NTNU
Norwegian University of
Science and Technology

Application of Nek5000 to dispersion simulations

Magnus Aarskaug Rud

Master of Science in Physics and Mathematics	
Submission date:	January 2016
Supervisor:	Anne Kværnø, MATH
Co-supervisor:	Carl Erik Wasberg, FFI

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

Simulations of turbulent flow and gas dispersion are performed using Nek5000, with and without a subgrid-scale model. The results are compared with data from wind-tunnel experiments and previously performed simulations in CDP. These comparisons show good agreement between the reference data and the results obtained in this thesis, and the simulation time is reported to be approximately five times faster for Nek5000. Two routines have been implemented to improve the work with complex geometries, and a laminar test case is used to confirm the importance of this work and the spectral accuracy of the solver.

Acknowledgements

I would like to thank my Supervisor at FFI, Carl Erik Wasberg for excellent guidance, and great support during the work with this thesis. I have truly enjoyed our many discussions over these past 20 weeks. Big thanks also to my supervisor Anne Kværnø at NTNU for great advices whenever requested.

I would also like to thank the turbulence-group at FFI for interesting discussions and our many lunch conversations, special thanks to Daniel Eriksson, Thor Gjesdal and Hannibal Fossum for providing reference data, numerical expertise and requested geometry.

Contents

Notation

CONVENTION	we let subscript h denote the discretized variables
$\mathbf{u}$	the velocity field
p	pressure
$\mathbf{f}$	source function
ν	kinematic viscosity
U	Freestream velocity / velocity scale
L	Length scale
$Re = UL/\nu$	Reynolds number
d	The dimension of the problem, in this thesis $d = 3$
Ω	Some general domain in $\mathbb{R}^d$
$s_{ij} = 1/2(\nabla\mathbf{u} + \nabla^T\mathbf{u})$	Strain-rate tensor
$\mathcal{S} = \lvert s_{ij}\rvert = \sqrt{2s_{ij}s_{ij}}$	Absolute value of the strain-rate tensor
τ_{ij}	subgrid-scale tensor
$\mathcal{A}$	The bilinear Laplace operator
$\mathcal{B}$	The bilinear divergence operator
$\mathcal{C}$	The trilinear convection operator
A	The stiffness matrix
D	The divergence matrix
C	The convection matrix
M	The mass matrix
H	The Helmholtz matrix
$\lvert\lvert u\rvert\rvert$	the L^2 norm of u
$\lvert\lvert u\rvert\rvert_m$	the H^m norm of u
$\lvert\lvert u\rvert\rvert_{m,k}$	the H^m norm of u on the domain Ω_k

Chapter 1

Introduction

The physics regarding fluids in motion are described mathematically by the Navier-Stokes (N-S) equations. They are a result of the conservation of momentum and mass and are stated in Eq. (2.1). This thesis is restricted to numerical solutions of the incompressible N-S equations. For a complete description of the necessary assumptions and simplifications it is referred to [1]. An important dimensionless quantity is the Reynolds number Re, which describes the ratio between momentum forces and viscous forces. For large Reynolds numbers the flow becomes turbulent and a large range of scales needs to be resolved. A lot of research has been devoted to determine the amount of energy present at the different scales of motion, and the interaction between them. These ideas has led to turbulence modelling which is based on the idea that the effect of the smallest turbulent motions can be modelled, while the larger motions are resolved by the numerical grid. In this thesis both laminar and turbulent flows will be solved numerically, and a physically motivated model for the smallest structures will be compared with a mathematical filter meant to stabilize the flow. In addition to solving the N-S equations the transport of a passive scalar will also be analysed and compared with a set of reference solutions. The software applied in this thesis is Nek5000 which is an implementation of the spectral element method initialized in the 80's. In addition to validate Nek5000 as a software for analyzing gas dispersion the work in this thesis also attempts to make Nek5000 more applicable to cases consisting of more complex geometry.

This thesis is divided in 3 parts. The first part which consists of the two following chapters are devoted to the physical understanding of Eq. (2.1), the solution methods applied and a thorough presentation of the Spectral Element Method. Chapter 4 gives the reader a brief introduction to the functionalities of the solver Nek5000 to motivate some of the implementation performed. The last three chapters describes the work performed by the author, a presentation of the results and the discussion of these.

1.1 A brief overview of the work done for this thesis

Before we end this introduction a brief overview of the work done for this thesis is listed here. It can be divided in three main sections and will be presented properly in Chapter 5 and 6.

- Nek5000 grid generation
  - Project edges onto a higher order arc.
  - General surface projection.
  - Testing of algorithm for different surfaces.
  - Curvature propagation.
  - Changes and adjustments to `mshconvert`.
- Laminar flow around cylinder
  - Generation of geometry and grid.
  - Setup of the problem with necessary input and output.
  - Simulations with Nek5000 with different grid size, polynomial order, time stepping method and other settings.
- Turbulent flow with gas dispersion
  - Grid generation.
  - Setup the problem with necessary input and output.
  - Implementation of the spatial averaging routine for the dynamic Smagorinsky model.
  - Simulations with Nek5000 with different grids, polynomial order,stabilization methods.
- Analytical work with the numerical filter.

Chapter 2

Problem description

This chapter will present the incompressible Navier-Stokes equations and give a description of the physical effects of each term. It will also present the idea behind Large Eddy Simulations and subgrid-scale models, and prepare the mathematical formulations that will be further analysed in Chapter 3.

2.1 The incompressible Navier-Stokes equations

With the assumption of an incompressible flow, the conservation of mass results in a divergence-free restriction on the velocity $\mathbf{u}$ and the incompressible N-S equations on a domain $\Omega \subset \mathbb{R}^d$ can be stated as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re}\Delta \mathbf{u} + \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0, \qquad \text{on } \Omega, \tag{2.1}$$

with appropriate boundary conditions which will be discussed in Chapter 2.1.1.

A widely studied simplification of these equations is the Stokes problem which is obtained by omitting the convective term. This simplification will be used as a starting point for the analysis of the full problem in Chapter 3.1, but first a brief description of each term in Eq. (2.1) and their mathematical influence on the problem will be given.

- $\partial \mathbf{u}/\partial t$ - The time-derivative of the velocity. For a steady state flow this term will be equal zero. The discretization of this term is often based on some implicit scheme.

- $\mathbf{u}\cdot\nabla\mathbf{u}$ - The convective term. Describes the transport due to the flow itself on each of its components. The term is not present in a Stokes problem. The mathematical operator corresponding to this term is non-linear and non-symmetric, and does therefore require an explicit scheme to be solved efficiently.

- ∇p - The pressure gradient. Removal of this term results in a pure advection diffusion problem. In the incompressible N-S equations the pressure is treated as a Lagrange multiplicator, assuring the projection of $\mathbf{u}$ onto a divergence free space.

- Re - The Reynolds number. Defined as UL/ν where U is a velocity scale, L is a length scale and ν is the viscosity of the fluid. Re describes the ratio between momentum forces and viscous forces. Notice that for large Reynolds numbers the unstable non-linear term dominates the transportation, and the flow can be characterized as turbulent.

- $\Delta\mathbf{u}$ - The diffusive term. Describes the natural diffusion of the fluid, and is inversely proportional to the Reynolds number. The corresponding mathematical operator stabilizes the system and it is therefore generally easier to solve the N-S equations for high-viscosity fluids. It should be mentioned that this term is a simplification of the Reynolds stress tensor that can be made under the assumption of incompressibility and constant viscosity. The more general formulation that will be used to replace $\nu\Delta\mathbf{u}$ in Chapter 2.3 is $\partial_j\left[\nu(\partial_j u_i + \partial_i u_j)\right] = 2\,\nabla\cdot(\nu s_{ij})$, the tensor s_{ij} is known as the strain-rate tensor.

- $\mathbf{f}$ - Force term. Describes external forces such as gravity,sources and sinks. Note however that for incompressible flow the gravity term is incorporated in the pressure term, $\nabla p := \nabla p + \rho\mathbf{g}$.

- $\nabla\cdot\mathbf{u}$ - The divergence-free condition is a result of the conservation of mass in an incompressible flow.

For large Reynolds numbers the huge span in length scales requires a fine mesh to solve Eq. (2.1) exactly. Because a fine mesh implies a high computational cost, a Direct Numerical Simulation (DNS) is not feasible for problems of a certain geometrical complexity. With Large Eddy Simulations (LES) the smallest structures are modelled, thus allowing a coarser mesh. In Chapter 2.3 this method and a particular model will be discussed. The Navier-Stokes equations can only be solved if the boundary conditions are given. The boundary conditions are not stated explicitly in Eq. (2.1) because they depend on the physical situation and belong as a specification to each individual case. The next subsection gives a quick overview of the different boundary conditions applied in this thesis.

2.1.1 Boundary conditions

Depending on the kind of flow and the geometry for a particular problem different boundary conditions are applied. In this section $\mathbf{n}$ and $\mathbf{t}$ will denote the normal vector and a tangential vector to the boundary surface. The boundary conditions applied for the cases investigated in this thesis will be given the names I,O,SYM and W for Inflow, Outflow, Symmetry and Wall. Their mathematical formulation and physical interpretation are given as

- I - Inflow, defining the velocity field on the boundary. Mathematically this is equivalent to non-homogeneous Dirichlet conditions.

$$\mathbf{u} = \mathbf{g}(\mathbf{x}, t). \tag{2.2}$$

- O - Outflow, letting the fluid flow “effortlessly” out through the boundary. Formally stated as

$$\frac{1}{Re}\nabla\mathbf{u}\cdot\mathbf{n} - p\mathbf{n} = 0. \tag{2.3}$$

- SYM - Symmetry, denying any flux through the boundary without disturbing the tangential velocity. Convenient to apply in an open channel where the streamwise direction is parallel to the boundary. Mathematically this is described as

$$\mathbf{u}\cdot\mathbf{n} = 0, \qquad (\nabla\mathbf{u}\cdot\mathbf{t})\cdot\mathbf{n} = 0. \tag{2.4}$$

- W - Wall, Representing a physical object. Also known as the no-slip condition which is based on the assumption that the fluid closest to the object moves with the same speed as the object itself. In this thesis all objects and geometries are kept still, so mathematically this is equivalent to homogeneous Dirichlet conditions.

$$\mathbf{u} = 0. \tag{2.5}$$

2.1.2 Weak formulation of N-S

The numerical algorithms applied in this thesis require a weak formulation of Eq. (2.1). Before the weak form is derived a few operators will be defined to simplify the final expression.

$$\begin{aligned}
(\mathbf{u}, \mathbf{v}) &= \int_\Omega \mathbf{u} \cdot \mathbf{v} d\Omega \\
\mathcal{A}(\mathbf{u}, \mathbf{v}) &= \nu(\nabla \mathbf{u}, \nabla \mathbf{v}) \\
\mathcal{B}(\mathbf{u}, p) &= -(\nabla \cdot \mathbf{u}, p) \\
\mathcal{C}(\mathbf{w}; \mathbf{u}, \mathbf{v}) &= (\mathbf{w} \cdot \nabla \mathbf{u}, \mathbf{v})
\end{aligned} \tag{2.6}$$

A weak formulation is obtained by multiplying with the test functions $\mathbf{v}$ and q in Eq. (2.1) and integrating over the entire domain,

$$\begin{aligned}
\int_\Omega \frac{\partial \mathbf{u}}{\partial t} \cdot \mathbf{v} d\Omega + \int_\Omega (\mathbf{u} \cdot \nabla)\mathbf{u} \cdot \mathbf{v} d\Omega &= -\int_\Omega \nabla p \cdot \mathbf{v} d\Omega + \nu \int_\Omega \Delta \mathbf{u} \cdot \mathbf{v} d\Omega + \int_\Omega \mathbf{f} \cdot \mathbf{v} d\Omega \\
\int_\Omega (\nabla \cdot \mathbf{u}) q d\Omega &= 0.
\end{aligned} \tag{2.7}$$

Introducing the compact inner product notation and applying the divergence theorem on the right hand side of the first equation yields

$$\begin{aligned}
(\frac{\partial \mathbf{u}}{\partial t}, \mathbf{v}) + (\mathbf{u} \cdot \nabla \mathbf{u}, \mathbf{v}) &= (\nabla \cdot \mathbf{v}, p) - \nu(\nabla \mathbf{u}, \nabla \mathbf{v}) + (\mathbf{f}, \mathbf{v}) \\
(\nabla \cdot \mathbf{u}, q) &= 0.
\end{aligned} \tag{2.8}$$

The contributions from the boundary as a result of the application of the divergence theorem are included in the force term. The choice of search spaces for the velocity and pressure will be justified in Chapter 3, but will be stated here to present the weak formulation. Let $V \subset H^1(\Omega)^3$ and $Q \subset L^2(\Omega)$, by using the notation introduced in Eq. (2.6) the weak formulation of the incompressible N-S equations can be stated as:

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

$$\begin{aligned}(\frac{\partial \mathbf{u}}{\partial t}, \mathbf{v}) + \mathcal{C}(\mathbf{u}; \mathbf{u}, \mathbf{v}) &= -\mathcal{B}(\mathbf{v}, p) - \mathcal{A}(\mathbf{u}, \mathbf{v}) + (\mathbf{f}, \mathbf{v}), \\ \mathcal{B}(\mathbf{u}, q) &= 0,\end{aligned} \tag{2.9}$$

$\forall\ (\mathbf{v}, q) \in V \times Q$.

To solve this equation numerically, everything has to be discretized and expressed using a set of basis functions. These basis functions can then describe any test functions $(\mathbf{v}_h, q_h)$, and the solution $(\mathbf{u}, p)$ can be approximated by a discretized solution $(\mathbf{u}_h, p_h)$. This also results in discrete bilinear operators that can be represented by matrices. The discretized system of equations can be stated as

$$M\frac{\partial \mathbf{u}_h}{\partial t} + C(\mathbf{u}_h)\mathbf{u}_h = D^T p_h - A\mathbf{u}_h + M\mathbf{f}_h, \tag{2.10}$$

$$D\mathbf{u}_h = 0. \tag{2.11}$$

2.2 The passive scalar equation

The N-S equations explain how a fluid will behave, and solving these provides a pressure-velocity field on the domain of interest. In many practical situations one is however interested in how a scalar such as heat or a neutral gas will move in this flow. The equation corresponding to the motion of a scalar ϕ in a velocity field $\mathbf{u}$ will be referred to as the passive scalar (PS) equation and is stated as

$$\rho c_p(\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla\phi) = \nabla \cdot (k\nabla\phi) + q_{vol}. \tag{2.12}$$

The constants k and ρc_p are interpreted depending on the scalar transported. For dispersion of neutral gas with ϕ as the volume concentration of the gas they resemble the viscosity and mass flux. The last term on the right hand side q_{vol} is the source term and is not included in this thesis since all gas enters the control volume as a boundary condition.

The passive scalar equation is solved in this thesis as a Helmholtz problem, by applying an explicit scheme on the convection term and an implicit scheme on the diffusion term.

This is similar to the discretization performed on the momentum equation which will be discussed in detail in Chapter 3.

2.3 Resolving the turbulent term using LES

When DNS is not feasible due to a high Reynolds number, LES is one of the most powerful tools for simulating turbulent flows. The idea is based on the fact that the small turbulent structures behave homogeneously and are therefore easy to model. This way the larger structures driven by geometry, inflow conditions and external forces can be simulated using a coarser grid while the effect of the small structures is modelled. LES will be introduced here in a mathematical fashion, starting by defining a filter function, although in many practical cases the filter function is not well defined. The reason for this is that the grid itself is often considered a filter, with the grid size as the filter width. As pointed out by Carati et al. [2] the filter is in this case nothing else but numerical discretization error.

2.3.1 Filter

The idea behind LES starts with defining a filter, which separates the modelled structures from the resolved ones. A filter in its general mathematical form introduced by Leonard [3] is given as

$$U^r(\mathbf{x},t) = \int_\Omega G_r(\mathbf{r},\mathbf{x})U(\mathbf{x}-\mathbf{r},t)d\mathbf{r}. \tag{2.13}$$

The filter is assumed to commute with the differential operators ∇, Δ and $\partial/\partial t$. By applying the filter on the N-S equations and making the given assumptions the filtered N-S equations can be stated as

$$\begin{aligned} \frac{\partial \mathbf{u}^r}{\partial t} + \mathbf{u}^r \cdot \nabla \mathbf{u}^r &= -\nabla p^r + \nu \Delta \mathbf{u}^r + \mathbf{f}^r - \nabla \cdot \tau, \\ \nabla \cdot \mathbf{u}^r &= 0, \end{aligned} \tag{2.14}$$

where τ in this case denotes the subgrid-scale (SGS) stress given as

$$\tau_{ij}(u_i, uj) = (u_i u_j)^r - u_i^r u_j^r. \tag{2.15}$$

This tensor is a consequence of applying the filter on the non-linear advection term, and it is this tensor that is modelled by a subgrid-scale model. See [4] for a full derivation of the application of a filter on the momentum equation.

2.3.2 Dynamic Smagorinsky-Lilly SGS model

The problem is now reduced to modelling the tensor τ_{ij}, and one of the most common SGS models is the dynamic Smagorinsky-Lilly model which is the one applied in this thesis. The initial progress of this model was made by Smagorinsky in 1963, and by Lilly in 1967 [5] who suggested the following model for the SGS tensor

$$\begin{aligned}
\tau_{ij} &= -2C_s l^2 \mathcal{S}^r s^r_{ij}, \\
s^r_{ij} &= \frac{1}{2}\left(\frac{\partial u^r_i}{\partial x_j} + \frac{\partial u^r_j}{\partial x_i}\right), \\
\mathcal{S}^r &= \sqrt{2 s^r_{ij} s^r_{ij}},
\end{aligned} \tag{2.16}$$

where l denotes the filter width, which for this thesis is equivalent to the grid size. The resolved strain rate s^r_{ij} can be calculated from the filtered velocity gradients, and the problem is now reduced to determining the constant C_s. There were several attempts to determine this constant for the entire domain, but in lack of a general constant applicable to all flow situations a dynamic constant C_d was presented to replaces $C_s l^2$ in Eq. (2.16). C_d is called the dynamic Smagorinsky constant, and is both time and space dependent. The general idea is that C_d is unaffected by the filter and from this assumption a computable expression is developed for the dynamical constant.

Let a, b denote two distinct filters with corresponding filter widths l_a, l_b. Throughout this thesis l_a will be the grid size. The second filter width l_b is in this thesis based the same elemental grid, but with a number of nodes $N_b = N_a - 2$. Hence the relation l_b/l_a, which is the only parameter needed for the simulation is estimated as $\lambda = (N_a - 1)/(N_b - 1)$. Filter b is often referred to as the test filter and is only included to provide an estimation of the dynamic Smagorinsky constant. Remember that a is the filter corresponding too the numerical discretization, while filter b is obtained by applying the filter described in Chapter 3.4.2 with $\alpha_i = 1$ for the highest modes. Let τ_{ij} and T_{ij} denote the stresses based on single- and double filtering operations on the N-S equations

$$\begin{aligned} \tau_{ij} &= (u_i u_j)^a - u_i^a u_j^a, \\ T_{ij} &= ((u_i u_j)^a)^b - (u_i^a)^b (u_j^a)^b. \end{aligned} \tag{2.17}$$

Applying the b filter on the first tensor in Eq. (2.17) allows us to define a new tensor L_{ij} that depends only on the a-filtered variables. The following relation is known as the Germano identity and was first introduced in 1991 [6],

$$L_{ij} = T_{ij} - (\tau_{ij})^b = (u_i^a u_j^a)^b - (u_i^a)^b (u_j^a)^b. \tag{2.18}$$

This tensor now depends on the a-filtered solution and not the resolved one, hence the identity in Eq. (2.18) provides a computable expression for L_{ij}.

Substituting the stress-tensors in Eq. (2.18) with their corresponding expression from Eq. (2.16) and assuming a dynamic constant unaffected by the filter one obtains an approximation for L_{ij} which is also computable,

$$\begin{aligned} L_{ij} &\approx 2C_s l_b^2 \mathcal{S}^{ab} s_{ij}^{ab} - 2(C_s l_a^2 \mathcal{S}^a s_{ij}^a)^b \\ &\approx 2C_s l_a^2 [\lambda^2 \mathcal{S}^{ab} s_{ij}^{ab} - (\mathcal{S}^a s_{ij}^a)^b] \\ &= 2C_d M_{ij}. \end{aligned} \tag{2.19}$$

$$M_{ij} = \lambda^2 \mathcal{S}^{ab} s_{ij}^{ab} - (\mathcal{S}^a s_{ij}^a)^b \tag{2.20}$$

$$C_d = C_s l_a^2 \tag{2.21}$$

$$\lambda = l_b / l_a \tag{2.22}$$

Minimizing the mean-square error between the exact L_{ij} as expressed in Eq. (2.18) and the Boussinesq-based approximation in Eq. (2.19) yields the best approximation for the dynamic Smagorinsky constant

$$C_d = \frac{c_{den}}{c_{num}} = \frac{M_{ij} L_{ij}}{2 M_{kl} M_{kl}}. \tag{2.23}$$

Note that the double indices imply summing. This expression is however not a stable option and to deal with this most implementations apply some sort of mean or smoothing in time and/or space when calculating the constant. In this thesis the smoothing is done in both time and space for the denominator and the numerator in Eq. (2.23). The spatial averaging routine is explained in Chapter 5.3 while the time average used is given as

$$\langle c_{den} \rangle^{n+1} = \epsilon c_{den}^{n+1} + (1-\epsilon)\langle c_{den} \rangle^{n}, \tag{2.24}$$

with $\epsilon = 0.001$. Another property of this expression is that the constant C_d is independent of the filter width, the only necessary variable is the coefficient $\lambda = l_b/l_a$. The assumption made in this model is that turbulence behaves as diffusion, similar to the kinematic viscosity a turbulent viscosity ν_t is defined which for this case is given as $\nu_t = C_d \mathcal{S}^a$.

Let us end this section by stating the filtered N-S equations with the LES using the dynamical Smagorinsky subgrid scale model, and remember that the diffusive term is written in general terms as $2\nabla \cdot (\nu s_{ij})$.

$$\begin{aligned} \frac{\partial \mathbf{u}^a}{\partial t} + \mathbf{u}^a \cdot \nabla \mathbf{u}^a &= -\nabla p^a + \mathbf{f}^a + 2\nabla \cdot (\nu + \nu_t) s_{ij}^a \\ \nabla \cdot \mathbf{u}^a &= 0. \end{aligned} \tag{2.25}$$

Notice that if v_t is a constant in the entire domain this equation would be equivalent to the one for a fluid with viscosity $\nu' = \nu + \nu_t$. The idea is that ν_t will be larger when the subgrid structures are significant and closer to zero when the flow is laminar. This is just one of many types of models that are based on an analogy between subgrid-scale stress and physical stress.

Chapter 3

Numerical algorithms

In the previous chapter the N-S equations were presented and reformulated in several ways without any details on how to actually solve the equations. This chapter aims to give a more detailed description of the solution methods applied. The choice of algorithms and solution spaces requires a more thorough analysis which will be performed on the steady Stokes problem. The time discretization applied will be discussed in Chapter 3.5.

3.1 Numerical concepts on the Stokes problem

The steady Stokes problem does not include the convection term or the time derivative but the highest order terms are all present and is therefore a valid problem to perform this necessary analysis [7]. The steady Stokes problem with homogeneous boundary conditions is given as

$$\begin{gathered} -\mu\Delta\mathbf{u} + \nabla p = \mathbf{f}, \qquad \nabla\cdot\mathbf{u} = 0, \qquad \text{in } \Omega, \\ \mathbf{u} = \mathbf{0} \text{ on } \partial\Omega. \end{gathered} \tag{3.1}$$

Applying the weak formulation to the Stokes problem implies a minimum requirement on the spaces for $\mathbf{u}$ and p, and their test functions. These spaces will be defined as

$$\begin{aligned} H_0^1(\Omega)^3 &= \left\{\mathbf{v} \in H^1(\Omega)^3 \mid \mathbf{v} = \mathbf{0} \text{ on } \partial\Omega\right\}, \\ L_0^2(\Omega) &= \left\{q \in L^2(\Omega) \mid \int_\Omega q\,dx = 0\right\}. \end{aligned} \tag{3.2}$$

The formulation can easily be extended to include inhomogeneous Dirichlet conditions on $\mathbf{u}$ by defining a lifting function as described in [8]. Note also that the pressure is only present through its gradient and is therefore not uniquely defined unless the extra constraint on the mean is defined, hence the 0 in L_0^2. The weak form can now be stated as

Find $(\mathbf{u}, p) \in H_0^1(\Omega)^3 \times L_0^2(\Omega)$ such that

$$\begin{aligned} \mathcal{B}(\mathbf{v}, p) + \mathcal{A}(\mathbf{u}, \mathbf{v}) &= (\mathbf{f}, \mathbf{v}), \\ \mathcal{B}(\mathbf{u}, q) &= 0. \end{aligned} \tag{3.3}$$

$\forall\ (\mathbf{v}, q) \in H_0^1(\Omega)^3 \times L_0^2(\Omega)$.

The numerical solution of this problem requires a discrete formulation of the weak form, with $(\mathbf{u}_h, p_h) \in V \times Q$ as the discrete solution. The discrete spaces V, Q are subspaces of $H_0^1(\Omega)^3, L_0^2(\Omega)$ equipped with the discrete $H^1(\Omega)^3$- and $L^2(\Omega)$-norm denoted $||\cdot||_V$ and $||\cdot||_Q$. For the discrete weak form to be well-posed it has to meet the requirements stated by the inf-sup condition. This condition is known from the study of saddle-point problems, and is often referred to as the Babuska-Brezzi condition due to their important results in [9] and [10]. The condition can be stated as

$$\inf_{q \in Q} \sup_{\mathbf{v} \in V} \frac{\mathcal{B}(\mathbf{v}, q)}{||\mathbf{v}||_V ||q||_Q} \geq b, \tag{3.4}$$

for some $b > 0$. Fulfilling this condition often implies a staggered grid, such that the pressure and the velocity are evaluated at different points. For a Spectral Element formulation of this problem a valid choice of subspaces (V, Q) is $\left(\left[P_N \cap H_0^1\right]^3, P_{N-2} \cap L_0^2\right)$. This will be referred to as the $P_N P_{N-2}$ formulation where P_N denotes the space of polynomials up to degree N. It was however proved by Guermond in [11] that for a particular formulation of the Stokes problem the approximation spaces do not need to fulfill the inf-sup condition. The numerical method corresponding to this formulation will for the sake of convenience use the same polynomial degree for pressure and velocity. The subspaces (V, Q) is therefore chosen as $\left(\left[P_N \cap H_0^1\right]^3, P_N \cap L_0^2\right)$, and will be referred to as the $P_N P_N$ formulation.

Before the analysis of the N-S equations can be taken any further the theory behind the Spectral Element Method will be presented in the following subsections.

3.2 Finite element method

Finite element method (FEM) is one of the most widely used numerical methods applied on problems within construction, flow simulation and many other areas. It offers a precise mathematical foundation and due to the local support of the basis functions it guaranties a sparse system. The decomposition of the geometrical domain into a finite amount of elements, makes it possible to create general algorithms applicable to all kinds of geometries. For the full mathematical foundation of FEM it will be referred to [8], but some of the key properties will be stated here to provide a thorough understanding of the spectral element method (SEM). Throughout this section p denotes the polynomial degree of the basis functions, h represents the average grid size, E is the total number of elements and d is the number of dimensions.

FEM provides an algorithm for solving any well-posed boundary value problem (BVP). The mathematical formulation is obtained by first finding the Galerkin formulation with a corresponding search space X and then choosing a discrete subspace $X_h^p \subset X$ spanned by the finite element basis functions $\{\phi_i^p\}$. The key property of the basis functions is that they only have local support in a small part of the domain. This is what gives rise to the resulting sparse linear system. By increasing the polynomial order, the number of grid points used to define the polynomial will need to increase as well. This implies either reducing the distance between the grid points or increasing the support of each basis function. Both approaches will reduce the sparsity of the final matrix. Another important aspect of FEM is the treatment of the domain Ω, on which a triangulation $\{\mathcal{T}_h\}$ is defined such that the original domain is divided into elements. By defining a reference element and a general mapping function, all the local contributions can be calculated by a generalized quadrature rule before being added to the global system of equations. This is a process tailored for parallelization, and can be generalized for a wide range of problems.

FEM is called a projection method since the solution $u_h \in X_h^p$ is a projection of the actual solution u of the BVP onto the discrete space X_h^p. Provided that the initial BVP is well-posed there exists two constants $M, \alpha > 0$ known as the bounded and coercivity constant such that the error of the solution can be reduced to a pure projection error.

The result is known as Cea's lemma,

$$||u - u_h||_X \leq \frac{M}{\alpha} \min_{v_h \in X_h^p} ||u - v_h||_X, \tag{3.5}$$

the solution u_h provided by the Galerkin method is known as the orthogonal projection of u onto X_h^p.

Before this section ends it is important to understand the two ways to increase accuracy and the effects these two ways have on the algorithm. Assume the solution of the BVP to be infinitely smooth and the domain be sufficiently regular. This yields an error $e = Ch^p$, C being some positive constant. Factors such as geometric complexity, condition number, non-linear operators and the continuity of the solution will all provide slightly more complicated error estimates. However for a simpler BVP such as the Poisson problem on the unit square, the error estimate is valid. A h-refinement will lead to an algebraic convergence of order p, while the sparsity of the system is conserved and the total algorithm does not change in any other way than increasing the number of elements. Keeping h constant and increasing p will provide exponential decay of the error, but the sparsity will be reduced and all integrals solved will require quadrature rules of higher order. A formal statement and numerical validation of the error estimate can be found in [7] chapter 2.6.

To sum up the discussion above a general error estimate from [8] is stated as a theorem,

Theorem 3.1. *Let $\Omega = [-1, 1]$ and $\{\Omega_k\}$ be the non-overlapping elements with a corresponding element size h_k. If $u_{|\Omega_k} \in H^\sigma(\Omega_k)$, then the following error estimate will hold for any $\sigma > p \geq 1$*

$$\inf_{u_h \in X_h^p} ||u - u_h||_1 \leq C \left(\sum_k h_k^{2p} |u|^2_{p+1,k} \right)^{1/2}. \tag{3.6}$$

3.3 Spectral methods

Spectral methods (SM) share a some of the mathematical ideas as FEM, but are not as widely used in real life problems. There are many ways to apply SM, and in this thesis only the Galerkin version with numerical integration (sometimes referred to as G-NI) will be considered and will be referred to only as SM. For a full introduction to SM and its applications to BVP see [12]. SM can be reduced to an interpolation problem such as FEM, and are interesting from a theoretical point of view due to its spectral convergence rate which allows you to obtain solutions of extremely high accuracy. The most important drawback of SM are the difficulties with applications to complex geometries. The Fourier basis gives rise to a very efficient method, but is restricted to periodic boundary conditions. Using the GLL-points and corresponding Lagrange polynomials allow the system of equations surging from a BVP can be constructed in an elegant way, but it is rarely sparse and often result in expensive calculations.

Applying SM on a BVP in one dimension requires a set of basis functions $\{\psi_i\}_N$ defined on the whole domain Ω. The discrete space $X_h(\Omega)$ spanned by the basis functions involves all polynomials up to degree N. A function u is projected onto X_h by the relation

$$u_h(x) = \sum_{i=0}^{N} a_i \psi_i(x). \tag{3.7}$$

Where the coefficients a_i are called the expansion coefficients. The basis and the belonging coefficients can be chosen in many different ways, in this thesis functions ψ_i will be the Lagrange polynomials based on the Gauss-Lobatto-Legendre (GLL) nodes. The reason for choosing these nodes is because it enables us to apply the Gauss-Lobatto quadrature rule. This is one of several existing Gauss-quadratures, and the only one allowing fixed endpoints which is the case for this thesis. For more detailed information on GL-quadrature and other quadrature rules it is referred to [13]. The GLL-nodes $\{\xi_i\}_{N+1}$ are given as the solutions of the equation

$$(1 - \xi^2) L_N'(\xi) = 0. \tag{3.8}$$

L_N being the Legendre polynomial of degree N, defined from the Sturm-Liouville problem

$$\frac{d}{dx}\left[(1 - x^2)\frac{d}{dx} L_n(x)\right] + n(n+1) L_n(x) = 0. \tag{3.9}$$

With equations Eq. (3.8) and Eq. (3.9) the local spectral basis functions ψ_j can be stated as

$$\psi_j(x) = \prod_{i \neq j}^{N} \frac{x - \xi_i}{\xi_j - \xi_i}. \tag{3.10}$$

$\{\xi_i\}$ being the solutions to Eq. (3.8). Note that $\psi_j(\xi_i) = \delta_{ij}$. The expansion coefficients in Eq. (3.7) are then chosen as $a_i = u_i := u(\xi_i)$.

This definition of the expansion coefficients is convenient since the actual value of the function in any point can just be read directly from the coefficients without having to sum all the contributions from the different polynomials. Creating a basis for 2 and 3 dimensions is done simply by taking the tensor product of the basis functions in each direction. To keep track of indices in this section $i, j, k = 1, \cdots, N$ is used for the Lagrange polynomials in one direction while $m, l, n = 1, \cdots, N^d$ will be used for the tensor product of the Lagrange polynomials spanning an entire element. For this particular example $l = i + N(j-1) + N^2(k-1)$, such that the three dimensional basis functions Ψ_l is given as

$$\Psi_l(\mathbf{x}) = \psi_i(x)\psi_j(y)\psi_k(z). \tag{3.11}$$

This expansion to multiple dimensions preserves the $\Psi_l(\boldsymbol{\xi}_m) = \delta_{lm}$. To clarify some of the concepts the SM approach will be applied on the Helmholtz equation

$$-\Delta u + \lambda u = f \quad \text{in } \Omega, \tag{3.12}$$

$$u = 0 \quad \text{on } \partial\Omega. \tag{3.13}$$

Ω will for this example be defined as the unit square $[-1, 1]^2$. Let us start by defining the space $V = H_0^1(\Omega)$ and assuming $f \in L^2(\Omega)$. The weak formulation after applying the divergence theorem can now be stated.

Find $u \in V$ st.

$$\int_\Omega \nabla u \cdot \nabla v d\Omega + \lambda \int_\Omega uv d\Omega = \int_\Omega fv d\Omega \qquad \forall v \in V \tag{3.14}$$

To solve this using SM the discrete space $V_h \subset V$ is defined as span$\{\Psi_l\}$ following the preceding definitions the discrete weak formulation is stated as

Find $u_h = \sum_l u_l \Psi_l \in V_h$ st.

$$\sum_l \left(u_l \int_\Omega \nabla \Psi_l \cdot \nabla \Psi_m d\Omega + u_l \lambda \int_\Omega \Psi_l \Psi_m d\Omega \right) = \int_\Omega f \Psi_m d\Omega \qquad \forall \Psi_m \in V_h. \tag{3.15}$$

The following step of this particular spectral method is evaluating the integrals by using the GLL-quadrature rule, the resulting system of equations is then given as

$$\sum_l \left(u_l \sum_n \rho_n \nabla \Psi_l(\boldsymbol{\xi}_n) \cdot \nabla \Psi_m(\boldsymbol{\xi}_n) + u_l \lambda \sum_n \rho_n \Psi_l(\boldsymbol{\xi}_n) \Psi_m(\boldsymbol{\xi}_n) \right) \tag{3.16}$$

$$= \sum_n \rho_n f \Psi_m(\boldsymbol{\xi}_n), \qquad \text{for } m = 1, \cdots, N^2, \tag{3.17}$$

where ρ_n is the quadrature weight for the nth node, and $\boldsymbol{\xi}_n$ is the vector containing the coordinates to the nth node. Note that all the indices $l, m, n = 1, \cdots, N^2$. This can be written in a compact matrix form as

$$(A + \lambda M) u_h = \tilde{f}, \tag{3.18}$$

where the elements in the matrices and vectors are given as

$$\begin{aligned} A_{lm} &= \sum_n \rho_n \nabla \Psi_l(\boldsymbol{\xi}_n) \cdot \nabla \Psi_m(\boldsymbol{\xi}_n), \\ M_{lm} &= \sum_n \rho_n \Psi_l(\boldsymbol{\xi}_n) \Psi_m(\boldsymbol{\xi}_n) = \rho_l \delta_{lm}, \\ (u_h)_l &= u(\boldsymbol{\xi}_l), \\ \tilde{f}_m &= \sum_n \rho_n f(\boldsymbol{\xi}_n) \Psi_m(\boldsymbol{\xi}_n) = \rho_m f(\boldsymbol{\xi}_j). \end{aligned} \tag{3.19}$$

From these equations it is clear that the mass matrix M is diagonal and the right hand side vector $\tilde{f}$ is easily calculated, while the stiffness matrix A is symmetric but full.

The method outlined in this section is similarly to FEM also a projection method, but by applying a different set of basis functions the projection error is different as well. This theorem and more information about spectral methods and their properties can be found in [14].

Theorem 3.2. *Let* $\Omega[-1,1]$ *and* $u \in H^\sigma(\Omega)$. *The projection of* u *onto* P_N *for any* $\sigma \geq 1$ *is given as*

$$\inf_{v_h \in P_N} ||u - v_h||_1 \leq CN^{1-\sigma}||u||_\sigma. \tag{3.20}$$

3.4 Spectral element method

In the early 1980's the idea to combine FEM and SM came along to obtain the flexibility and sparse properties of FEM combined with the spectral convergence rate provided by SM. The result was the Spectral element method (SEM). Several formulations were investigated mainly by Patera and Maday in the papers [15], [16], [17] with important contributions from Fischer, Rønquist and several more. It is important to understand that when solving the N-S equations the efficiency of the solution method is crucial. The algorithm has to be parallelizable and the development of the super-computers and computational clusters has played an important role in deciding which variants of SEM is applied today. The idea is to divide the domain of the BVP into elements as in FEM and then use spectral basis functions of higher degree with support only within one single element.

In the previous subsection the power of spectral methods was illustrated on the unit square in two dimensions. But the limitations when it comes to more complex geometry rapidly affects the spectral convergence rate. Let $\hat{\Omega}$ be the reference element $[-1,1]^d$, the standard procedure when working on a deformed geometry Ω with SM is to first create a map $\mathcal{F} : \hat{\Omega} \to \Omega$. An example of this map is the Gordon-Hall procedure described in Chapter 3.4.4. The Jacobian is then given as the transposed tensor derivative of $\mathcal{F}$, which in two dimension is written as

$$\mathbf{J} = (\nabla\mathcal{F})^T = \begin{bmatrix} \frac{\partial \mathcal{F}_1}{\partial x} & \frac{\partial \mathcal{F}_1}{\partial y} \\ \frac{\partial \mathcal{F}_2}{\partial x} & \frac{\partial \mathcal{F}_2}{\partial y} \end{bmatrix}, \qquad J = \det(\mathbf{J}). \tag{3.21}$$

This allows us to transform both derivatives and integrals to the reference domain, let $\boldsymbol{\xi} = [\xi, \eta]^T$ denote the axis in the reference domain corresponding to $\mathbf{x} = [x, y]^T$ in the deformed domain. The transformation is performed according to the following identities

$$\begin{aligned} d\mathbf{x} &= \mathbf{J} d\boldsymbol{\xi} \\ \int_\Omega f(\mathbf{x}) d\mathbf{x} &= \int_{\hat{\Omega}} \hat{f} J d\boldsymbol{\xi} \\ \nabla u &= \mathbf{J}^{-T} \hat{\nabla} \hat{u}. \end{aligned} \tag{3.22}$$

Here $\hat{u}, \hat{f}$ are obtain by simply substituting $\mathbf{x}$ with $\mathcal{F}(\boldsymbol{\xi})$ and $\hat{\nabla}$ is the partial differential operator wrt. $\boldsymbol{\xi}$. The important thing to notice here is that whenever an integral is solved and a derivative is introduced the Jacobian appears in the equation. When applying the GLL-quadrature to solve the integrals, equality is guaranteed if and only if the function integrated is of polynomial degree $2n-1$ or less, and the error gets bigger with increasing polynomial degree. A higher order Jacobian could imply a large error in the quadrature.

Although the whole domain Ω is deformed, the deformation of each element $\{\Omega_k\}$ is normally a lot less crucial. This gives SEM a huge advantage and allows it to obtain accurate results even in complicated domains.

Let us again consider the Helmholtz problem Eq. (3.13), but this time on a more general domain Ω. The set of elements $\{\Omega_k\}$ is defined such that $\Omega_i \bigcap \Omega_j$ is either empty, a vertex or a line and $\Omega = \bigcup_{k=1}^{K} \Omega_k$. By applying SEM to Eq. (3.13) the corresponding weak formulation can be stated.

For all elements Ω_k Find $u_{h,k} \in X_k^N$ such that

$$\int_{\Omega_k} \nabla u_{h,k} \cdot \nabla v_{h,k} d\Omega + \lambda \int_{\Omega_k} u_{h,k} v_{h,k} d\Omega = \int_{\Omega_k} f v_{h,k} d\Omega \qquad \forall v_{h,k} \in X_k^N. \tag{3.23}$$

Where $X_k^N = H_0^1(\Omega_k) \bigcap P_N(\Omega_k)$. The same discretization procedure as performed for the pure spectral case is now done for each of the subdomains Ω_k,

$$\sum_i \left(u_i \int_{\Omega_k} \nabla \psi_i \cdot \nabla \psi_j d\Omega + u_i \lambda \int_{\Omega_k} \psi_i \psi_j d\Omega \right) = \int_{\Omega_k} f \psi_j d\Omega \qquad \forall \psi_j \in V_h. \tag{3.24}$$

Since the elements can be deformed a Gordon-Hall map is constructed to map the coordinates to the reference element $\hat{\Omega} = [-1,1]^d$. Applying the identities from Eq.

(3.22) to Eq. (3.24) yields

$$\begin{aligned}\sum_i \left(u_i \int_{\hat{\Omega}_k} (\mathbf{J}_k^{-T}\hat{\nabla}\hat{\psi}_i)^T(\mathbf{J}_k^{-T}\hat{\nabla}\hat{\psi}_j) J_k d\hat{\Omega} + u_i\lambda \int_{\hat{\Omega}_k} \hat{\psi}_i\hat{\psi}_j J_k d\hat{\Omega} \right) &= \int_{\hat{\Omega}_k} \hat{f}\psi_j J_k d\hat{\Omega} \qquad \forall \psi_j \in V_h. \\ \sum_i \left(u_i \int_{\hat{\Omega}_k} \hat{\nabla}^T\hat{\psi}_i \mathbf{J}_k^{-1}\mathbf{J}_k^{-T}\hat{\nabla}\hat{\psi}_j J_k d\hat{\Omega} + u_i\lambda \int_{\hat{\Omega}_k} \hat{\psi}_i\hat{\psi}_j J_k d\hat{\Omega} \right) &= \int_{\hat{\Omega}_k} \hat{f}\psi_j J_k d\hat{\Omega} \qquad \forall \psi_j \in V_h.\end{aligned} \tag{3.25}$$

Notice how the integrals depend on the local Jacobian $\mathbf{J}_k$ and its determinant J_k. The local matrices A_k, M_k and the loading vector f_k are gathered from each element. Equivalently as for FEM the global matrices has to be assembled from all the local matrices corresponding to each subdomain. This procedure is general and if the elements are chosen to be sufficiently small it can be performed on almost any deformed domain as opposed to SM.

3.4.1 Convergence properties

This subsection will present an error estimate that is comparable to Theorem 3.1 and 3.2. The Spectral Element Method can as mentioned earlier be regarded as a projection method similar to FEM and SM. It was proved in [18] that spectral convergence can be achieved only by requiring the solution u to be sufficiently regular within each element. This as opposed to pure spectral methods which requires regularity in the entire domain is an advantage when working with turbulent flows in complex domains. For the sake of clarity the relevant spaces for the following error estimates are repeated here, $X_k^N = H_0^1(\Omega_k) \bigcap P_N(\Omega_k)$, and $P_{N,k} = \{v \in L^2(\Omega), v_{|\Omega_k} \in P_N(\Omega_k)\}$.

Theorem 3.3. *Let $\Omega = [-1, 1]$ and $\{\Omega_k\}$ be the set of non-overlapping elements. If $u_{|\Omega_k} \in H^\sigma(\Omega_k)\ \forall\ k$, the following will hold for any $\sigma \geq 1$*

$$\inf_{v_h \in X_k^N} ||u - v_h||_1 \leq CN^{1-\sigma} \left(||u||_1 + \sum_{k=1}^{K} ||u||_{\sigma,k} \right). \tag{3.26}$$

It should be mentioned that the result in Theorem 3.2 holds for SEM as well. Note that this result is for a spatial approximation of a function u, although SEM provides great error estimates the solution of the N-S equations are also restricted by the errors due to temporal discretizations. This will be further discussed in Chapter 3.5.

3.4.2 Filtering

Although SEM provides spectral convergence in space, a non-sufficient resolution of the smallest structures often leads to spurious oscillations and an unstable scheme as shown in chapter 2.4.1.2 in [7]. In [19] a filter-based stabilization is introduced for SEM applied on the Navier-Stokes equations. The idea is to project a part $0 < \alpha < 1$ of the solution onto a polynomial space of lower order, explicitly they define the filter F_α as

$$F_\alpha = \alpha I_{N-1} + (1-\alpha) I_d. \tag{3.27}$$

Where I_{N-1} is the projector from P_N to P_{N-1} and I_d is the identity operator. The parameter α is recommended to be somewhere in the interval $(0.05, 0.3)$. The effect of F_α in Legendre space is analysed by Pasquetti and Xu in [20]. A quick demonstration of how the filter works will however be given here.

Let $u = \sum_{i=0}^{N} \hat{u}_i L_i$ be some function, where L_i denote the Lagrange polynomial of order i and $\hat{u}_i$ the corresponding coefficient. The effect of the filter can be given as

$$F_\alpha u = (1-\alpha)\hat{u}_N L_N + \hat{u}_{N-1} L_{N-1} + (\hat{u}_{N-2} + \alpha \hat{u}_N) L_{N-2} + \sum_{i=0}^{N-3} \hat{u}_i L_i. \tag{3.28}$$

From this identity the effect of the filter becomes clear, it is simply removing a part α from the highest mode N to the mode $N-2$. The rest of the coefficients remain unchanged. For a full derivation and discussion on this matter it is referred to chapter 6.5.1 in [7].

The filter in Eq. (3.28) is in this thesis applied such that the k highest Legendre modes are dampened with coefficients $\alpha_i = (\frac{k-i}{k})^2 \alpha_0$, with α_0 being the filter constant for the highest mode. The action of the filter can therefore be described as a matrix equation in two steps, first adding the contribution from the higher polynomials to the lower ones and then subtracting the contribution from the higher polynomials. Let u be the vector containing the Legendre coefficients.

$$\begin{aligned} u^* &= (I + \mathcal{F}_{\alpha,k}) u, \\ u^{**} &= (I - I_{\alpha,k}) u^*. \end{aligned} \tag{3.29}$$

In this case $I_{\alpha,k}$ is a diagonal matrix diag$(0, 0, \cdots, \alpha_{k-1}, \alpha_{k-2}, \cdots, \alpha_0)$ while $\mathcal{F}_{\alpha,k}$ is also sparse and with the same non-zero entries as $I_{\alpha,k}$, but this time they are located on the second sup-diagonal (removing the 2 first zeros).

$$\begin{aligned}(I_{\alpha,k})_{i,j} &= \delta_{i,j}\alpha_{i-(n-1)}, \qquad \text{if } i > n-k+1, \\ (\mathcal{F}_{\alpha,k})_{i,j} &= \delta_{i,j+2}\alpha_{i-(n-1)}, \qquad \text{if } i > n-k+1.\end{aligned} \tag{3.30}$$

The filter is proved to be an effective stabilization method and it preserves the spectral convergence rate. Another property of interest is that the filtering procedure does not imply dissipation of energy, let the energy norm be defined as $E(u) = ||u||^2_{L_2}$. By applying Parseval's identity [21] the difference in energy between the original solution and the filtered solution is given as

$$\epsilon = E(u) - E(F_\alpha u) \tag{3.31}$$

$$= 2\alpha\hat{u}_N(\hat{u}_N||L_N||^2 + \hat{u}_{N-2}||L_{N-2}||^2) - \alpha^2\hat{u}_N^2(||L_N||^2 + ||L_{N-2}||^2) \tag{3.32}$$

$$\approx \frac{2\alpha}{N}\left[(1-\frac{\alpha}{2})\hat{u}_N^2 + \hat{u}_N\hat{u}_{N-2}\right], \tag{3.33}$$

which can take both positive and negative values depending on the sign and size of $\hat{u}_N\hat{u}_{N-2}$. By applying the known norm of the Legendre polynomials the deduced absolute error ϵ of the filtered energy is of order $\epsilon \sim \alpha/N$. The approximation $||L_N||^2 \approx ||L_{N-2}||^2 \approx 1/N$ have been used to achieve the result in Eq. (3.33).

3.4.2.1 A physical approach to the filter

A good physical description of the filter has not yet been presented, this subsection is an attempt to show the resemblance between known properties of some differential equations and the mathematical filter described in the previous section.

Let u be some smooth continuous function, and let $\bar{u} = [\hat{u}_1 L_1(x), \hat{u}_2 L_2(x), \cdots]$ be the vector with the corresponding Legendre basis times the appropriate coefficients. The matrix P_k is a diagonal matrix with ones on the k last entries and 0 on the rest.

Let us consider the two initial value problems

$$\begin{aligned} \frac{\partial v}{\partial t} &= \lambda_1 \Delta v, \qquad v(0) = v_0, \\ \frac{\partial u}{\partial t} &= -\lambda_2 u, \qquad u(0) = u_0, \end{aligned} \tag{3.34}$$

with $\lambda_1, \lambda_2 > 0$. The second IVP in Eq. (3.34) has a known analytical solution $u = u_0 e^{-\lambda_2 t}$ which decays exponentially in time. The first IVP is known as the heat equation and is known to diffuse or smooth out the initial condition. Now let the operators -1 and Δ on the right hand sides of Eq. (3.34) be applied only on the k highest Legendre polynomials. By applying the Legendre decomposition and the truncated operators the system can then be written as

$$\begin{aligned} \frac{\partial \bar{v}}{\partial t} &= \lambda_1 \Delta P_k \bar{v}^T, \qquad v(0) = v_0, \\ \frac{\partial \bar{u}}{\partial t} &= -\lambda_2 P_k \bar{u}^T, \qquad u(0) = u_0. \end{aligned} \tag{3.35}$$

The Laplace operator applied in a Legendre space is known to take a sparse upper triangular form with entries only on even sup-diagonals. By naming this matrix A the entries are given as

$$A_{ij} = 4||L_i||^{-2} \sum_{k=0}^{(j-i)/2-1} ||L_{j-1-2k}||^{-2}, \qquad \text{if } 2 \leq j - i = mod(2). \tag{3.36}$$

This relation is derived in Appendix A.

The matrix-matrix product AP_k will lead to a Laplacian matrix where the $N-k$ first columns are zero while the last k remains unchanged. This modified Laplacian will be denoted A^*. Doing a first order implicit Euler on Eq. (3.35) can be written as

$$\begin{aligned} (I - \lambda_1 \Delta t A^*) u^{n+1} = u^n &\implies u^{n+1} = (I - \lambda_1 \Delta t A^*)^{-1} u^n, \\ (I + \lambda_2 \Delta t P_k) u^{n+1} = u^n &\implies u^{n+1} = (I + \lambda_2 \Delta t P_k)^{-1} u^n. \end{aligned} \tag{3.37}$$

Notice the resemblance to the application of the filter in Eq. (3.29). The second step is identical if

$$\alpha_i = \lambda_2 \Delta t / (1 + \lambda_2 \Delta t) \approx \lambda_2 \Delta t. \tag{3.38}$$

The truncated Laplacian A^* is not equal to the corresponding filter matrix $\mathcal{F}_{\alpha,k}$, but there are some similarities between these to matrices. They are both upper triangular with zeros along the diagonal and positive entries on the second sup-diagonal. The main difference is that the Laplacian applied on the Nth Legendre function L_N affect all the terms $L_{N-2}, L_{N-4}, \cdots$, while the filter only affects L_{N-2}. To illustrate this effect Figure 3.1 shows the non-zero entries of the Laplacian matrix, the red dots indicate the non-zero entries that are not present in the filtering matrix. If however filtering is applied only on the last two nodes, then $\mathcal{F}_{\alpha,2} = A^*$ if $\alpha_i = \lambda_1 \Delta t A_{n-2-i,n-i}$, for $i = 0, 1$ Using the known formula of the norm from Eq. (A.2) we can write this as

$$\alpha_i = \lambda_1 \Delta t(2(n-i)-3)(2(n-i)-2). \tag{3.39}$$

If one were to insist that the Laplacian should work locally in Legendre space, in other

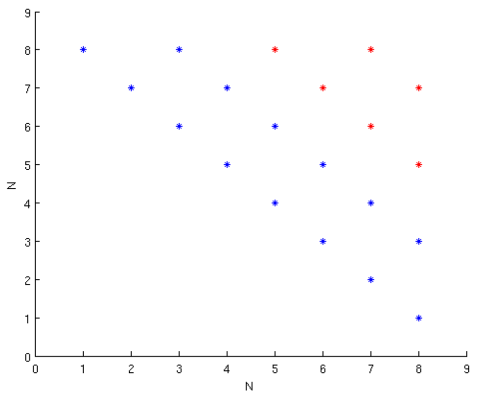

FIGURE 3.1: *The non-zero entries of the Laplacian matrix applied on the last eight Legendre modes. The blue dots are present in both the Laplacian matrix and the filter matrix while the red dots are only non-zero for the Laplacian.*

words imposing $A_{ij} = 0$ for $j - i > 2$, then the Laplacian and the filtering would have the same non-zero entries and by choosing α_i as indicated in Eq. (3.39) would yield equality. Notice in particular the similarity of applying a filter and including a SGS model, with ν_T as a constant in Eq. (2.25) the term $\nabla \nu_T s_{ij}$ reduces to $\nu_T \Delta \mathbf{u}$.

This way of considering the filtering procedure is similar to the variational multiscale (VMS) approach to LES first introduced by Hughes [22]. This method is based on the assumption that the unresolved structures have a negligible effect on the larger scales, hence the SGS model is only included for the small but still resolved scales of motion.

3.4.3 Aliasing

When evaluating the integral surging from the non-linear term in the N-S equations the polynomial to be integrated can be of order $2P+(P-1)$ or even higher depending of the Jacobian. Notice that the number of GLL-nodes are $N = P + 1$, and the corresponding quadrature rule is said to be of order N. An Nth order GLL-quadrature is known to solve an integral exactly when the integrand is of order $2N - 3 = 2P - 1$ or less. Hence the error surging from this evaluation can be of significant size. Applying a quadrature rule of a not sufficiently high order results in an amplification of the lower modes, attempting to compensate for the higher order modes omitted. Since a spectral element method arguably has a good accuracy these "variational crimes" should not be committed, and it is therefore common practice to solve this particular integral using a quadrature rule of order $3/2N$. The concept and illustrative examples are given in Chapter 2.4 in [7]. Aliasing is however not always a necessity, if the grid resolution is sufficiently fine the non-linear term will not need to be solved using a higher order quadrature rule, an example of this is presented in Chapter 6. This is one of the time vs. accuracy questions one have to decide for each problem. Instead of applying the GLL-quadrature "designed" for the basis functions the functions has to be evaluated in a new set of GLL-points with 3/2 as many nodes. This is a costly process and should only be applied when absolutely necessary.

3.4.4 Gordon-Hall algorithm

Working with complex geometries often requires one or more elements to be deformed. The integrals surging from the weak formulation are solved using a quadrature rule over a reference domain $\hat{\Omega} = [-1, 1]^d$ for efficiency and implementation purposes. The Gordon-Hall algorithm is a general method that creates an isometric map from an arbitrary simply connected domain to $\hat{\Omega}$. Let $\tilde{\mathbf{x}}$ be the mapping function from the reference domain to the physical domain given on the form

$$\tilde{\mathbf{x}} = \sum_i \sum_j \sum_k \mathbf{x}_{ijk} l_i(r) l_j(s) l_k(t). \tag{3.40}$$

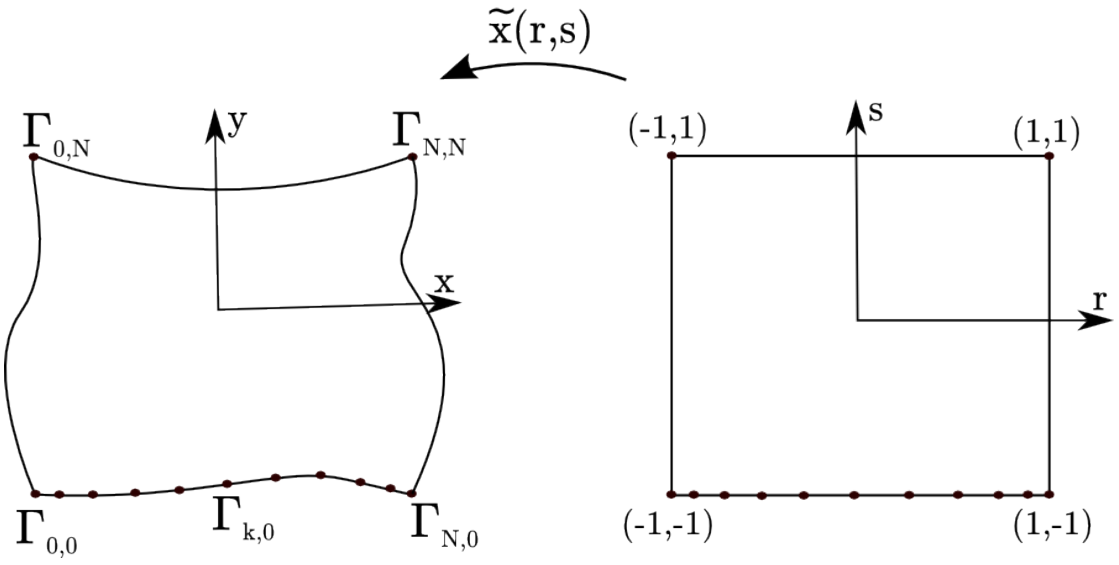

FIGURE 3.2: *An illustration of how the Gordon-Hall algorithm creates a map from the reference element to the deformed element. The GLL-points are drawn along the edge $\Gamma_{k,0}$ on the deformed element which corresponds to $s = -1$ on the reference element.*

l_i being the ith Lagrange polynomial. The full description of the algorithm with helpful figures can be found in [23] chapter 4. Without going to much into the mathematical foundation of this method a more intuitive and implementable presentation of the method will be provided in this chapter. For simplicity a two-dimensional domain will be considered here, and the 3D case will be an easy expansion of the algorithm presented here. Consider a deformed domain $\Omega \in \mathbb{R}^2$ (Figure 3.2), with $\Gamma_{i,j}$ representing the discrete boundary coordinates. The four vertices can then be expressed as $\Gamma_{0,0}, \Gamma_{0,N}, \Gamma_{N,0}\Gamma_{N,N}$. Let ϕ_0, ϕ_N be defined as

$$\phi_0(\xi) = \frac{1-\xi}{2}, \qquad \phi_N(\xi) = \frac{1+\xi}{2}. \tag{3.41}$$

Let $\{\xi_0, \ldots, \xi_N\}_{N+1} = \{-1, \ldots, 1\}_{N+1}$ be the GLL-points corresponding to the Lagrange polynomial of order N. An important property for the functions in Eq. (3.41) is that $\phi_0(\xi_0) = \phi_N(\xi_N) = 1$ and $\phi_0(\xi_N) = \phi_N(\xi_0) = 0$.

The algorithm provides a stepwise routine depending on the complexity of the domain. The first step is to create a mapping to a polygon spanned from the vertices of Ω.

$$\begin{aligned}\tilde{\mathbf{x}}_{i,j} &= \Gamma_{0,0}\phi_0(r_i)\phi_0(s_j) \\ &+ \Gamma_{0,N}\phi_0(r_i)\phi_N(s_j) \\ &+ \Gamma_{N,0}\phi_N(r_i)\phi_0(s_j) \\ &+ \Gamma_{N,N}\phi_N(r_i)\phi_N(s_j)\end{aligned} \tag{3.42}$$

If the edges are straight the algorithm ends here, but for curved edges a second step is performed adding the deformation of the edges.

$$\begin{aligned}\tilde{\mathbf{x}}_{i,j} = \tilde{\mathbf{x}}_{i,j} &+ (\Gamma_{i,0} - \tilde{\mathbf{x}}_{i,0})\phi_0(s_j) \\ &+ (\Gamma_{i,N} - \tilde{\mathbf{x}}_{i,N})\phi_N(s_j) \\ &+ (\Gamma_{0,j} - \tilde{\mathbf{x}}_{0,j})\phi_0(r_i) \\ &+ (\Gamma_{N,j} - \tilde{\mathbf{x}}_{N,j})\phi_N(r_i)\end{aligned} \tag{3.43}$$

In 3D the additional knowledge of the faces may be applied to create mappings from elements with deformed faces as a third step. The only difference when applying this algorithm in three dimensions is that you need to include ϕ for a third coordinate t_k and the number of vertices and edges are 8 and 12 instead of 4 and 4.

3.5 Time integration for incompressible N-S

So far the spatial discretization by SEM has been described in detail and so far been proved to yield spectral convergence, but for unsteady flows the development in time is determined by the temporal discretization and puts an additional restriction on the convergence rate. Because the N-S equations are computationally demanding to solve exactly a large number of splitting methods have been developed, all attempting to find the ideal balance between speed and accuracy. The most common set of solution methods are called projection methods, which first calculates a velocity field that does not fulfill the divergence-free condition and then projecting this field onto a divergence-free space. For an extensive discussion regarding projection methods it is referred to [24]. The projection is done by solving a Poisson equation for the pressure.

3.5.1 Operator-splitting techniques

In this chapter a_j, b_j will denote the coefficients for some explicit and implicit scheme. Let us consider a simplified transient problem

$$\frac{du}{dt} = f(u,t)u + g(t)u. \tag{3.44}$$

f is here a function of u and t, while g is only dependent of the time t. Let superscript denote the time step, such that $g^n = g(n\Delta t)$ for some fixed time step Δt. One step applying a kth order Backward Difference scheme (BDFk) yields

$$\sum_{j=0}^{k} b_j u^{n+1-j} = \Delta t f^{n+1} u^{n+1} + \Delta t g^{n+1} u^{n+1}. \tag{3.45}$$

Now notice that $f^{n+1} = f(u^{n+1}, t^{n+1})$ requires that u is known at time t^{n+1} which is not achievable at the current step. This term is therefore approximated by a kth order explicit scheme leading to

$$\sum_{j=0}^{k} b_j u^{n+1-j} = \Delta t \sum_{j=0}^{k} a_j f^{n-j} u^{n-j} + \Delta t g^{n+1} u^{n+1}. \tag{3.46}$$

Now the terms can be ordered such that only the implicit terms are present on the left hand side,

$$(b_0 - \Delta t g^{n+1})u^{n+1} = -\sum_{j=1}^{k} b_j u^{n+1-j} + \Delta t \sum_{j=0}^{k} a_j f^{n-j} u^{n-j}. \tag{3.47}$$

This way of solving Eq. (3.44) allows easy invertible terms to be solved implicitly while non-linear terms can be extrapolated. In the Navier-Stokes equation this strategy will be applied to split the non-linear term from the rest. In this thesis the schemes BDFk and a kth order extrapolation (EXTk) for $k = 2, 3$ are applied, the coefficients can be found in [25].

It should be mentioned that the explicit/implicit schemes introduced in this section are stable only for operators with eigenvalues below a certain limit γ that depends on the scheme. When applied on the N-S equations it is the convection term $\Delta t M^{-1} C(\mathbf{u})\mathbf{u}$ as observed in Eq. (3.58) that is evaluated by an explicit scheme and it is these eigenvalues that needs to be constrained. Let h^e and u^e denote the grid size and the maximum absolute velocity in element e. According to [7] the maximum eigenvalue of $M^{-1}C(\mathbf{u})$ can be estimated as $\lambda_{max} \approx c \max_e u^e P^2/h^e$, where c is some constant obtained from numerical experiments and analytical results. Notice that $\Delta x^e_{min} = h^e_{min}/P^2$ due to the distribution of the GLL-nodes. This relation leads to an easily computable stability criteria analogous to the CFL-condition,

$$\Delta t \lambda_{\max} = c u^e_{max} \frac{\Delta t}{\Delta x^e_{min}} \leq \gamma. \tag{3.48}$$

This inequality is used to adjust the time step such that the development in time is maximized within the stability criteria for the scheme.

3.5.2 Operator integrating factor schemes (OIFS)

The operator-splitting method described in the previous chapter may lead to an unstable scheme, and require small time steps. OIFS is a similar method but it offers a more stable scheme and is more efficient by using a multi-stage method that allows larger advances in time. The presentation of the method is presented here in a computational fashion, for a full description and derivation of the method it is referred to Maday et al [26].

Throughout this section the NS-equation will be considered in its operational form as introduced in Eq. (2.11)

$$M\frac{d\mathbf{u}}{dt} + C(\mathbf{u})\mathbf{u} = -A\mathbf{u} + D^T p + M\mathbf{f}, \qquad D\mathbf{u} = 0 \tag{3.49}$$

Now let $Q(t)$ be an operator such that $Q(t^{n+1}) = I$ and

$$\frac{dQ(t)M\mathbf{u}}{dt} = Q(t)M\frac{d\mathbf{u}}{dt} + \frac{d}{dt}\left[Q(t)M\right]\mathbf{u}, \tag{3.50}$$

$$= Q(t)M\frac{d\mathbf{u}}{dt} + Q(t)C(\mathbf{u})\mathbf{u}. \tag{3.51}$$

This way Eq. (3.49) can be written as

$$\frac{dQ(t)M\mathbf{u}}{dt} = Q(t)(-A\mathbf{u} + D^T p + M\mathbf{f}). \tag{3.52}$$

Evaluating this equation with a BDFk-scheme results in a system

$$\sum_{j=0}^{k} b_j Q(t^{n+1-j}) M\mathbf{u}^{n+1-j} = \Delta t\, Q(t^{n+1})(-A\mathbf{u}^{n+1} + D^T p^{n+1} + M\mathbf{f}^{n+1}). \tag{3.53}$$

Applying the fact that $Q(t^{n+1}) = I$ enables Eq. (3.54) to be written as

$$b_0 M\mathbf{u}^{n+1} + \sum_{j=1}^{k} b_j Q(t^{n+1-j}) M\mathbf{u}^{n+1-j} = \Delta t(-A\mathbf{u}^{n+1} + D^T p^{n+1} + M\mathbf{f}^{n+1}). \tag{3.54}$$

Notice how all the easily invertible operators are evaluated implicitly, while the convective non-linear term is hidden in the BDFk scheme. OIFS allows the terms in the sum to be calculated in an elegant fashion. First of all the auxiliary variable $\tilde{\mathbf{u}}_j$ is defined such that $Q(t^{n+1-j})M\mathbf{u}^{n+1-j} = M\tilde{\mathbf{u}}_j$ thus enabling the summation expression to be found by solving the initial value problem

$$\begin{aligned} M\frac{d\tilde{\mathbf{u}}_j}{ds} &= -C(\tilde{\mathbf{u}}_j(s))\tilde{\mathbf{u}}_j(s), \qquad t^{n+1-j} \leq s \leq t^{n+1} \\ \tilde{\mathbf{u}}_j(t^{n+1-j}) &= \mathbf{u}(t^{n+1-j}). \end{aligned} \tag{3.55}$$

Notice how the integrational factor $Q(t)$ is never evaluated directly.

The final scheme applied for solving Eq. (3.49) when applying OIFS consists of one implicit scheme for solving Eq. (3.54) and an explicit multistage scheme for solving Eq. (3.55). When applied in this thesis the first scheme corresponding to the b_j coefficients is an implicit BDFk-scheme while the second is an explicit 4th order Runge-Kutta scheme (RK4). Solving Eq. (3.55) with a multi-stage method implies a bit more work per time step, but as it stabilizes the routine larger advances in time can be made and the overall efficiency improves.

3.5.3 Fractional step - $(P_N P_N)$

The formulation applied in this method is justified by Guermond in [11] and is based on the following formulation of the Stokes problem

$$\begin{aligned} -\mu\Delta\mathbf{u} + \nabla p = \mathbf{f} \text{ in } \Omega, &\qquad \mathbf{u} = \mathbf{0} \text{ on } \partial\Omega, \\ \Delta p = \nabla\cdot\mathbf{f} \text{ in } \Omega, &\qquad \partial_n p = (-\nabla\times\nabla\times\mathbf{u} + \mathbf{f})\cdot\mathbf{n} \text{ on } \partial\Omega. \end{aligned} \tag{3.56}$$

This formulation is obtained by taking the divergence of the momentum equation and applying the divergence-free condition to obtain the second equation in Eq. (3.56). The vector identity $-\Delta\mathbf{u} = -\nabla\nabla\cdot\mathbf{u}+\nabla\times\nabla\times\cdot\mathbf{u}$ combined with the divergence-free condition yields the boundary condition for the pressure.

Fractional step exploits this algebraic splitting and the algorithm can be divided into four separate steps. The momentum equation will still be considered in its operational form

$$M\frac{d\mathbf{u}}{dt} + C(\mathbf{u})\mathbf{u} = -A\mathbf{u} + D^T p + M\mathbf{f}. \tag{3.57}$$

Where $M, A, D, C(\mathbf{u})$ denotes the mass integral, Laplacian, gradient and non-linear operator. A schematic overview of the method is stated below, where the equations on the right hand side are solved and the updated solution is stated on the left hand side. By performing these steps the solution $(\mathbf{u}, p)$ is developed one time step from $(\mathbf{u}^n, p^n)$ to $(\mathbf{u}^{n+1}, p^{n+1})$.

$$\begin{aligned}
\mathbf{u}^* &= -\sum_{j=1}^{k} b_j \mathbf{u}^{n+1-j} + \Delta t M^{-1}\left(\sum_{j=1}^{k} a_j C(\mathbf{u}^{n+1-j})\mathbf{u}^{n+1-j} + M\mathbf{f}^{n+1}\right),\\
DM^{-1}D^T p^{n+1} &= \frac{1}{\Delta t} D\mathbf{u}^*,\\
\mathbf{u}^{**} &= \mathbf{u}^* + \Delta t M^{-1} D^T p^{n+1},\\
(b_0 M + \Delta t A)\mathbf{u}^{n+1} &= M\mathbf{u}^{**}.
\end{aligned} \tag{3.58}$$

As earlier mentioned this method is convenient because it allows us to handle the different terms with different solution techniques. Hence the term including the non-linear skew-symmetric advection matrix $C(\mathbf{u})$ will be approximated by an kth order extrapolation (EXTk) scheme. The first step can also be evaluated in an OIFS procedure to gain stability. This implies using the discretization introduced in Eq. (3.54) with only the forcing function on the right hand side, and solving the IVP Eq. (3.55) with RK4 to obtain $\mathbf{u}^*$.

The second equation is the Poisson equation for the pressure which assures a divergence-free velocity field, and it is this step along with step three that allows this to be classified as a projection method.

Note that $p \in L^2 \supset H^1$ hence the Poisson equation is somewhat different from the one normally studied in textbooks. Another difficulty is the treatment of the boundary conditions. Ideally the BC's should be determined by the velocity field $\mathbf{u}^{n+1}$, but since this solution is yet to be calculated the intermediate velocity field $\mathbf{u}^*$ is used to impose the boundary conditions. With p^{n+1} known the third equation is simply an update of the velocity to impose the divergence-free condition. Now the last equation is solved implicitly due to the nice symmetric structure of the Laplacian operator. This results in a system equivalent to the Helmholtz problem as discussed in Chapter 3.3 and 3.4.

This method provides an efficient algorithm, and by subtracting $1/\Delta t M^{-1} D^T p^n$ from the first equation in 3.58 the Poisson equation can be solved for $\delta p^{n+1} = p^{n+1} - p^n$ instead of p^{n+1}. This yields a scheme with errors in the L_2-norm of orders $\mathcal{O}(\Delta t)$ for pressure and $\mathcal{O}(\Delta t^2)$ for velocity as stated in [24].

3.5.4 Discrete splitting - $(P_N P_{N-2})$

The fractional step method is a splitting method based on the idea that two analytical operators can be applied in sequence and still provide a good result. The method presented in this section makes no such assumption and splits the discrete system of equation instead of applying the operators in sequence. The algorithm presented here is similar to the Uzawa algorithm, but with some adjustments to make it more efficient. The detailed description regarding the implementation in Nek5000 is found in [27].

To start the explanation of the method we continue considering the incompressible N-S equations in their operational form

$$\frac{1}{\Delta t} M\mathbf{u}^{n+1} - D^T p^{n+1} + A\mathbf{u}^{n+1} = M\tilde{\mathbf{f}}^{n+1}, \qquad D\mathbf{u}^{n+1} = 0. \tag{3.59}$$

The outline of the method is based on Eq. (3.3), but with some changes. Since this is a method for the unsteady N-S equation the time derivative has to be included, and the non-linear term which is treated explicitly as studied in Chapter 3.5.1 and 3.5.2 is added as a part of the right hand side function. So $M\tilde{\mathbf{f}}^{n+1}$ does in this equation incorporate both the original forcing function, the non-linear term and the explicit part of the time-derivative from the BDFk-scheme. By doing this reformulation the unsteady Stokes problem is obtained and algorithms studied for this problem can be applied. By defining the matrix $H = 1/\Delta t M + A$ Eq. (3.59) can be written as

$$\begin{pmatrix} H & -D^T \\ D & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^{n+1} \\ p^{n+1} \end{pmatrix} = \begin{pmatrix} M\tilde{\mathbf{f}}^{n+1} \\ 0 \end{pmatrix}. \tag{3.60}$$

The splitting that will be done in the next step will use the pressure difference $\delta p^{n+1} = p^{n+1} - p^n$ as was done for the fractional step algorithm. Eq. (3.61) can be restated as

$$\begin{pmatrix} H & -D^T \\ D & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^{n+1} \\ \delta p^{n+1} \end{pmatrix} = \begin{pmatrix} M\tilde{\mathbf{f}}^{n+1} + D^T p^n \\ 0 \end{pmatrix}. \tag{3.61}$$

Solving this exactly is known as the Uzawa algorithm and is computationally demanding and converge slowly. To overcome this issue simplifications and reformulations are made which saves a lot of computational time at the cost of accuracy. The system is rewritten using a LU-factorization of the matrix in Eq. (3.61), which will allow the solution to be found in two separate steps. This requires the inverse of H that will be replaced by an approximation $Q \approx H^{-1}$. The matrix decomposition is given as

$$\begin{pmatrix} H & -D^T \\ D & 0 \end{pmatrix} \approx \begin{pmatrix} H & 0 \\ -D & -DQD^T \end{pmatrix} \begin{pmatrix} I & -QD^T \\ 0 & I \end{pmatrix}. \tag{3.62}$$

Applying these two matrices leads to a two step algorithm on the form

$$\begin{pmatrix} H & 0 \\ -D & -DQD^T \end{pmatrix} \begin{pmatrix} \mathbf{u}^* \\ \delta p^{n+1} \end{pmatrix} = \begin{pmatrix} M\mathbf{f}^{n+1} + D^T p^n \\ 0 \end{pmatrix}, \tag{3.63}$$

$$\begin{pmatrix} I & -QD^T \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{u}^{n+1} \\ \delta p^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{u}^* \\ \delta p^{n+1} \end{pmatrix} \tag{3.64}$$

To clarify what is going on, a brief description of each step is given, The first step in Eq. (3.63) corresponds to an initial solution of the velocity using the old pressure value, notice that this will not guarantee a divergence-free velocity. The second step in Eq. (3.63) is the discrete Poisson equation for the pressure and will make sure that the pressure corresponds to a divergence-free flow. The first step in Eq. (3.64) is just a projection of the velocity field onto a divergence-free space. The final step is of no real value and will instead be replaced by an update of the pressure $p^{n+1} = p^n + \delta p^{n+1}$.

In this thesis the approximation of the inverse Helmholtz matrix was of first order, $Q = \Delta t M^{-1}$. Higher order approximations can be made, but this is a convenient definition since M is a diagonal matrix.

Unlike the fractional step method this way of solving the N-S equations requires the discrete spaces for velocity and pressure to meet the inf-sup condition stated in Eq. (3.4). It also induces a discrete splitting error. The splitting error induced by this scheme has been a topic of discussion for many years, see for instance the discussion between Perot and Abdallah in [28], [29] and [30]. The author will not choose sides in this debate, but state that there will be some numerical differences between these two methods which will be presented in Chapter 6.

The method yields a scheme with errors in the L_2-norm of orders $\mathcal{O}(\Delta t)$ for pressure and $\mathcal{O}(\Delta t^2)$ for velocity as proved in [24], for a second order BDF scheme.

Chapter 4

Application of Nek5000

Many numerical solvers for turbulent flows are available on the market. From large commercial softwares such as Fluent which runs as a black-box solver, to full open-source codes such as Nek5000 and OpenFOAM. The solvers can vary in the numerical method; Finite volume, Finite Difference, Finite Element, Spectral Element etc., the particular algorithm for resolving the pressure-velocity coupling, for instance fractional step, pressure correction, inexact factorization and Uzawa. The type of simulation available also varies from solver to solver, whether they apply RANS, LES, DNS or a variety of these. Although most solvers offer multiple of the settings listed above it is important to be aware of their strengths and weaknesses before choosing which one to use. This chapter is devoted to the handling of Nek5000, and can serve as a brief introduction to the code.

4.1 Nek5000 basics

Nek5000 is a flow solver developed mainly by Paul Fischer and has through the past 20 years had several contributors. It is an open-source code applicable to many different types of flow and it has been put a lot of effort into the parallelization of the code, guaranteeing optimal speedup. All the parallelization is accessed through subroutines and functions, enabling the user to make advanced functions without having to deal directly with the MPI library. With SEM as the numerical method applied it is possible to obtain very accurate results.

Nek5000 provides some basic tools for generation of mesh. For more complex geometries this tool cannot compare with more visual softwares such as ICEM from ANSYS that exports mesh to several numerical solvers such as Fluent and Nastran. An automatic way of converting a mesh created in ICEM to the format required by Nek5000 is therefore very useful. The way the mesh is created in this thesis is visualized in Figure 4.1.

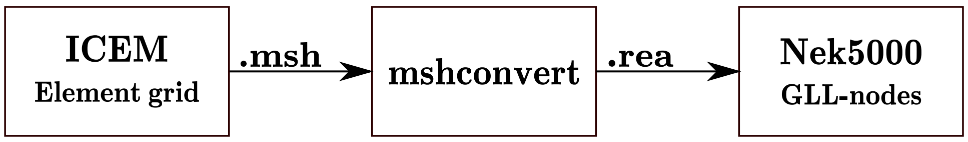

FIGURE 4.1: *Visualization of how the mesh is created. The elemental mesh is first generated using ICEM, the script mshconvert converts this to a .rea-file and finally the distribution of the GLL-nodes is done during the initialization in Nek5000*

So far Nek5000 has supported three automatic routines for generating curved edges; circles in 2-D geometries, spherical shell elements and a general 2nd degree interpolation. Further manipulation of the element edges is left to the user to define manually in `usrdat2` for each particular problem. One of the objectives of this thesis is to make Nek5000 more user-friendly and create automatic routines to handle complex geometry. Before the work regarding the mesh routines are further elaborated an overview of the file structure will be presented.

4.2 Editable files

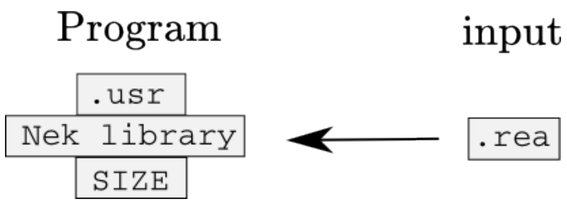

FIGURE 4.2: *Visualization of how the file structure in Nek5000 is built up.*

Working with Nek5000 requires knowledge to some practicalities. The program is recompiled for every case and the user specify all the necessary information in the three files `{.rea,.usr,SIZE}`. `.usr` and `SIZE` are compiled with the standard Nek5000 library using makenek which creates the executable file `nek5000`, they can be considered the surface and the core of the entire program. The `.rea` file contains case-specific information read during the initialization of the compiled program. The user guide [25] contains a tutorial that explains the necessary steps on how to get started with Nek5000. The next chapters will try to give some understanding on how the user is able to make the changes necessary for each case. Figure 4.3 illustrates how the files work together.

4.2.1 SIZE

Since Nek5000 is mostly based on Fortran77 all memory allocations are done statically and must be specified explicitly before runtime. Most of the variables used to determine the memory usage are stated in `SIZE`. The size of the working arrays necessary to perform the calculations are mostly defined by the upper limits of elements, processors, scalars and of course the polynomial degree of the local Lagrange functions. These variables define the sizes of almost all the arrays used in the program so it is important to define these variables as accurately as possible to optimize memory usage. The `SIZE` file can be considered as the necessary base of Nek5000.

4.2.2 .REA

Lines	Section Name	Specifications
103	PARAMETERS	All problem-specific variables
K	PASSIVE SCALAR DATA	Convective and diffusive constants for scalars
K	LOGICAL SWITCHES	Boolean variables defining the solution method
E	MESH DATA	All nodes and elements are specified here
E	CURVED SIDE DATA	All the curved sides are specified here
E	FLUID BC	BC type for all elements and their faces
E	THERMAL BC	Thermal BC type for all elements and their faces
K	PRESOLVE/RESTART	Filename(s) of an initialized solution
K	INITIAL CONDITIONS	possibilities to specify IC further
K	OUTPUT FIELD	information that will be written to file

TABLE 4.1: *An overview of the different sections in .rea. E represents a predefined number depending on your problem which scales roughly as the number of elements, while* $K \approx 1 - 25$ *are user defined numbers.*

In `.rea` all the problem specific parameters are given. While the content in `SIZE` is an absolute necessity to even compile the program, `.rea` contains variables that are not used until the initialization of the case. The structure of the file is given in Table 4.1. Of the 103 variables specified in the beginning of the file there are roughly 50 of them that are used. Note that apart from the mesh information `.rea` restricts itself to single variables and boolean flags while `.usr` needs to be applied for more advanced implementations.

4.2.3 .usr

This file contains a series of standard routines open for modification by the user. In addition the user is free to specify new routines if needed. A description of these routines are given in the Nek5000 User manual [25]. A list of those frequently used for this thesis are described below,

- `userbc` - Define the boundary conditions on the inflow boundary.
- `uservp` - Define variable properties, impose the eddy viscosity when applying LES.
- `userchk` - Read inflow data, and specify the output.
- `usrdat2` - Project the geometry onto a deformed general surface. The details of how this routine is used will be specified further in Chapter 5.
- `usrdat3` - Defines the interpolation algorithm that is applied to the inflow data.

In addition to these routines all user-defined functions are specified in this file. The LES implementation in Nek5000 is based on several subroutines in addition to those stated above. A list of some of the variables and functions applied for the implementations in this thesis are stated in Appendix B. The `.usr` file can be considered as the surface of Nek5000, easily accessible for the user.

4.3 The basics of the solver

The most important building blocks in Nek5000 are the `fluid` and `heat` functions that solves the N-S and passive scalar equations. The N-S solver works in different ways depending on the mathematical formulation enabled whereas the PS equation, which does not depend on the pressure, is solved similarly for P_NP_N and P_NP_{N-2}. For the sake of clarity, Figure 4.3 shows how the algorithms are called from the main routine. This is a simplified flow chart that does not include choices such as filters, preconditioners, LES-model, de-aliasing etc. but it explains how the two main algorithms are selected using the boolean variable `ifsplit`.

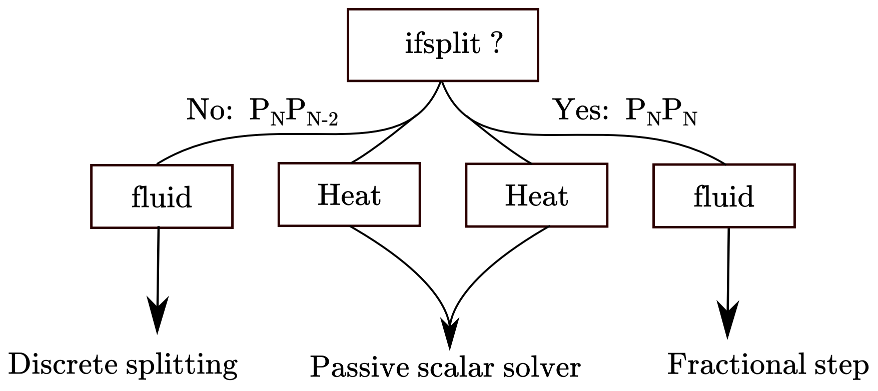

FIGURE 4.3: *Visualization of the steps in Nek5000.*

The description of the routines corresponding to discrete splitting and fractional step are found in Chapter 3.5.4 and 3.5.3. For further details regarding the implementation in Nek5000 it is referred to [27] and [31].

As mentioned before an important difference between these two implementations is the fact that the $P_N P_N$ implementation is based on an analytical splitting algorithm, while the $P_N P_{N-2}$ formulation is based on a splitting of the discrete scheme. The boundary condition of the pressure Poisson equation in the fractional step method is often mentioned as a major difference between these two schemes, and a reason to choose the latter. It is however stated and proved in for instance [24] that the discrete splitting also introduces an erroneous boundary condition weakly.

4.4 Nek5000 for complex geometries

Nek5000 has its own mesh generator *prenek* that allows the user to define geometries and curves using analytical expressions. This tool does however have limited visual support, and can not compare with for instance ICEM when working with more complex geometries.

For curved geometries that can be expressed analytically such as bent cylinders, ellipsoids etc. the recommended procedure is to project the points of interest onto the surface in `usrdat2`. Even for a simple shape such as a sphere some implementation has to be done and it demands that the user has knowledge to Fortran77 and the structure of Nek5000. The necessary implementation consists of two steps

1. determine the faces that belong to the deformed surface
2. project the predefined GLL-points onto the deformed surface

This can be done without too much work for shapes with a known analytical expression such as a cylinder or a sphere, but for some general CAD geometry it is no way to perform this projection routine. This is a vulnerable point for a SEM solver since the elements generated by the mesh are relatively coarse. Many Finite volume based solvers do not support curved elements simply because the complex geometries are resolved with a sufficiently high resolution and it is of no interest to approximate them any better. However for a spectral element solver it is necessary to address this problem since spectral convergence for the approximated solution in P_N is not achievable if the geometry is only represented in P_1 or P_2.

4.4.1 converting mesh from ICEM to Nek5000

For more advanced geometry it is considered a necessity to have some visual mesh generator. In this thesis all of the meshes have been made in ICEM and then converted using the python script `mshconvert` as illustrated in Figure 4.1. Since a lot of the work done regarding the mesh was done based on this script a brief description will be given here.

ICEM is able to write mesh files for many different solvers, including Fluent and Nastran. The original `mshconvert` script can take any of these two formats as input and write a corresponding `.rea` file with a predefined parameter list. It supports curved edges by applying the midpoint notation, and all velocity boundary conditions available in Nek5000. It also has a propagation parameter for a cylinder, this allows element edges inside the domain to be affected by the curvature of the geometry. The idea is that a smooth transition of the element deformation, will improve the Jacobians and lead to better accuracy.

Chapter 5

Implementation

This chapter aims to present the implementations done in this thesis. The routines implemented are meant to develop Nek5000 functionalities in the encounter with complex geometries.

As shown in Figure 4.1 the coarse element grid is created in ICEM and then converted using the existing python script `mshconvert` while the distribution of GLL-nodes and the simulation itself was performed in Nek5000. The input file `.rea` contains exact information about the corners, limited information about the edges, and no information about the faces of the elements. Hence the distribution of the GLL-nodes on a non-regular surface requires data exceeding the one in `.rea`.

Two advancements have been made in this thesis regarding the work with irregular geometries in Nek5000. The first part is a fully automatic procedure that projects any edge onto a circle segment. This is a convenient method when working with cylinder geometries and other similar shapes. An equivalent routine exists for 2D geometry, and some special cases of 3D geometry as well. The routine created for this thesis is however a general method in 3D that allows an edge to be specified as a circle segment in any plane independent of the coordinate axes. The second part is an attempt to improve the possibilities to work with general boundary surfaces by creating a semi-automatic procedure allowing the user to represent any geometry with polynomials of the same order as the basis functions for the velocity. The algorithms are presented in Chapter 5.

Propagation of curved geometry was also experimented with. The effect observed on the grid was very small, and a similar test was done in two dimensions on an annulus by Korczak and Patera [17]. Their results suggests that the accuracy is not affected in any significant way by doing a similar propagation. Based on these results and my own observation it was decided to rather focus on the two routines mentioned above.

5.1 Project edges onto a specified arc

The Gordon Hall (GH) algorithm that is described in Chapter 3.4.4 was already implemented as a function in the Nek library. By defining the GLL-nodes on the curved edges such that they correspond to an arc, the GH-algorithm is able to distribute the internal nodes accordingly.

The curved edges are specified in `.rea` and have until now been read as a second degree polynomial or as a part of a spherical shell. The routine `xyzarc()` was created to process curved edges specified in `.rea` with a radius and a center. It can be considered as an alternative to the already implemented `xyzquad()` in `genxyz.f` which generates curved edges represented as second order polynomials. The algorithm is described below and Figure 5.1 gives a visual representation of the situation.

The two end nodes of the edge are denoted a and b. The midpoint of the linear edge is named c, while θ is the full angle of the circle sector, cc is the center coordinates, g denotes the vector containing the GLL-points in $[-1, 1]$ and r will be the radius.

```
l = a-b                          ! vector between the corner nodes
c = (a+b)/2                      ! midpoint location
h = c-cc                         ! height of the framed triangle
θ = arctan(abs(l)/2*abs(h))       ! half the angle of the circle sector
g = g*θ                         ! angles to the GLL-points on the circle-sector
!---------- Finding the intersecting points ----------!
!---- x on the line l, and extend x-cc to the arc ----!
do k=1,lx1             ! for the number of nodes in one direction
   α = h*tan(g[k])              ! offset from the midpoint on l
   x = c-α*l/abs(l)              ! actual coordinate on l
   m = x-cc                      ! hypotenuse of the imposed triangle
   edge(k) = cc+r*m/abs(m)       ! final coordinate on the arc
enddo
```

This code defines the GLL-nodes on a circle sector corresponding to the radius and circle center provided in section `CURVED SIDE DATA` in `.rea`. The remaining operation is to call the Gordon Hall algorithm and create the internal GLL-points. To make this imple-

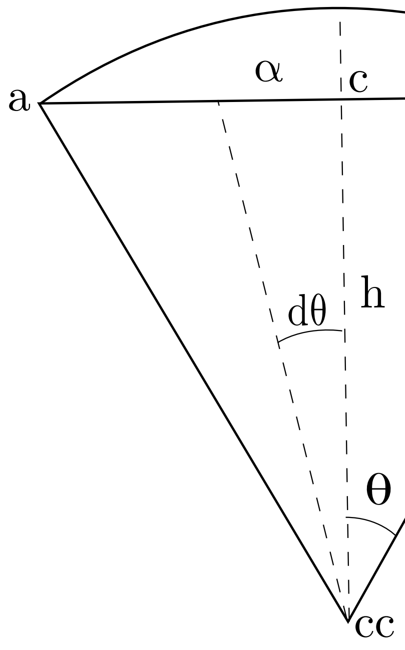

FIGURE 5.1: *A sketch of the curved edge and the variables necessary to calculate the projection*

mentation fully automatic a small change in the script `mshconvert` was also necessary. Since ICEM provides the midpoints of the curved edges, a function `points2circ` was added to convert the midpoint information originally provided to a circle center and its corresponding radius.

5.2 General surface projection

The routine `xyzarc()` enables the user to more accurately represent circular edges. For more complex geometry such as actual terrain and other surfaces without any analytic expression, large elements make the geometrical representation difficult. Theoretically the GLL-points can be projected onto a non-analytical surface, but since the element mesh is created in a different program the necessary information is not available to Nek5000. The idea is to create an additional surface mesh in ICEM such that the nodes in this mesh describes the surface fine enough to distribute the GLL-nodes correctly in Nek5000. During the initialization of the mesh in Nek5000 the program can read this information and project the GLL-nodes onto the provided surface. The routine was made as automatic as possible, and can be summarized in these three steps

1. Create initial mesh and convert to `.rea` applying `mshconvert`.

2. Create refined surface mesh on the non-regular surface.

3. Enable projection by adding the line `call surfprojection(n,a,b)`.

To explain the input parameters, let us first agree upon some naming conventions. The set of surface points is denoted S, while f_e denotes the set of points in element e that belong to the face that is going to be projected onto the surface described by S. The number of interpolation points is denoted n, which has a maximum upper limit of three, since the surface mesh is recommended to be a tetrahedra mesh. Setting $n = 1$ simply projects a point in f_e to the closest one in S. The interpolation algorithm used for this experiment is an inverse distance interpolation which applies the inverse distance squared as weights. The distance between a point $\mathbf{x}_0 \in f_e$, and a point $\mathbf{x}_i \in S$ is given as

$$||\mathbf{x}_0 - \mathbf{x}_i||_a = a||\mathbf{x}_0 - \mathbf{x}_i||_{l^2} + (1-a)||\mathbf{n}_0 - \frac{(\mathbf{x}_0 - \mathbf{x}_i)}{||\mathbf{x}_0 - \mathbf{x}_i||_{l^2}}||_{l^2}, \tag{5.1}$$

with $\mathbf{n}_0$ as the surface unit normal vector of the initial element face in the point $\mathbf{x}_0$.

The last parameter is provided to determine the size of the working array, which is denoted `wrk` in the pseudo code for this routine. The projection is done element wise, and to avoid unnecessary calculations a subset $S_e \subset S$ is used in Eq. (5.1). Let $\mathbf{x}_e$ be the point at the center of face f_e with radius r_e, the subset is then formulated as $S_e = \{\mathbf{x} \in S, ||\mathbf{x}_e - \mathbf{x}|| < b \cdot r_e\}$.

Let $\{\mathbf{x}\}_m$ be the m closest points to $\mathbf{x}_0$ in the sense of the norm defined by Eq. (5.1). The new point $\mathbf{x}^*$ is chosen such that it minimizes the distance to the points $\{\mathbf{x}\}_m$ along the line $\mathbf{x}_0 + k\mathbf{n}_0$. Mathematically this can be stated as,

$$\begin{aligned} &\min_{k\in\mathbb{R}} E(k), \\ E(k) &= \sum_{i=1}^{m} w_i|\mathbf{x}_0 + k\mathbf{n}_0 - \mathbf{x}_i|^2, \\ w_i &= |\mathbf{x}_0 - \mathbf{x}_i|^{-1}. \end{aligned} \tag{5.2}$$

The choice of the weighting function is based on the intuitive idea that the points that initially were closest to $\mathbf{x}_0$ should have bigger influence when minimizing along the normal. The minimization problem is a second degree equation with respect to k and can therefore be solved exactly. To do this let us define the vectors $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{x}_0$ and then find the k_0 such that $E'(k_0) = 0$,

$$E'(k) = 2\sum_{i=1}^{m} w_i \mathbf{n}_0 \cdot (k\mathbf{n}_0 - \tilde{\mathbf{x}}_i), \tag{5.3}$$

$$E'(k_0) = 0 \implies k_0 = \frac{1}{\sum_{i=1}^{m} w_i} \sum_{i=1}^{m} w_i \mathbf{n}_0 \cdot \tilde{\mathbf{x}}_i, \tag{5.4}$$

$$\mathbf{x}^* = \mathbf{x}_0 + k_0 \mathbf{n}_0. \tag{5.5}$$

An important argument for projecting the points in this manner is that for a plane element face all nodes move parallel to each other thus maintaining their relative position.

In addition to the standard Nek5000 library the file `surfpro.f` needs to be added to the folder `trunk/nek/` along with all the other scripts applied by Nek5000. This implementation could be done directly in `.usr`, but it is of practical interest to keep this file as tidy as possible. Two external files are also generated by the modified `mshconvert` script for the algorithm to work. The file `surf.i` contains all the coordinates to the points on the refined surface, while the element and face number of all the faces that are going to be projected are written to `bdry.i`.

The algorithm is best explained through a simple box with a non-regular floor. An example of this situation is the hill of Ekeberg. Before describing the algorithm let $E_{tot} = n_x n_y n_z$ be the total number of elements, N is the polynomial degree and let us for simplicity assume that $n_x = n_y = n_z$ such that $E = E_{tot}^{2/3}$ is the number of elements containing a face on the non-regular surface. The number of points on the refined surface N_s should be approximately EN^4 in order to describe the surface for all the GLL-points that belong to the boundary. This estimate assumes that the surface mesh is equidistantly distributed whereas the GLL-nodes are denser along the boundaries of each element, $\Delta x_{min} = \mathcal{O}(1/N^2)$.

The pseudo code for the algorithm is listed below with the temporal costs commented out.

```
iwrkelem = read(bdry.i)
wrksurf = read(surf.i)
do e,f in iwrkelem   !O(E)
  wrk = create_working_surface(e,f,wrksurf)  !O(EN^4)
  do i in GLL-nodes   !O(N^2)
    interp = init_interpolation_array() !O(1)
    do j in wrk   !O(N^4)
      update_int_array(interp,wrk(j)) ! O(1)
    enddo
    set_new_point(interp,wrk,i,e,f) ! O(1)
  enddo
  fix_GLL() !O(N^3)
enddo
fix_geom()
```

To understand the algorithm a short description of the functions is given in the list below, note that these are idealized routines simplified to explain the flow of the total algorithm, for all auxiliary subroutines implemented see Appendix B.2.3.

- create_working_surface(e,f,wrksurf) – Loops through all the nodes in $wrksurf$ and adds the surface coordinates within a certain radius to the center of face f on element e to the array wrk. This saves time in the search for interpolation points for each GLL-node.
- init_interpolation_array() – initializing the array containing the closest points on the surface for the current GLL-node.
- update_int_array(interp,wrk(j)) – compares the surface point j to the already existing interpolation points in *interp* and adds it to the list if it is found to be closer to the ith GLL-node.
- set_new_point(interp,wrk,i,e,f) – updating the new GLL-point determined by the surface points in interp.
- fix_GLL() – There is a risk after distributing the GLL-points on the surface that some of the internal GLL-points falls outside the element. This function distributes all internal GLL-points correctly between the projected face and the opposite.

- fix_geom() – An already existing Nek routine that makes sure the manipulation of the geometry is consistent with neighbouring elements, and distributes the internal GLL-points correctly.

Although this routine is only called once, and therefore will not contribute significantly to the total runtime of the program it is desirable to have a fast algorithm. Another analysis important to be made is the amount of extra storage space needed for this algorithm. By analysing the pseudo code the time of the algorithm should be of order $\mathcal{O}(E(EN^4 + N^2N^4 + N^3)) = \mathcal{O}(EN^4(E + N^2))$ and the amount of additional storage space will be of order $\mathcal{O}(EN^4 + E + N^2) = \mathcal{O}(EN^4)$. By reading `surf.i` directly in `create_working_surface` the amount of storage space drops to $\mathcal{O}(E + N^2)$, but this requires opening and reading through the entire file for each element.

The routine attempts to be as automatic as possible and the only implementation necessary is a call from `usrdat2` with 3 input variables.

Now an illustrative example of how this method is applied by the user. Say you have a project called "myFlow", and the mesh and surface mesh created in ICEM are named mesh_myFlow and surfmesh_myFlow. The following commands are then executed

```
>> ./mshconvert --mesh mesh_myFlow
    --reafile init.rea --outfile myFlow.rea
    --tol 1e-3 --temperature True --curvetype A

>> ./mshconvert --mesh surfmesh_myFlow
    --mesh_format surface
```

After running the commands above the only thing left is to add `call surfpro(n,a,b)` to `usrdat2`.

This is an initial framework for surface projection that seems to work well for the geometries tested in this thesis. The code is also structured such that it is easy to manipulate and several ideas for further work on this algorithm will be discussed in Chapter 7.

5.2.1 Test of the projection routine

To test the algorithm described in Chapter 5.2 the hill of Ekeberg in Oslo and a helicopter body was used. These surfaces were loaded as `.tin` files in ICEM and coarse hex meshes were created around the surfaces. The domains are presented in Figure 5.2 and 5.3, notice the coarse element sizes in the figures. Especially in the hill of Ekeberg the mesh edges clearly give a bad representation of the actual geometry. These geometries were chosen because they resemble a typical problem with spectral elements. Since the initial element mesh is relatively coarse it does not capture all the details in the geometry and the GLL-nodes distributed on the faces corresponding to the unstructured surface will be misplaced. With the routine described in Chapter 5.2 the surface is approximated accurately by higher order polynomials. The algorithm restricts itself to relatively

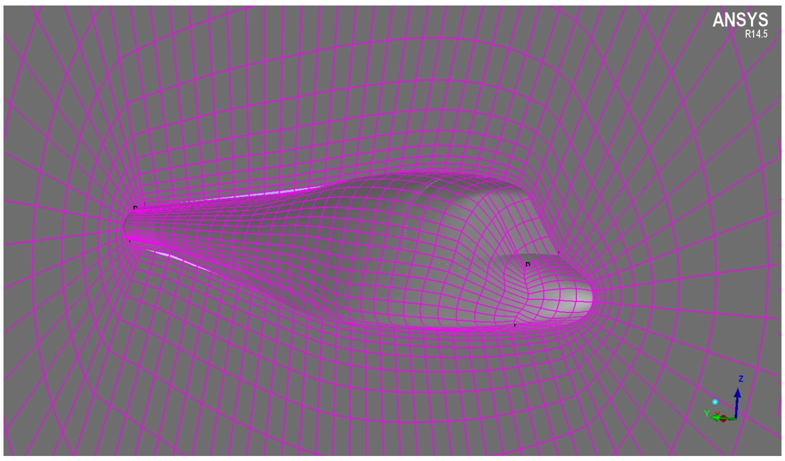

Figure 5.2: *Body of a helicopter.*

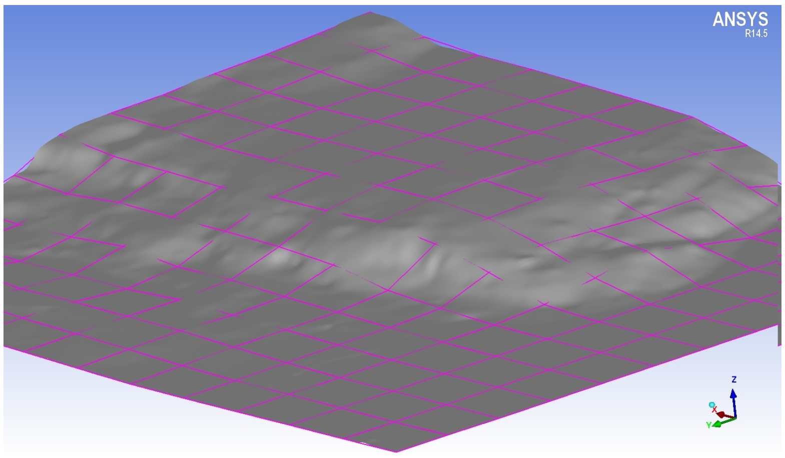

Figure 5.3: *Hill of Ekeberg.*

smooth surfaces, since the polynomial describing the surface is typically of order $P \approx 7$.

5.3 Spatial averaging routine

The dynamic Smagorinsky model has previously been implemented in Nek5000 for flow in a channel. The SGS model as described in Chapter 2 depends on an averaging routine to calculate the dynamic Smagorinsky constant. The previous implementation in Nek5000 applies a planar average routine in addition to averaging in time. The planar averaging is based on the assumption that the Smagorinsky constant is the same for all points with equal distance to the walls of the channel. This is a case specific averaging routine based on the assumption of homogeneous turbulence in the entire plane, hence only applicable to flows in idealized geometries.

When applying dynamical Smagorinsky to Case 1 a new spatial mean routine had to be applied for it to be stable. It was first attempted to average only in time, but this proved not to be sufficient. It was therefore implemented a routine for taking the average within each element, let c^e_{num}, c^e_{den} denote the numerator and the denominator in Eq. (2.23). The means are then calculated as

$$c^e_{num} = \frac{1}{V}\int_{\Omega_e} c^e_{num} d\,\Omega \approx \frac{1}{V}\sum_{i=1}^{N^3} \rho_{i,e} c^e_{num,i}. \tag{5.6}$$

And similarly for c^e_{den}. The coefficients $\rho_{i,e}$ are found in the array `BM1(lx1,ly1,lz1)` in the file `MASS`.

Chapter 6

Case studies and results

This chapter will present the cases investigated and the results acquired. The main tools in addition to Nek5000 used to obtain the results in this chapter are *ANSYS ICEM* and *python*. For post-processing *Visit* and *Matlab* were used.

6.1 Case 1: Gas dispersion in a simplified urban area

The scenario investigated in this case is dispersion of a neutral gas in a rectangular tunnel with four cubic blocks placed as obstacles. The blocks have sides $h = 0.109$m and represent a set of buildings forming a street canyon. The gas is released from a circular source on ground level and is translated by the wind field through the canyon, see Figure 6.1. In this figure h has been used as the length scale. The dotted lines indicate the positions where data is collected.

Scaling the entire domain with the size of the boundary layer $H = 1$m restricts it to the box $0.0 \leq x/H \leq 4.96, -1.75 \leq y/H \leq 1.75, 0 \leq z/H \leq 1.5$. The four cubic boxes are centered around $(1.4315, 0)$ with a distance h between each box. The source is placed with its center in $(0.396, 0)$ and has a radius $r = 0.0515$. The grid used for the computations consists of 14747 elements and a polynomial degree of 7, this makes the total number of nodes $N \approx 5.2$mill. There are no curved elements in this case, so the circular source is created simply by requiring that the inflow velocity is non-zero only for the nodes on the floor within that particular circle. The corresponding area is

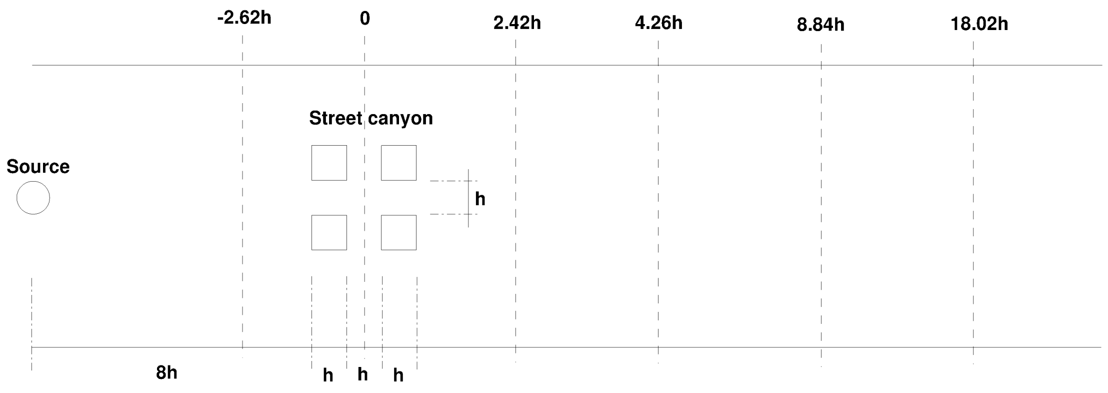

FIGURE 6.1: *Schematic overview of the domain from above. The data is collected along the dotted lines.*

somewhat different from that of a circle $\approx 5\%$, but this is scaled correctly during the post-processing.

The simulations are performed using Large Eddy Simulation (LES) with the dynamic Smagorinsky-Lilly subgrid-scale model and by applying the polynomial filtering routine that is available in Nek5000. The release of gas will result in a plume that is advected with the wind field, see Figure 6.3. The concentration of the released gas at the indicated positions in Figure 6.1 are compared with experimental wind-tunnel data and simulations performed in CDP [32]. For clarification some of the variables repeatedly mentioned in this section are stated explicitly in Table 6.1.

Variable	value	unit	commentary
H	1	m	length scale of the domain
h	0.109	m	the sides of the cubic boxes
Q	50	$\mathrm{dm}^3/\mathrm{min}$	gas release from source
U_{ref}	≈ 1.08	m/s	reference value of U

TABLE 6.1: *Essential variables, U_{ref} is calculated as a time average of the velocity in x-direction at a point far away from the floor and walls and will therefore vary by a small amount from case to case.*

The inflow condition at $x = 0$ has to be mapped onto the boundary at each time step. To mimic the situation in the wind-tunnel the velocity field on the inflow was generated in a different simulation performed in CDP. The inflow velocity was written to file every 0.0013s for a total of 28s and had to be interpolated onto the domain for the simulations in Nek5000 since the grid was not identical. The right plot in Figure 6.2

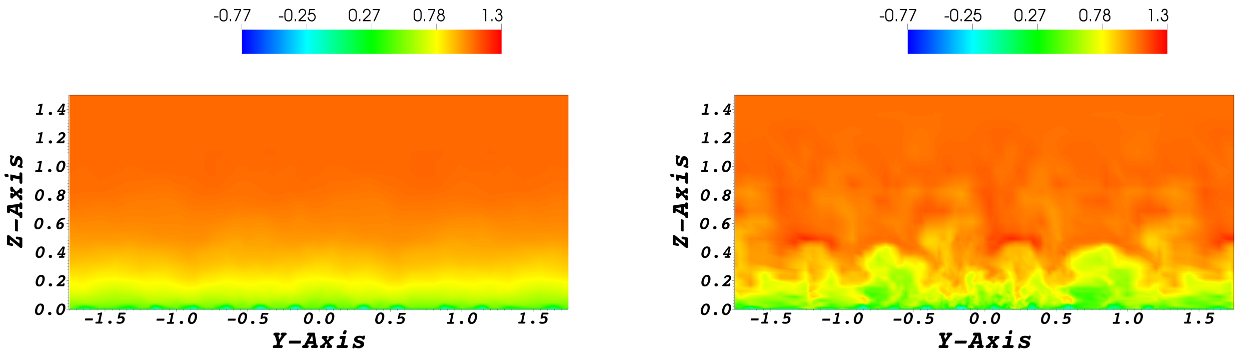

FIGURE 6.2: *The averaged (left) and instantaneous (right) x-velocity on the inflow boundary.*

is an instantaneous picture of the inflow velocity in x-direction, notice how the pattern repeats itself along the y-axis. This is because the inflow data was generated in a smaller channel, approximately 1/3 of the width of the computational domain used for the data sampling. An interpolation algorithm implemented at FFI was applied to adjust the inflow data to the computational mesh, this was done directly in `.usr`.

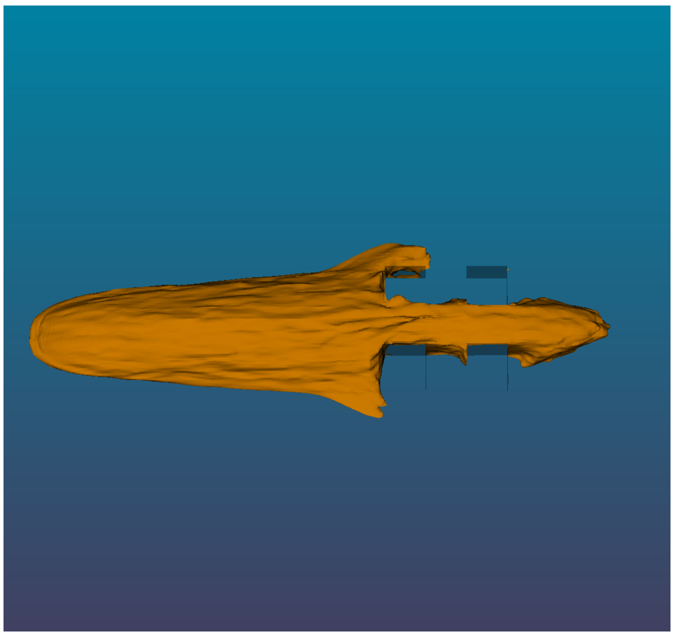

FIGURE 6.3: *An iso-surface of the average concentration with* $C = 0.03$ *after 22 seconds of sampling. Streamwise direction is from left to right.*

The simulations in Nek5000 were performed in the following manner; first 6 seconds of flow-through to initialize the velocity field in the channel, followed by 8 seconds of gas release to initialize the gas concentration. After assuring that the wind-field was correctly created and the released gas had reached the measurement lines furthest from the source, the data sampling of 22 seconds started.

The mesh used in the simulations performed in Nek5000 and the one performed in CDP are different, and the resolution in the part of the domain close to the cubes is described in Table 6.2. It is evident that the resolution is better in the simulations done in CDP and especially in the z-direction.

Solver	n_x	n_y	n_z
CDP	28	28	64
Nek5000	22	22	36

TABLE 6.2: *Number of nodes used to represent one cube.*

6.1.1 Results - Gas dispersion

This case is a part of a larger project designed to evaluate different solvers ability to perform simulations of gas dispersion. The N-S equations are solved using the $P_N P_N$ formulation with the fractional step method, IOFS with a target CFL-number equal 2.0 was enabled to maximize the time step as recommended in [25]. It should be mentioned that when activating the SGS model and deactivating the filter the simulation is more probable to blow up. This effect is captured in Figure 6.4 that shows how the Smagorinsky model does not damp spurious velocity modes in the same degree as the filter.

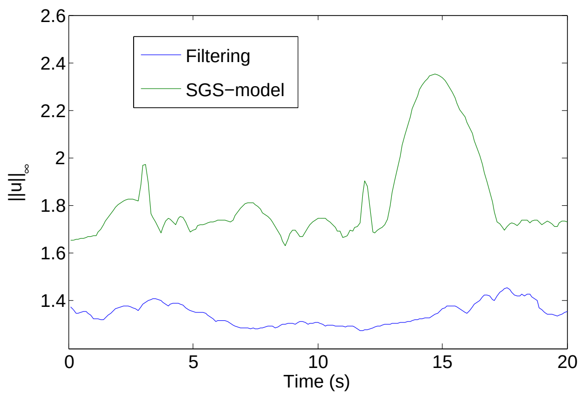

FIGURE 6.4: $||\mathbf{u}||_\infty$ *as a function of time. The green line represents the simulation with the dynamic Smagorinsky SGS model and the blue line represents the filtering with* $\alpha = 0.05$ *and a quadratic decay on the last 3 modes.*

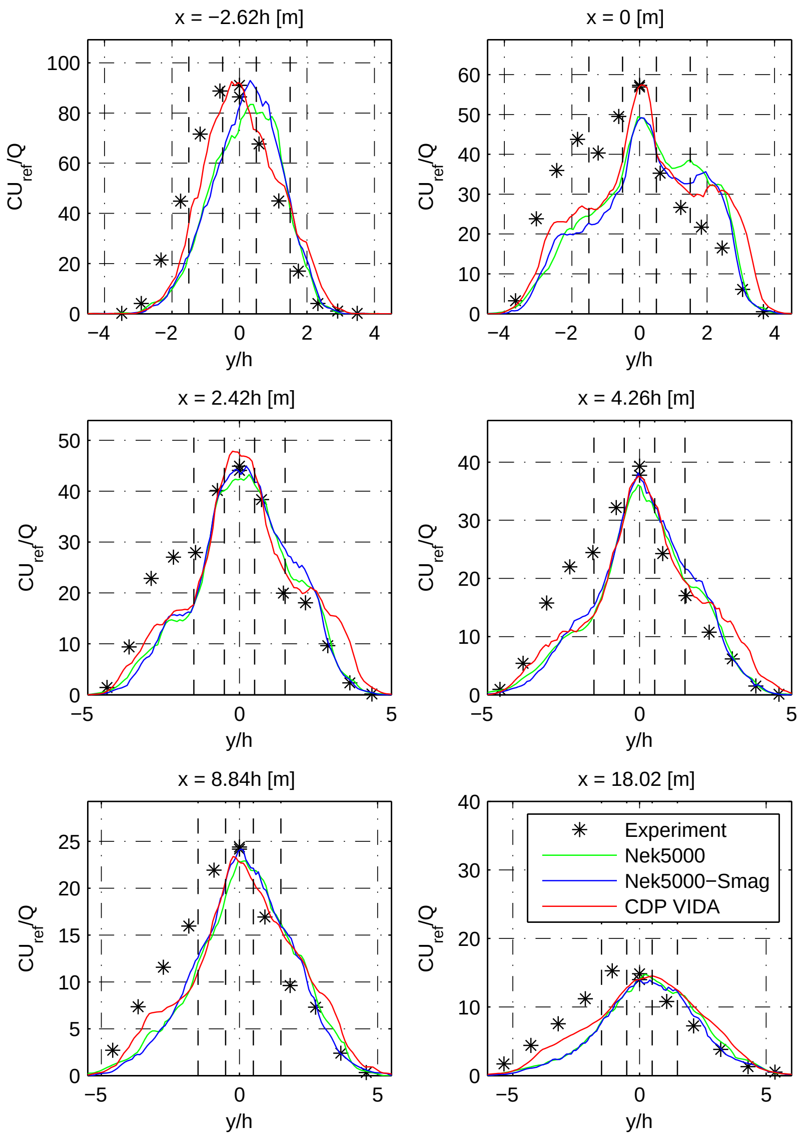

FIGURE 6.5: *Time-averaged concentration with a sample time of* 22.00*s at* $z/H = 0.025$ *plotted and scaled with the free-stream velocity and the emission rate. Compared with wind-tunnel data and results from CDP. Two dashed lines on either side of the centerline represent the canyon.*

Figure 6.5 shows the scaled concentration along the dotted lines in Figure 6.1. According to this figure Nek5000 does indeed capture the important features of the mean concentration. At the two first measurement lines the results are slightly skewed to the right, this is to some degree also the case for the CDP simulations but not for the experiment. A possible explanation could be that the inflow condition favours one of the sides of the domain, or simply that the sampling time is not sufficiently long.

The results also indicate that the difference between the SGS model and the filtering is not that large, if anything the SGS model shows a tendency to estimate higher concentration peaks. In particular the first plot indicates a significant difference. An important difference between the filtering and the SGS model is that the filter works based on the current state of the flow whereas the amount of diffusion added by the SGS model is strongly biased by the previous states of the flow. This could lead to local observations of either too much or too little smoothing.

The concentration along the vertical measurement lines is plotted in Figure 6.6 and overall Nek5000 provides good results according to the reference solutions. The largest difference is found close to the wall right in the middle of the cubes. In particular the simulation with filtering underestimates the concentration in this domain. The resolution of the mesh used for the Nek5000 simulations in this area is notably worse than for the CDP-simulations. And in the middle of the cubes neither one of the filter or the Dynamic Smagorinsky model are able to correct this. The $P_N P_N$ formulation is known to produce splitting errors of significant sizes close to the wall, and could play an important role in this part of the domain.

The simulations were also performed using the $P_N P_{N-2}$ formulation with filtering, and the results were similar to those observed above. The SGS model did however not function at all for the $P_N P_{N-2}$ implementation, and resulted in a system crash in one of the earliest time steps. It was beyond the scope of this work to correct this presumable bug in the Nek5000 implementation.

It was also experimented with different meshes. An attempt was made to improve the results at the measurement lines in between the cubes by increasing the resolution in this particular area, this was done by using the refinement functionality in ICEM. The rest of the mesh was made coarser to keep the amount of elements at approximately the same level. The amount of nodes used to describe one cube as presented in Table 6.2 were for this simulation $(n_x, n_y, n_z) = (21, 21, 63)$. The simulation was performed using $P_N P_N$ with filtering, but the results were similar to the ones previously obtained. It could seem like a refinement of the mesh in the x and y direction could be necessary to achieve better results.

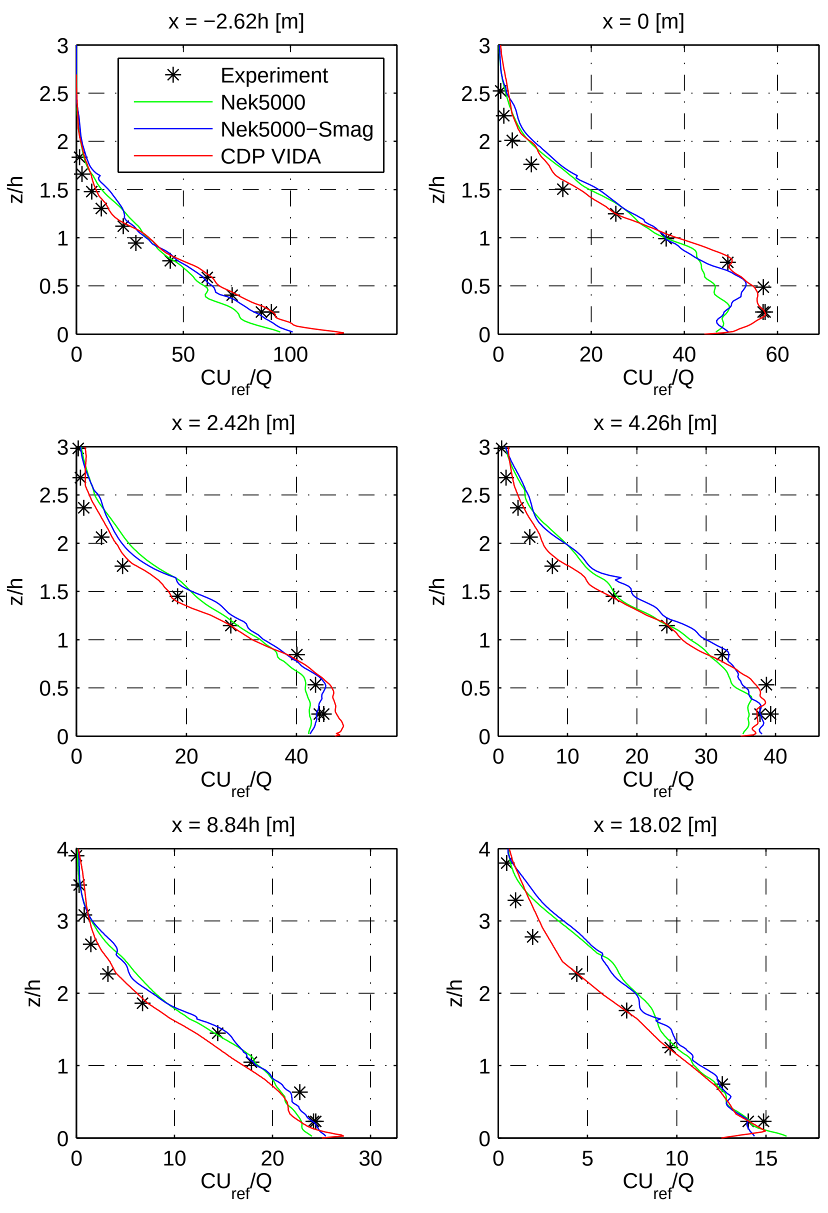

FIGURE 6.6: *Time-averaged concentration with a sample time of* 22.00s *at* $y = 0$ *plotted and scaled with the free-stream velocity and the emission rate. Compared with wind-tunnel data and results from CDP.*

As for the performance of Nek5000 the results are baffling. With the same number of processors, same accuracy criteria, and approximately the same number of nodes in the grid, Nek5000 simulated one second of flow more than five times as fast as a numerical code comparable to CDP! This was done without any particular tuning and with a time step about 2/3 of the one used in CDP.

To check how Nek5000 scales for the number of processors, one second of flow was simulated using the $P_N P_{N-2}$ formulation on the grid with 14747 elements, $N = 7$, $DT = 0.001$ and both IOFS and filtering was activated. The results are presented in Figure 6.7.

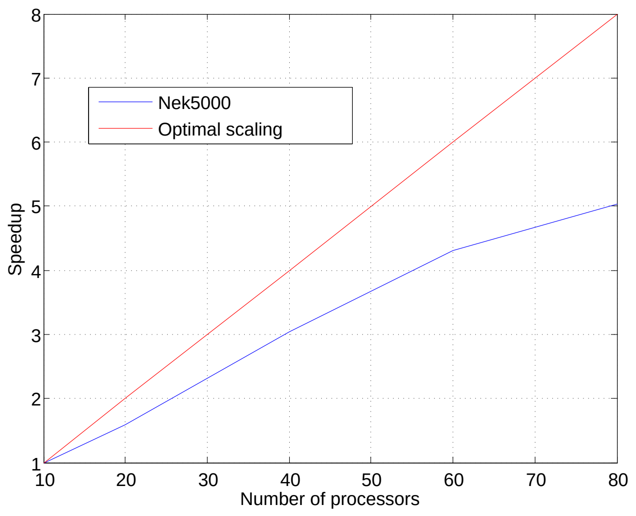

FIGURE 6.7: *Scaling results, from case 1 with* 14747 *elements,* $P_N P_{N-2}$ *formulation using filtering and* $N = 7$.

This is far from optimal speedup which is the theoretical upper limit, and Nek5000 were perhaps expected to perform better than these results indicate. This performance test is done without doing any tweaking of the settings, and it is probable that by adjusting some parameters in `.rea` a better speedup could be obtained.

6.2 Case 2: Drag and lift on a cylinder

A standard benchmark case for flow solvers is presented in [33]. The case is to calculate the drag and lift coefficients on a cylinder in a rectangular channel. The setup for the domain and boundary conditions are given in Figure 6.8. The constants applied in the description of the geometry and the coefficient scales are listed in table Table 6.3. Finding the drag and lift coefficient requires a calculation of the velocity field around the cylinder which is done by solving the unsteady N-S equations until a steady flow is

Constant	Value	Property
H	0.41m	Width and height for the channel
D	0.1m	Diameter of the cylinder and length scale
U	0.2m/s	Velocity scale
ν	10^{-3}m^2/s	Kinematic viscosity of the fluid
Re	20	Reynolds number

TABLE 6.3: *Constants for case 2*

reached. This implies that the spatial accuracy will dominate the error and one would expect great results in Nek5000 due to its spectral convergence rate.

The flow is laminar with Reynolds number $Re = 20$ so all the challenges arising when dealing with turbulent flow does not come to play in this case. The drag and lift forces

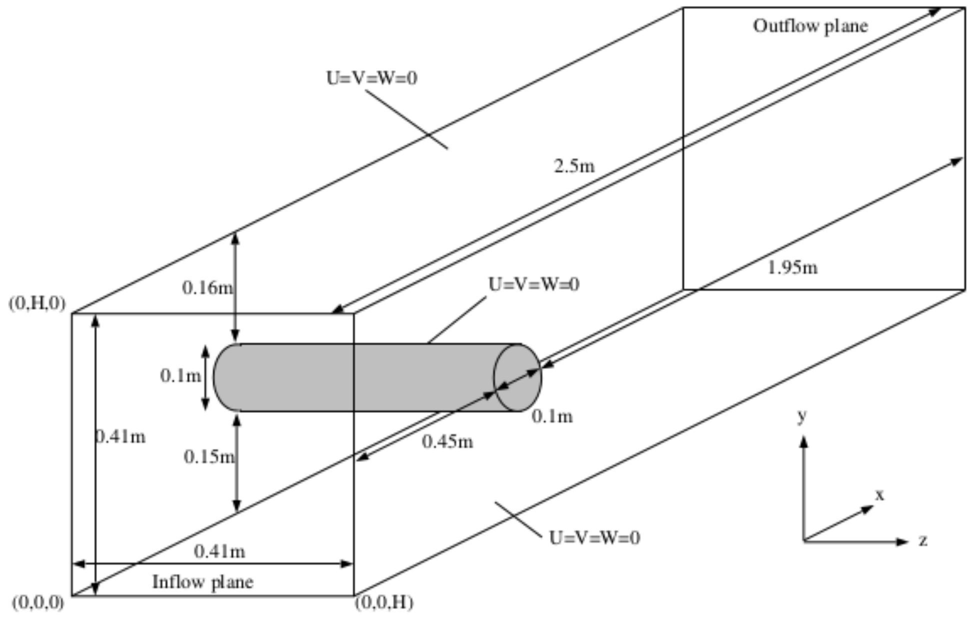

FIGURE 6.8: *Computational domain and boundary conditions.*

on a surface S are given as

$$F_D = \int_S (\rho\nu \frac{\partial v_t}{\partial n} n_y - p n_x) dS, \qquad F_L = -\int_S (\rho\nu \frac{\partial v_t}{\partial n} n_x + p n_y) dS. \tag{6.1}$$

v_t is the tangential velocity, $\mathbf{n} = [n_x, n_y, 0]$ is the unit vector normal to the surface S and the tangent velocity vector is defined as $\mathbf{t} = [n_y, -n_x, 0]$.

Surface integrals in Nek5000 are solved numerically, $\int_S f dS = \sum f_i A_i$, where f is some function and A_i is the quadrature weight to the ith GLL-node for the surface integral over face S. A_i corresponds to a two dimensional mass matrix in Nek5000 available for all elements and all faces in the array `area(lx1,lz1,6,lelt)`.

The coefficients corresponding to these forces known as the drag and lift coefficients are given by the formulas

$$c_D = \frac{2F_D}{\rho U^2 DH}, \qquad c_L = \frac{2F_L}{\rho U^2 DH}. \tag{6.2}$$

Nek5000 provides functions for calculating lift and drag on any user-specified object. The function is called `drag_calc(scale)`, with the input parameter defined by the user, for this case `scale` $= 2/(\rho U^2 DH)$. Apart from this the function `set_obj()` has to be modified to create an object that consists of pointers to all the faces on the cylinder. The mesh around the cylinder is illustrated in Figure 6.9. Initially this case was solved

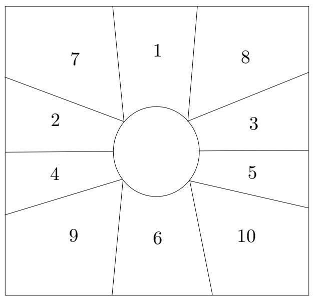

FIGURE 6.9: *Initial mesh around cylinder.*

using a second degree polynomial to describe the circle segments corresponding to each element. With the implementation of the new routine as described in Chapter 5.1 the circle segments could be represented with the same order as the polynomials used for the velocity. The importance of the error resulting from the second degree approximation of the circle segments is presented in Chapter 6.

An additional test performed on this case was how different settings in Nek5000 affect the estimation of the drag and lift coefficients. Perhaps most curious is whether the $P_N P_N$ or $P_N P_{N-2}$ formulation is applied. Note that the pressure in the latter formulation is not defined on the boundary of the cylinder and does therefore need to be extrapolated onto the surface in order for the integral to be calculated. It is however more likely that this effect will be negligible compared with the splitting error implied by both schemes, since they both impose an erroneous boundary condition on the pressure.

6.2.1 Results - benchmark comparison

The results are presented in Table 6.4, and they confirm that the treatment of the geometry is essential, both coefficients are computed with significantly better accuracy. Compared with the results from the other softwares applied in [33] Nek5000 performs

# of Cells	Software	c_D	c_L	%**Err** c_D	%**Err** c_L
2124030	Nek5000 (mid)	6.18349	0.008939	0.030	4.19
2124030	Nek5000 (arc)	6.18498	0.009413	0.005	0.10
3145728	CFX	6.18287	0.009387	0.04	0.15
3145728	OpenFOAM	6.18931	0.00973	0.06	3.5
3145728	FEATFLOW	6.18465	0.009397	0.01	0.05

Table 6.4: *Results for the drag and lift coefficients with reference values* $c_D = 6.18533$ *and* $c_L = 0.009401$. $P = 11$ *for the simulations in Nek5000.*

just as well or better in most cases. It should be mentioned that the division of the grid is created in a different manner for Nek5000 so the comparison is not as direct as it may seem from the table.

6.2.2 Results - internal adjustments

As discussed in Chapter 4 there are many adjustments available in Nek5000. To understand the actual effect on the results, several different settings were investigated for this case and the results are presented in Table 6.5. The spectral convergence is also confirmed in Figure 6.10 by calculating the lift coefficient error for increasing polynomial degree.

The setting that has the biggest impact on the result is the $P_N P_N$ scheme which clearly performs worse than the others. It would seem like the error on the boundary known to be present in both schemes has a bigger effect in the $P_N P_N$ formulation. Use of the IOFS method was expected to have a negative effect on the accuracy as stated in [23], this is however not observed for this case. The filtering does have a negative effect on calculation of the lift coefficient as expected, but it is of relatively small significance which confirms the analytical results from Eq. (3.33).

Be aware that these results are obtained from a laminar test case and does not in any way suggest any optimal adjustment for Nek5000. It is however important to be aware that the activation of de-aliasing does not guarantee a better result. For a well-resolved flow

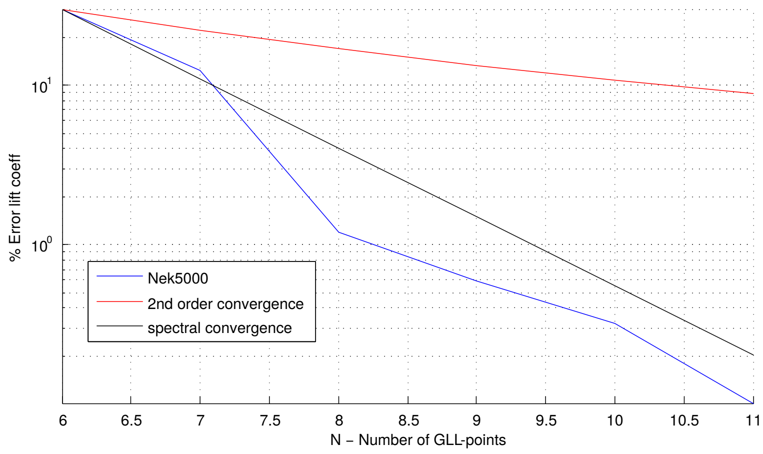

FIGURE 6.10: *The logarithm of the error plotted against the polynomial degree. All results are with $P_N P_{N-2}$ and de-aliasing, and they are solved without using the characteristic scheme or any filtering. Two lines illustrating a second order convergence and spectral convergence is plotted to illustrate the convergence rate obtained in Nek5000.*

situation such as $N = 11$ for this case, the effect of aliasing is negligible and applying a higher order quadrature to resolve the non-linear term is a waste of computational time. For $N = 7$ the results show that activation of de-aliasing provides a better estimate, so the decision to activate de-aliasing or not depends on the type of flow and the resolution of the mesh. The fact that the aliased solution for $N = 11$ performs better than the rest, is probably because of the accuracy of the reference solution.

	Settings					% Error	
#	N	ifsplit	Dealiasing	IOFS	Filter	c_D	c_L
1	11	No	Yes	No	No	0.005	0.10
2	11	No	No	No	No	0.005	0.03
3	11	No	Yes	No	Yes	0.006	0.31
4	11	No	Yes	Yes	No	0.005	0.11
5	11	Yes	Yes	No	No	0.012	2.35
6	7	No	Yes	No	No	0.002	12.24
7	7	No	No	No	No	0.013	25.04

TABLE 6.5: *Test of solver settings in Nek5000.*

When designing the mesh it is important to be aware of how the computational time depends on the polynomial degree. In user manual for Nek [25] it states that the computational time for one time step is of order $\mathcal{O}(EN^4) = \mathcal{O}(EP^4)$. This is confirmed in the left plot in Figure 6.11. Another important consequence of increasing the polynomial degree is that the time step decreases with a factor $\mathcal{O}(P^2)$ due to the CFL-condition seen in Eq. (3.48). The computational time per time unit is therefore expected to be $\mathcal{O}(EP^6)$ which is confirmed in the right plot in Figure 6.11.

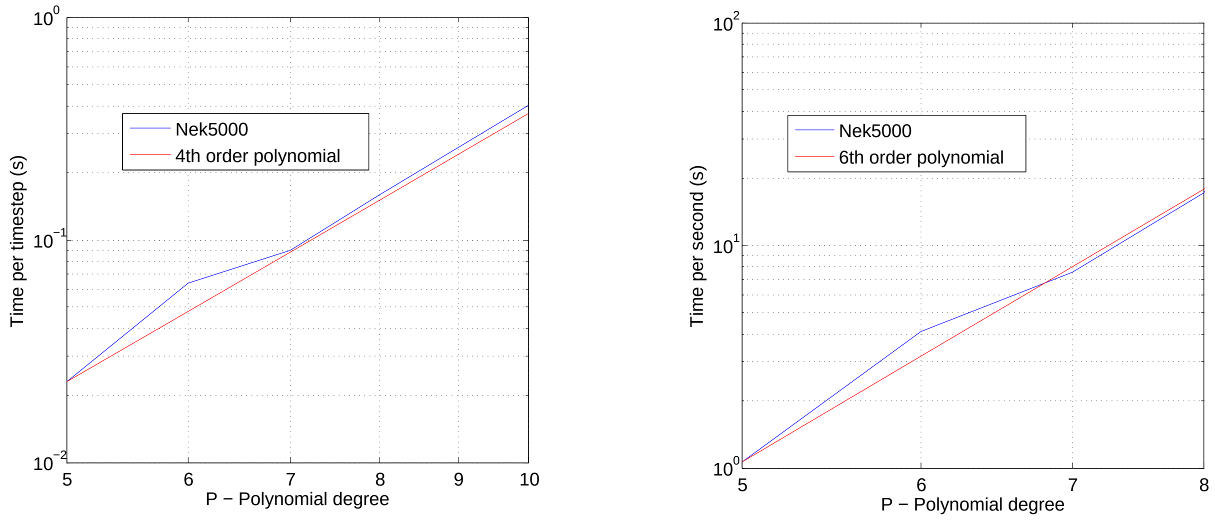

FIGURE 6.11: *Computational time for one time step and for one second in Nek5000 as a function of the polynomial degree P.*

Chapter 7

Concluding remarks

Nek5000 has proven to give accurate results with a relatively coarse mesh as presented in Chapter 6. This is as expected since it is based on a higher order method known to yield great accuracy. As for the performance it is no doubt that the effort put into the efficiency of the code has paid off. The possibility to obtain results 5-10 times faster than similar softwares is an important factor to keep in mind when choosing which software to use. The polynomial degree is chosen by changing a single parameter, which makes performance tests and accuracy adjustments simple to do.

Since the mathematical formulation in Nek5000 is based on a tensor product of the basis functions in each direction it is limited to the use of hex-mesh. This is no problem for the geometries studied in this thesis, but for complex geometry the use of tetrahedral mesh is mandatory.

Another aspect worth noticing is the filtering procedure which has some similarities to variational multiscale as presented in Chapter 3.4.2.1. It would be interesting to design some test cases to further investigate its properties.

The fact that Nek5000 is an open-source code is also a huge advantage to other black-box solvers. Although the user community is not that large there are several committed users that provide their help on short notice. Many different examples are available, and the user guide contains a nice introduction with everything needed to get the program up and running. The documentation of variables and functions are however sometimes missing, which was the main motivation behind the creation of Appendix B.

7.1 Further work

The surface projection could be expanded and improved in many ways. Making it iterative such that the points are projected more controlled towards the surface is one possibility. Experimentation with the weighting function and the distance norm should be done to see what provides best results. Treating the points along the edges differently, since they have a larger risk of being affected by the neighbouring element. Omitting the projection for certain points, especially those on edges that limits to other boundary conditions. Restricting the relative or absolute change to avoid too much disturbance in the original mesh. These are just some ideas that might be worth having in mind if applying or expanding this method.

For the simple array case it was experimented with different meshes and Nek5000 seemed to have no problems with refined grids from ICEM. The refinement functionality does however not provide the same flexibility to adjust the number of elements. Perhaps even better results could have been achieved by making an o-grid around the most important part of the domain, say for instance the box, $x/H < 3$, $|y|/H < 0.4$, $z/H < 0.15$ the nodes could have been distributed in a more economical fashion and an overall better result could have been achieved.

Finding out why the dynamic Smagorinsky model did not work for the $P_N P_{N-2}$ is also something that should be investigated further. As well as experimentation with other LES-models.

The effect of the filter is also something that could be studied further, perhaps it would be possible to support the choice of alpha based on some analytical result.

Appendix A

Properties of Legendre polynomials

The Legendre polynomials are a group of orthogonal polynomials on $(-1, 1)$ satisfying the recurrence relation

$$(n+1)L_{n+1}(x) = (2n+1)xL_n(x) - nL_{n-1}(x), \tag{A.1}$$

where $L_0(x) = 1$ and $L_1(x) = x$. Some other useful properties are the L_2-norm and the derivatives which can be found in any texbook, for instance [12],

$$\begin{aligned} ||L_n(x)||^2 &= \int_{-1}^{1} L_n(x)dx = \frac{2}{2n+1}, \\ L'_{n+1}(x) &= (2n+1)L_n(x) + L'_{n-1}(x). \end{aligned} \tag{A.2}$$

By continuing the recursive form of the second line in Eq. (A.2) one obtains

$$\begin{aligned} L'_{n-1}(x) &= (2(n-2)+1)L_{n-2}(x) + L'_{n-3}(x), \\ L'_{n-3}(x) &= (2(n-4)+1)L_{n-4}(x) + L'_{n-5}(x), \\ &\vdots \end{aligned} \tag{A.3}$$

Which ultimately enables an expression for the derivative based on the lower order Legendre polynomials, explicitly given as

$$\begin{aligned} L'_{n+1}(x) &= (2n+1)L_n(x) \\ &\quad + (2(n-2)+1)L_{n-2}(x) \\ &\quad + (2(n-4)+1)L_{n-4}(x) + \cdots \end{aligned} \tag{A.4}$$

Notice that the coefficients in front of each polynomial is 2 times the inverse norm of the same polynomial squared, allowing the expression to be simplified to

$$\begin{aligned} L'_{n+1}(x) &= 2||L_n||^{-2}L_n(x) \\ &\quad + 2||L_{n-2}||^{-2}L_{n-2}(x) \\ &\quad + 2||L_{n-4}||^{-2}L_{n-4}(x) + \cdots \end{aligned} \tag{A.5}$$

In this thesis the second order derivatives will be used and by the result in Eq. (A.4) they will be given as

$$\begin{aligned} L''_{n+1}(x) &= 2||L_n||^{-2}L'_n(x) \\ &\quad + 2||L_{n-2}||^{-2}L'_{n-2}(x) \\ &\quad + 2||L_{n-4}||^{-2}L'_{n-4}(x) + \cdots \end{aligned} \tag{A.6}$$

Applying the same recurrence relations from Eq. (A.4) to the rhs. derivatives and assembling the terms allows the second derivative of the Legendre polynomial to be written as

$$\begin{aligned} \frac{1}{4}L''_{n+1}(x) &= ||L_n||^{-2}||L_{n-1}||^{-2}L_{n-1}(x) \\ &\quad + \left(||L_n||^{-2} + ||L_{n-2}||^{-2}\right)||L_{n-3}||^{-2}L_{n-3}(x) \\ &\quad + \left(||L_n||^{-2} + ||L_{n-2}||^{-2} + ||L_{n-4}||^{-2}\right)||L_{n-5}||^{-2}L_{n-5}(x)\cdots \end{aligned} \tag{A.7}$$

This can be written in a compact form as

$$L''_{n+1}(x) = \sum_{j=1}^{\lceil n/2 \rceil} \Lambda_{n+1,n+1-2j}L_{n+1-2j}(x), \tag{A.8}$$

$$\Lambda_{n+1,n+1-2j} = 4||L_{n+1-2j}||^{-2}\sum_{k=0}^{j-1}||L_{n-2k}||^{-2}. \tag{A.9}$$

The sum $\Lambda_{i,j}$ can be considered as the coefficient describing the amount of L_i'' which is described by L_j. For the sake of clarity Λ is restated here with indices i, j

$$\Lambda_{i,j} = 4||L_j||^{-2} \sum_{k=0}^{(i-j)/2-1} ||L_{i-1-2k}||^{-2}. \tag{A.10}$$

Appendix B

Variables and Functions in Nek5000

B.1 Variables

The Nek manual provides information on many of the variables given in the .rea and SIZE file. It is however no list of useful variables defined in other files. In Table B.1 some of the variables that have been frequently used in .usr subroutines are listed. Note that all the variables listed in this table are defined outside of SIZE, .rea and .usr

B.2 Functions

B.2.1 standard calculations found in math.f or navier1.f

nekasgn(ix,iy,iz,ie) Assigns the coordinates of node `(ix,iy,iz)` in element ie to the common variables `x,y,z`.

facind(kx1,kx2,ky1,ky2,kz1,kz2,nx1,ny1,nz1,f) Assigns the index limits of a face `f` with `nx1,ny1,nz1` points in each spatial direction.

zwgll(zg,wg,nx1) Get the nx1 GLL-points and weights to `zg` and `wg`.

cadd(zg,c,nx1) Adding a constant `c` to a vector `zg` of length `nx1`.

AVG	
uavg(ax1,ay1,az1,lelt)	Averaged values of u, similar for v,w,p
urms(ax1,ay1,az1,lelt)	Variance of u, similar for v,w,p
vwms(ax1,ay1,az1,lelt)	Covariance of vw
tavg(ax1,ay1,az1,lelt,ldimt)	Averaged values of t and all passive scalars
GEOM	
xm1(lx1,ly1,lz1,lelt)	X-coordinates for the velocity mesh
xm2(lx2,ly2,lz2,lelv)	X-coordinates for the pressure mesh
unx(lx1,lz1,6,lelt)	Surface normals
area(lx1,lz1,6,lelt)	Surface integral weights
INPUT	
cbc(6,lelt,0:ldimt1)	Boundary conditions of each face
ccurve(12,lelt)	Curved side character
curve(12,6,lelt)	Curved side information
PARALLEL	
lglel(lelt)	Mapping from local to global element index
gllel(lelg)	Mapping from global to local element index
SOLN	
vx(lx1,ly1,lz1,lelv)	X-velocity
t(lx1,ly1,lz1,lelv,ldimt)	Temperature and passive scalars
vtrans(lx1,ly1,lz1,lelt,ldimt1)	Diffusive constant to additional scalars
vdiff(lx1,ly1,lz1,lelt,ldimt1)	Convective constants to additional scalars
TSTEP	
istep	Current iteration step
iostep	Output step frequency
time	Current time
tstep	Current time step
dt	Time step
dtlag(10)	The previous 10 time steps
bd(10)	Max 10 backward difference coeffs
ab(10)	Max 10 extrapolation coeffs (Adam-Bashforth)
WZ	
zgm1(lx1,3)	GLL points for x,y and z directions
OTHER	
x,y,z	Local coordinates assigned by nekasgn()
ux,uy,uz	Local velocities assigned by nekasgn()
temp	Local temperature assigned by nekasgn()
nio	Processor node number
ndim	Number of dimensions
nelv	Number of elements for velocity mesh
nelt	Number of elements for the t-mesh
pm1 (lx1,ly1,lz1,lelv)	Pressure mapped to mesh 1

TABLE B.1: *useful variables in Nek, the bold capital sections denote the separate files in /trunk/nek/.*

cmult(zg,c,nx1) Multiplying every element of vector `zg` of length `nx1` with `c`.

chsign(wrk,nx1) change the sign of every element in vector wrk of length `nx1`.

cfill(zg,c,nx1) Fill vector `zg` of length `nx1` with the constant `c`.

rzero(zg,nx1) Fill vector `zg` of length `nx1` with zeroes.

rcopy(zg,zg2,nx1) copy all elements from vector `zg2` to vector `zg`, both of length `nx1`.

B.2.2 Functions regarding mesh and distribution of GLL-points

gh_face_extend(x,zg,n,type,ie,v) The Gordon hall algorithm described in chapter 3, the type variable denotes whether the algorithm should use vertices, edges or faces to distribute the inner GLL-points.

xyzlin(xl,yl,zl,nxl,nxl,nxl,ie,ifaxl) Generate bi- or trilinear mesh.

fix_geom() Routine for assuring mesh consistency and re distributing the internal GLL-points correctly on the updated geometry.

B.2.3 Additional auxiliary functions implemented for this thesis

readsurfpro(wrksurf,iwrkelem) Read the coordinates from `surf.i` to `wrksurf` and add the elements and faces in `bdry.i` to `iwrkelem`.

getfaceinfo(mid,radius,wrk,ie,f) Estimate the midpoint `mid` and `radius` of the face `f` in element `ie`.

createworkingarray(wrkbdry,wrksurf,mid,iter,beta,rad,f) updating `wrkbdry`, Get the points in `wrksurf` within a distance `rad` from the point `mid`.

fix_gll(ie,f) Redistribute the GLL-points between face `f` and the opposite face in element `ie` to make sure that all points lie within the element.

getface(kx1,kx2,ky1,ky2,kz1,kz2,wrk,n,ie) assigning the values of the face in element `ie` corresponding to the index limits `kx1,kx2`... to the array `wrk(n,n,3)` .

getsurfnorm(sn,ix,iy,iz,f,ie) Providing the surface normal `sn` at point `ix,iy,iz` of element `ie` and face `f`.

calcerror(error,lambda,sn,wrk,radius) calculate the distance `error` from the initial GLL-point to a given point on the surface.

interp_up(iinterp,rinterp,n,error,k) Update interpolation points, validate if the new point `k` with distance `error` from the current point is closer than the existing interpolation points in `iinterp`.

set_new_pt(iinterp,rinterp,ix,iy,iz,ie) defining the position of the new GLL-point on the surface.

getlimits(k,n,kx1,kx2,ky1,ky2,kz1,kz2) Get the index limits `kx1,kx2`... corresponding to edge `k` with `n` GLL-points.

setcoords(xq,yq,zq,xedge,yedge,zedge,nxl,k) copy the updated edges `xedge` to the initial edges `xq`.

getcoords(xq,yq,zq,xedge,yedge,zedge,nxl,k) copy the node information from the initial edge `xq` to `xedge`.

checkcorner(is,ix,iy,iz,kz1,kz2,ky1,ky2,kx1,kx2) Check if the current GLL-point is a corner.

addcorners(wrkbdry,iter,ie,f) Add the corners of the element to `wrkbdry` so that they can be used to do the projection of neighbouring points.

B.2.4 Additional functions for mshconvert

This section contains the additional functions written to supplement the script `mshconvert.py`. In addition to the functions listed here, minor extensions were also done in the already existing functions.

getreastart(name) Use the first section of the file `name.rea` as a template for the new `.rea` file.

getreaend(name) Use the last section of the file `name.rea` as a template for the new `.rea` file.

fixbc(name) Fix lower case inflow boundary conditions.

fixthermalbc(name,iftemp) Include thermal boundary conditions if the flag `temperature=true`.

points2circ(x1,x2,x3) Find the radius and midpoint of the circle correponding to 3 points on an arc.

write_surface_file() Write the file `surf.i`.

write_surf_list() Write the file `bdry.i`.

fixSIZE() Add or change the variables that describes the sizes of `surf.i` and `bdry.i` in the local `SIZE` file.

Bibliography

[1] F. M. White. *Viscous fluid flow.* McGraw-Hill series in mechanical engineering. McGraw-Hill, New York, 1991. ISBN 0-07-069712-4. URL http://opac.inria.fr/record=b1096847.

[2] D. Carati, G. S. Winckelmans, and H. Jeanmart. On the modelling of the subgrid-scale and filtered-scale stress tensors in large-eddy simulation. *Journal of Fluid Mechanics*, 441:119–138, 8 2001. ISSN 1469-7645. doi: 10.1017/S0022112001004773. URL http://journals.cambridge.org/article_S0022112001004773.

[3] A. Leonard. Energy cascade in large eddy simulations of turbulent fluid flows. *Adv. Geophys.*, 18A:237–248, 1974.

[4] S. B. Pope. *Turbulent Flows.* Cambridge University Press, 2000. ISBN 9780511840531. URL http://dx.doi.org/10.1017/CBO9780511840531. Cambridge Books Online.

[5] D. K. Lilly. The representation of small scale turbulence in numerical simulation experiments. In *IBM Scientific Computing Symposium on environmental sciences*, pages 195–210, Yorktown heights, 1967.

[6] M. Germano, U. Piomelli, P. Moin, and W. H. Cabot. A dynamic subgrid-scale eddy viscosity model. *Phys. Fluids A*, 3:1760–1765, 1991.

[7] G. Karniadakis and S. Sherwin. *Spectral/hp element methods for computational fluid dynamics.* Oxford University Press, 2013.

[8] A. Quarteroni. *Numerical Models for Differential Problems, 2. edition.* Springer, 2014.

[9] I. Babuska. The finite element method with Lagrangian multipliers. *Numerische Mathematik*, 20:179–192, 1972/73. URL http://eudml.org/doc/132183.

[10] F. Brezzi. On the existence, uniqueness and approximation of saddle-point problems arising from Lagrangian multipliers. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 8(R2):129–151, 1974. URL http://eudml.org/doc/193255.

[11] J. L. Guermond. Nonstandard nonconforming approximation of the Stokes problem, i: Periodic boundary conditions.

[12] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral methods : evolution to complex geometries and applications to fluid dynamics*. Scientific computation. Springer, Berlin, 2007. ISBN 978-3-540-30727-3. URL http://opac.inria.fr/record=b1127560.

[13] E. Süli and D. F. Mayers. *An Introduction to Numerical Analysis*. Cambridge, 2006.

[14] C. Canuto, M.Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods: Fundamentals in Single Domains*. Scientific Computation. Springer Berlin Heidelberg, 2007. ISBN 9783540307266. URL https://books.google.no/books?id=DFJB0kiq0CQC.

[15] Y. Maday and A. T. Patera. Spectral element methods for the incompressible Navier-Stokes equations. In *IN: State-of-the-art surveys on computational mechanics (A90-47176 21-64). New York, American Society of Mechanical Engineers, 1989, p. 71-143. Research supported by DARPA.*, volume 1, pages 71–143, 1989.

[16] A. T. Patera. A spectral element method for fluid dynamics: Laminar flow in a channel expansion. *Journal of Computational Physics*, 54(3):468–488, Jun 1984. ISSN 0021-9991. doi: 10.1016/0021-9991(84)90128-1. URL http://dx.doi.org/10.1016/0021-9991(84)90128-1.

[17] K. Z. Korczak and A. T. Patera. An isoparametric spectral element method for solution of the Navier-Stokes equations in complex geometry. *Journal of Computational Physics*, 62(2):361–382, Feb 1986. ISSN 0021-9991. doi: 10.1016/0021-9991(86)90134-8. URL http://dx.doi.org/10.1016/0021-9991(86)90134-8.

[18] Y. Maday, A. T. Patera, and E. M. Ronquist. *A well-posed optimal spectral element approximation for the Stokes problem.* National Aeronautics and Space Administration, Langley Research Center, 1987.

[19] P. F. Fischer and J. Mullen. Filter-based stabilization of spectral element methods. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 332(3):265 – 270, 2001. ISSN 0764-4442. doi: http://dx.doi.org/10.1016/S0764-4442(00)01763-8. URL http://www.sciencedirect.com/science/article/pii/S0764444200017638.

[20] R. Pasquetti and C.J. Xu. Comments on "Filter-based stabilization of spectral element methods". *Journal of Computational Physics*, 182(2):646 – 650, 2002. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.2002.7178. URL http://www.sciencedirect.com/science/article/pii/S0021999102971780.

[21] N. Young. *An Introduction to Hilbert Space*. Cambridge mathematical textbooks. Cambridge University Press, 1988. ISBN 9780521337175. URL https://books.google.no/books?id=_igwFHKwcyYC.

[22] T. J. R. Hughes, L. Mazzei, and K. E. Jansen. Large eddy simulation and the variational multiscale method. *Computing and Visualization in Science*, 3(1-2):47–59, 2000. ISSN 1432-9360. doi: 10.1007/s007910050051. URL http://dx.doi.org/10.1007/s007910050051.

[23] M. O. Deville, P. F. Fischer, and E. H. Mund. High-order methods for incompressible fluid flow. 2002. doi: 10.1017/cbo9780511546792. URL http://dx.doi.org/10.1017/CBO9780511546792.

[24] J. L. Guermond, P. Minev, and Jie Shen. An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 195(44–47):6011 – 6045, 2006. ISSN 0045-7825. doi: http://dx.doi.org/10.1016/j.cma.2005.10.010. URL http://www.sciencedirect.com/science/article/pii/S0045782505004640.

[25] *NEK5000 documentation*, 2015. URL http://nek5000.github.io/NekDoc/Nek_users.pdf.

[26] Y. Maday, A. T. Patera, and E. M. Rønquist. An operator-integration-factor splitting method for time-dependent problems: Application to incompressible fluid flow. *Journal of Scientific Computing*, 5(4):263–292, 1990. ISSN 0885-7474. doi: 10.1007/BF01063118. URL http://dx.doi.org/10.1007/BF01063118.

[27] P. F. Fischer and J. W. Lottes. Hybrid Schwarz-multigrid methods for the spectral element method. *J. Sci. Comput*, pages 45–78.

[28] J. B. Perot. An analysis of the fractional step method. *Journal of Computational Physics*, 108(1):51 – 58, 1993. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1993.1162. URL http://www.sciencedirect.com/science/article/pii/S0021999183711629.

[29] S. Abdallah. Comments on the fractional step method. *Journal of Computational Physics*, 117(1):179 – 180, 1995. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1995.1055. URL http://www.sciencedirect.com/science/article/pii/S0021999185710558.

[30] J. B. Perot. Comments on the fractional step method. *Journal of Computational Physics*, 121(1):190 – 191, 1995. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1995.1189. URL http://www.sciencedirect.com/science/article/pii/S0021999185711898.

[31] A. G. Tomboulides, J. C. Y. Lee, and S. A. Orszag. Numerical simulation of low mach number reactive flows. *J. Sci. Comput.*, 12(2):139–167, June 1997. ISSN 0885-7474. doi: 10.1023/A:1025669715376. URL http://dx.doi.org/10.1023/A:1025669715376.

[32] *CDP VIDA - Cascade Technologies*. URL http://www.cascadetechnologies.com/pdf/VIDA.pdf.

[33] S. Turek and M. Schäfer. Recent benchmark computations of laminar flow around a cylinder, 1996.