in section 3.4.1. This is, nonetheless, not a limitation in the implementations done as a part of this work, but more of a LES specific problem.

7.2 Significance

The work done in this thesis has constructed a rigid fundamental basis for further investigation and development of the LES framework with *Oasis*, then in combination with unstructured meshes, complex turbulent flows, flows containing challenging transitional effects, and similar. *Oasis* is now able to handle general LES simulations of any type, it boasting traditional but still successful models as those applying the dynamic procedure of Germano, and more modern differential operator models as e.g. the σ model.

For the industry, and for students, people or research groups who are genuinely interested in doing simulations applying LES, *Oasis* has as a result of the work done here become a good, robust, and efficient tool. It may be applied e.g. in courses at educational institutions where CFD and LES are important topics, as simple LES simulations can be carried out, the effect of the different LES models can be directly analysed and tested, the implemented code may be presented to students, and much more. From an industrial point of view *Oasis* has possibly become a more interesting tool, as it now may be used for LES of complex, incompressible turbulent flows of any type.

7.3 Future Work

Future work will be directed not only towards improving the current implementations, but also towards implementing new and more interesting eddy viscosity models. As showed in Ch. 4, the amount of time spent on optimizations, especially for the Dynamic Smagorinsky model, is quite high. It is, however, most likely room for improvement and optimization of the algorithms, such that the amount of extra time required for LES computations could be even further reduced.

As an example, extending the already implemented static models into their dynamic relatives is an easy task, as most ingredients for both procedures have been implemented already. When it comes to implementation of new models, a good starting point would be static and dynamic mixed models of different types. Both the traditional mixed models presented in section 3.3.7, in addition to more modern dynamic mixed models, which e.g. apply either the WALE model, the σ model, or other eddy viscosity models as purely diffusive parts, are interesting procedures.

As for the explicitly required filtering procedures both the test filters for the dynamic models, and those required as the mesh filters for the mixed models, should be further investigated. These procedures are in general problematic for unstructured meshes, as no well-defined and completely general filtering procedures exist. In addition, the dynamic models all contain the free parameter α, representing the test to mesh filter ratio. In this implementation a value of $\alpha = 2$ has been applied, as this is a widely used solution that has been tested and proven to be good. However, some suitable analysis should be done in terms of the tweaked GTHF applied in this work and α, to further justify the eventual choice of value for this parameter.

The erroneous results obtained with WALE, σ and the Dynamic Smagorinsky model for the channel flow case, should definitely be further investigated. As discussed in the corresponding section, it seems like this is a problem that has been scarcely investigated: similar results exists, but explanations and research done on the topic seems very scarce. In their original papers both Meneveau et al. [36] and Toda et al. [33] received good results for their Lagrangian Dynamic model and static σ model, respectively. Both these models failed here, hence investigation on how to obtain similar results through *Oasis* should be done. As shown in the channel flow section, this problem is directly connected to the wall behaviour of the LES models, where again this wall behaviour most likely is coupled with the wall problems of LES for unresolved turbulent boundary layers. Three routes may be taken to further test this case: (a) sufficiently resolve the boundary layer region, (b) invoke wall functions, or (c) find/develop LES models that are able to tackle this problem directly. Hopefully solution (a) or (b) would lead to better results for the models which in general were problematic. This is also important from a validation point of view, as better, and hopefully good, results should be obtained for this case.

In addition to further analysing already tested experiments, new test cases like decaying isotropic turbulence, a turbulent plane jet, or other traditional benchmarking methods should be investigated. As was done with the FDA case, the solver and the capabilities of the LES models should also be further tested for other non-trivial, non-standard, complex cases, where experimental or DNS data is available.

Bibliography

[1] L. da Vinci (author), J. P. Richter (editor), *The notebooks of Leonardo da Vinci*, Dover Publications (1970), ISBN 978-04-86-22572-2.

[2] O. Reynolds, *On the dynamical theory of incompressible viscous fluids and the determination of the vriterion*, Philosophical Transactions of the Royal Society of London. A, Vol. 186 (1895) 123-64.

[3] J. Smagorinsky, *General circulation experiments with the primitive equations*, American Meteorological Society, Vol. 91, Iss. 3 (1963) 99-164.

[4] S. B. Pope, *Turbulent flows*, 1st ed., Cambridge University Press (2000), ISBN 978-05-21-59886-6.

[5] P. Sagaut, *Large eddy simulation for incompressible flows*, 3rd ed., Springer (2006), ISBN 978-35-40-26344-9.

[6] J. Bø, A. W. Bergersen, K. V.-Sendstad, M. Mortensen, *Implementation, verification and validation of large eddy simulation models in Oasis*, MekIT'15, CIMNE (2015).

[7] Ansys FLUENT.
URL `http://www.ansys.com/Products/Simulation+Technology/\Fluid+Dynamics/Fluid+Dynamics+Products/ANSYS+Fluent`.

[8] Star-CD.
URL `http://www.cd-adapco.com/products/star-cd%C2%AE`.

[9] OpenFOAM - The open source CFD toolbox.
URL `http://www.openfoam.com/`.

[10] M. Y. Hussaini, T. A. Zang, *Spectral methods in fluid dynamics*, Annual Review of Fluid Mechanics, Vol. 19 (1987) 339-67.

[11] B. Fornberg, *A practical guide to pseudospectral methods*, Cambridge University Press, Cambridge (2009), ISBN 978-05-11-62635-7.

[12] R. A. Gingold, J. J. Monaghan, *Smoothed particle hydrodynamics - theory and application to non-spherical stars*, Monthly Notices of the Royal Astronomical Society, Vol. 181 (1977) 375-89.

[13] F. J. Higuera, S. Succi, R. Benzi, *Lattice gas dynamics with enhanced collisions*, Europhysics Letters, Vol. 9, Num. 4 (1989) 345-49.

[14] F. J. Higuera, J. Jimenez, *Boltzmann approach to lattice gas simulations*, Europhysics Letters, Vol. 9, Num. 7 (1989) 663-68.

[15] A. J. Chorin, *The numerical solution of the navier-stokes equations for an incompressible fluid*, Bulletin of the American Mathematical Society, Vol. 73 (1967) 928-31.

[16] J. C. Simo, F. Armero, *Unconditional stability and long-term behavior of transient algorithms for the incompressible navier-stokes and euler equations*, Computer Methods in Applied Mechanics and Engineering, Vol. 111, Iss. 1-2 (1994) 111-54.

[17] A. Logg, K. A. Mardal, G. N. Wells, *Automated solution of differential equations by the finite element method*, Springer (2012), ISBN 978-3-642-23099-8.

[18] Mortensen, M. & Valen-Sendstad, K. *Oasis - a high-level/high-performance open source navier-stokes solver*, Computer Physics Communications, Vol. 188 (2015) 177-88.

[19] The FEniCS Project.
URL `http://fenicsproject.org/`.

[20] Oasis.
URL `https://github.com/mikaem/Oasis`.

[21] Oasis, the LES development branch.
URL `https://github.com/mikaem/Oasis/tree/LES`.

[22] R. D. Moser, J. Kim, N. N. Mansour, *Direct numerical simulation of turbulent channel flow up to $Re_\tau = 590$*, Physics of Fluids, Vol. 11, Iss. 4 (1999) 943-45.

[23] J. C. Del Alamo, J. Jimenez, P. Zandonade, R. D. Moser, *Scaling of the energy spectra of turbulent channels*, Journal of Fluid Mechanics, Vol. 500 (2004) 135-44.

[24] J. V. Boussinesq, *Essai sur la théorie des eaux courantes*, Mémoires présentés par divers savants à l'Académie des Sciences, T23 (1877).

[25] D. K. Lilly, *The representation of small-scale turbulence in numerical simulation experiments*, Proceedings of the IBM Scientific Computing Symposium on Environmental Sciences (1967).

[26] A. Leonard, *Energy cascade in large-eddy simulations of turbulent fluid flows*, Advances in Geophysics, Vol. 18, Part A (1974) 237-48.

[27] J. W. Deardorff, *A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers*, Journal of Fluid Mechanics, Vol. 41, Iss. 2 (1970) 453-80.

[28] J. W. Deardorff, *The use of subgrid transport equations in a three-dimensional model of atmospheric turbulence*, Journal of Fluids Engineering, Vol. 95, Iss. 3 (1973) 429-38.

[29] P. R. Spalart, W.-H. Jou, M. Strelets, S. R. Allmaras, *Comments on the feasibility of LES for wings, and on a hybrid RANS/LES approach*, Proceedings of the First AFOSR International Conference on DNS/LES (1997).

[30] S.-H. Peng, W. Haase, *Advances in hybrid RANS-LES modelling*, Notes on Numerical Fluid Mechanics and Multidisciplinary Design, Vol. 97 (2008), ISBN 978-3-540-77815-8.

[31] E. R. Van Driest, *On turbulent flow near a wall*, Journal of the Aeronautical Sciences, Vol. 23, Iss. 11 (1956) 1007-11.

[32] F. Ducrous, F. Nicoud, *Wall-adapting local eddy viscosity models for simulations in complex geometries*, Flow, Turbulence and Combustion, Vol. 62, Iss. 3 (1999) 183-200.

[33] H. B. Toda, O. Cabrit, G. Balarac, S. Bose, J. Lee, H. Choi, F. Nicoud, *A subgrid-scale model based on singular values for LES in complex geometries*, Center for Turbulence Research, Proceedings of the Summer Program 2010 (2010).

[34] M. Germano, *A dynamic subgrid-scale eddy viscosity model*, Physics of Fluids, Vol. 3, Iss. 7 (1991) 1760-65.

[35] D. K. Lilly, *A proposed modification of the Germano subgrid-scale closure method*, Physics of Fluids, Vol. 4, Iss. 3 (1992) 633.

[36] C. Meneveau, T. S. Lund, W. H. Cabot, *A Lagrangian dynamic subgrid-scale model of turbulence*, Journal of Fluid Mechanics, Vol. 319 (1996) 353-85.

[37] H. Baya Toda, K. Truffin, F. Nicoud, *Is the dynamic procedure appropriate for all SGS models?*, V European Conference on Computational Fluid Dynamics, ECCOMAS CFD (2010).

[38] D. R. Chapman, G. D. Kuhn, *The limiting behaviour of turbulence near a wall*, Journal of Fluid Mechanics, Vol. 170 (1986) 265-92.

[39] E. Bou-Zeid, C. Meneveau, M. Parlange, *A scale-dependent Lagrangian dynamic model for large eddy simulation of complex turbulent flows*, Physics of Fluids, Vol. 17, Iss. 2 (2005) Art. no. 025105.

[40] A. Yoshizawa, K. Horiuti, *A statistically-derived subgrid-scale kinetic energy model for the large-eddy simulation of turbulent flows*, Journal of the Physical Society of Japan, Vol. 54, No. 8 (1985) 2834-39.

[41] S. Ghosal, T. S. Lund, P. Moin, K. Akselvoll, *A dynamic localization model for large-eddy simulation of turbulent flows*, Journal of Fluid Mechanics, Vol. 286 (1995) 229-55.

[42] V. C. Wong, *A proposed statistical-dynamic closure method for the linear or nonlinear subgrid-scale stresses*, Physics of Fluids, Vol. 4, Iss. 5 (1992) 1080-82.

[43] J. Bardina, J. H. Ferziger, W. C. Reynolds, *Improved subgrid scale models for large eddy simulation*, The American Institute of Aeronautics and Astronautics (AIAA), 13th Fluid and Plasmadynamics Conference (1980).

[44] Y. Zang, R. L. Street, J. R. Koseff, *A dynamic mixed subgrid-scale model and its application to turbulent recirculating flows*, Physics of Fluids, Vol. 5, Iss. 12 (1993) 3186-96.

[45] A. W. Vreman, B. Geurts, H. Kuerten, *On the formulation of the dynamic mixed subgrid-scale model*, Physics of Fluids, Vol. 6, Iss. 12 (1994) 4057-59.

[46] G. A. Tabor, M. H. Baba-Ahmadi,*Inlet conditions for large eddy simulation: a review*, Computers & Fluids, Vol. 39, Iss. 4 (2010) 553-67.

[47] U. Piomelli, *Wall-layer models for large-eddy simulations*, Progress in Aerospace Sciences, Vol. 44, Iss. 6 (2008) 437-46.

[48] C. B. da Silva, O. Metais, *On the influence of coherent structures upon interscale interaction in turbulent plane jets*, Journal of Fluid Mechanics, Vol. 473 (2002) 103-45.

[49] N. Park, S. Lee, J. Lee, H. Choi, *A dynamic subgrid-scale eddy viscosity model with a global model coefficient*, Physics of Fluids, Vol. 18, Iss. 12 (2006).

[50] A. W. Vreman, *An eddy viscosity subgrid-scale model for turbulent shear flow: algebraic theory and applications*, Physics of Fluids, Vol. 16, Iss. 10 (2004) 3670-81.

[51] D. You, P. Moin, *A dynamic global-coefficient subgrid-scale eddy viscosity model for large-eddy simulation in complex geometries*, Physics of Fluids, Vol. 19, Iss. 16 (2007) Art. no. 065110.

[52] S. Singh, D. You, *A dynamic global-coefficient mixed subgrid-scale model for large-eddy simulation of turbulent flows*, International Journal of Heat and Fluid Flow, Vol. 42 (2013) 94-104.

[53] J. Lee, H. Choi, N. Park, *Dynamic global model for large eddy simulation of transient flow*, Physics of Fluids, Vol. 22, Iss. 7 (2010) Art. no. 125109.

[54] Cascade Technologies.
URL *http://www.cascadetechnologies.com/.*

[55] Cascade Tecnhologies, blog entry January 1st 2013.
URL *http://www.cascadetechnologies.com/breaking-the-1-million-core-barrier-in-computational-science/.*

[56] U. Piomelli, *Large-eddy simulation: achievements and challenges*, Progress in Aerospace Sciences, Vol. 35, Iss. 4 (1999) 335-62.

[57] S. B. Pope, *Ten questions concerning the large-eddy simulation of turbulent flows*, New Journal of Physics, Vol. 6 (2004) 1-24.

[58] C. Fureby, *Towards the use of large eddy simulation in engineering*, Progress in Aerospace Sciences, Vol. 44, Iss. 6 (2008) 381-96.

[59] R. Bouffanais, *Advances and challenges of applied large-eddy simulation*, Computers & Fluids, Vol. 39, Iss. 5 (2010) 735-38.

[60] Y. Zhiyin, *Large-eddy simulation: past, present and the future*, Chinese Journal of Aeronautics, Vol. 28, Iss. 1 (2015) 11-24.

[61] NumPy - the fundamental package for scientific computing with Python, URL `http://www.numpy.org/`.

[62] J. Volker, *Large eddy simulation of turbulent incompressible flows*, Springer (2004), ISBN 978-35-40-40643-3.

[63] K. Jansen, *A stabilized finite element method for computing turbulence*, Computer Methods in Applied Mechanics and Engineering, Vol. 174, Iss. 3-4 (1999) 299-317.

[64] J. R. Bull, M. D. Piggott, C. C. Pain, *A finite element LES methodology for anisotropic inhomogeneous meshes*, Turbulence, Heat and Mass Transfer, 7th International Symposium (2012).

[65] M. Germano, *Differential filters for the large eddy numerical simulation of turbulent flows*, Physics of Fluids, Vol. 29, Iss. 6 (1986) 1755-66.

[66] SymPy - a Python library for symbolic mathematics, URL `http://www.sympy.org/en/index.html`.

[67] Online data for fully developed turbulent channel flow, provided by Moser, Kim and Mansour, and Jimenes et al., among others. URL `http://turbulence.ices.utexas.edu/data/`.

[68] S. Schmidt, H. M. Blackburn, *Spectral element based dynamic large eddy simulation of turbulent channel flow*, 14th Australasian Fluid Mechanics Conference (2001).

[69] G.-H. Cottet, V. Vasilyev, *Comparison of dynamic Smagorinsky and anisotropic subgrid-scale models*, Center for Turbulence Research, Proceedings of the Summer Program 1998 (1998).

[70] P. Lampitella, *Large eddy simulation for complex industrial flows*, PHD-thesis, Polytechnic University of Milan, Milan, Italy (2014).

[71] U.S. Food and Drug Administration's Computational Round Robin #1. `https://fdacfd.nci.nih.gov/interlab_study_1_nozzle`.

[72] S. F. C. Stewart, *Assessment of CFD performance in simulations of an idealized medical device: results of FDA's first computational interlaboratory study*, Cardiovascular Engineering and Technology, Vol. 3, Iss. 2 (2012) 139-60.

[73] T. Passerini, A. Quaini, T. Villa, A. Veneziani, S. Canic, *Validation of an open source framework for the simulation of blood flow in rigid and deformable vessels*, International Journal for Numerical Methods in Biomedical Engineering, Vol. 29, Iss. 11 (2013) 1192-213.

[74] Y. T. Delorme, K. Anupindi, S. H. Frankel, *Large Eddy Simulation of FDA's idealized medical device*, Cardiovascular Engineering and Technology, Vol. 4, Iss. 4 (2013) 392-407.

[75] S. Bhushan, D. K. Walters, G. W. Burgreen, *Laminar, turbulent, and transitional simulations in benchmark cases with cardiovascular device features*, Cardiovascular Engineering and Technology, Vol. 4, Iss. 4 (2013) 408-426.

[76] P. Hariharan, M. Giarra, V. Reddy, S. W. Day, K. B. Manning, S. Deutsch, S. F. C. Stewart, M. R. Myers, M. R. Berman, G. W. Burgreen, E. W. Paterson, R. A. Malinauskas, *Multilaboratory particle image velocimetry analysis of the FDA benchmark nozzle model to support validation of computational*

fluid dynamics simulations, Journal of Biomechanical Engineering, Vol. 133, Iss. 4 (2011) Art. no. 041002.