NTNU

Project TMA4500

Application of nek5000 to dispersion simulations

Author:
Magnus Aarskaug Rud

Supervisor:
Anne Kværnø

Research Group Name
IMF

December 2015

NTNU

Abstract

Faculty Name
IMF

Master Thesis

Application of nek5000 to dispersion simulations

by Magnus Aarskaug Rud

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl.

Contents

Notation

CONVENTION we let subscript h denote the discretized variables

Chapter 1

Introduction

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

Chapter 2

Problem description

What should be said about the stokes problem and the infsup condition??

2.1 The Incompressible Navier-Stokes equation

The physics regarding fluids in motion are described mathematically by the Navier-Stokes (N-S) equation. The equations can be derived in many ways, and it is referred to [1] for a complete description of the necessary assumptions and simplifications. The general idea is to conserve momentum and mass in a control domain providing a system of two equations. In this thesis only the incompressible N-S equation will be considered, with the assumption of an incompressible flow the conservation of mass results in a divergence free restriction on the solution u. Without further introduction the incompressible N-S equations are stated as

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} &= \mathbf{f} + \nabla \cdot \tau, \\ \nabla \cdot \mathbf{u} &= 0. \end{aligned} \tag{2.1}$$

These equations have been studied for centuries and different physical situations lead to different simplifications and sets of equations. Examples are the Euler equations, Stokes problem and Darcy flow. In particular the Stokes problem is applied a lot as an initial test of the full N-S problem. Before attempting to solve these equations it is important to understand the role of each term and their mathematical influence on the problem.

- $\partial \mathbf{u}/\partial t$ - The time-derivative of the flow, for a steady state flow this term will be equal zero. The discretization of this term is often based on some implicit scheme in order to improve stability.

- $\mathbf{u} \cdot \nabla \mathbf{u}$ - The convective term, describes the transport due to the flow itself on each of its components. The term is not present in Stokes problem. The mathematical operator corresponding to this term is non-linear and non-symmetric, and does therefore require the equations to be solved by some iteration procedure. Even with linear advection the operator is a source to instability and needs to be handled carefully.

- $\nabla \cdot \tau$ - $\tau = -p + \nu[\nabla \mathbf{u} + \nabla^T \mathbf{u}]$ is known as the Reynolds stress tensor for incompressible flows. The term $\nabla \cdot \tau$ simplifies to $-\nabla p + \nu \Delta \mathbf{u}$ if the velocity is assumed to be divergence free and the viscosity is assumed to be constant. It is a symmetric tensor, hence another 6 unknowns is introduced.

- $\nu \Delta \mathbf{u}$ - The diffusive term describes the natural diffusion of the fluid. The effect of diffusion is determined by the viscosity of the fluid. The corresponding mathematical operator stabilizes the system and it is therefore generally easier to solve the N-S equations for high-viscosity fluids.

- ∇p - The pressure gradient, Removal of this term results in a pure advection diffusion problem.

- $\nabla \cdot \mathbf{u}$ - The divergence free condition from the mass equation.

- $\mathbf{f}$ - General term describing external body forces such as gravity. For incompressible flow the gravity term is incorporated in the pressure term, $\nabla p = \nabla p + \rho \mathbf{g}$.

- Re - The Reynolds number defined as UL/ν where U is the velocity scale, L is the length scale and ν is the viscosity of the fluid. Re describes the relation between the biggest length scales of the flow and the viscous length scales. Notice how for large Reynolds number the unstable non-linear term dominates the transportation.Turbulent flows are characterized by high Reynolds number.

For large Reynolds number the huge span in length scales requires an extremely fine mesh if the equations 2.1 are to be solved exactly. Because a fine mesh implies a high computational cost a direct numerical solution (DNS) is not feasible for problems of a

certain geometrical complexity. There are many different approaches on how to resolve the turbulent term and in this thesis the main approach will be through Large Eddy Simulations (LES) which will be discussed in the following section.

2.1.1 Weak formulation of N-S

Let us first assume constant viscosity, enabling the simplification $\nabla \cdot \tau = -\nabla p + \nu \Delta \mathbf{u}$. The numerical algorithms applied in this thesis requires a weak formulation of equation 2.1. Before the weak form is derived a few operators will be defined in order to simplify the final expression.

$$(\mathbf{u}, \mathbf{v})_{L_2} = \int_\Omega \mathbf{u} \cdot \mathbf{v} d\Omega \tag{2.2}$$

$$\mathcal{A}(\mathbf{u}, \mathbf{v}) = (\nabla \mathbf{u}, \nabla \mathbf{v})_{L_2} \tag{2.3}$$

$$\mathcal{B}(\mathbf{u}, p) = (\nabla \cdot \mathbf{u}, p)_{L_2} \tag{2.4}$$

$$\mathcal{C}(\mathbf{w}; \mathbf{u}, \mathbf{v}) = (\mathbf{w} \cdot \nabla \mathbf{u}, \mathbf{v})_{L_2} \tag{2.5}$$

A weak formulation is obtained by multiplying with a test function $\mathbf{v}$ and integrating over the entire domain.

$$\begin{gathered} \int_\Omega \frac{\partial \mathbf{u}}{\partial t} \cdot \mathbf{v} d\Omega + \int_\Omega (\mathbf{u} \cdot \nabla) \mathbf{u} \cdot \mathbf{v} d\Omega = -\int_\Omega \nabla p \cdot \mathbf{v} d\Omega + \nu \int_\Omega \Delta \mathbf{u} \cdot \mathbf{v} d\Omega, \\ \int_\Omega (\nabla \cdot \mathbf{u}) \mathbf{q} d\Omega = 0. \end{gathered} \tag{2.6}$$

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

$$\begin{gathered} (\frac{\partial \mathbf{u}}{\partial t}, \mathbf{v}) + (\mathbf{u} \cdot \nabla \mathbf{u}, \mathbf{v}) = (\nabla \cdot \mathbf{v}, p) - \nu (\nabla \mathbf{u}, \nabla \mathbf{v}), \\ (\nabla \cdot \mathbf{u}, q) = 0. \end{gathered} \tag{2.7}$$

Finally, by using the notation introduced in 2.5 the weak formulation of the incompressible N-S equation can be stated as:

Find $(\mathbf{u}, p) \in H^1(\Omega) \times L^2(\Omega)$ such that

$$\begin{aligned}(\frac{\partial \mathbf{u}}{\partial t}, \mathbf{v}) + \mathcal{C}(\mathbf{u}; \mathbf{u}, \mathbf{v}) &= \mathcal{B}(\mathbf{v}, p) - \nu\mathcal{A}(\mathbf{u}, \mathbf{v}), \\ \mathcal{B}(\mathbf{u}, q) &= 0.\end{aligned} \tag{2.8}$$

$\forall\ (\mathbf{u}, p) \in H^1(\Omega) \times L^2(\Omega)$.

In order to solve this equation numerically, everything has to be discretized and expressed using a particular set of basis functions. This procedure will be discussed more thoroughly in chapter 3, but the idea is that the solution $(\mathbf{u}, p)$ is approximated by a discretized solution $(\mathbf{u}_h, p_h)$ and the bilinear operators can be represented by matrices. The discretized system of equations can be stated as

$$M\frac{\partial \mathbf{u}_h}{\partial t} + C(\mathbf{u}_h)\mathbf{u}_h = Dp_h - \nu A\mathbf{u}_h, \tag{2.9}$$

$$D^T\mathbf{u}_h = 0. \tag{2.10}$$

2.2 The passive scalar equation

The N-S equation explains how a fluid will behave and solving it provides a complete pressure-velocity field of the domain of interest, but it does not provide the answer of how heat will distribute in this flow, or how a gas will spread. The equation corresponding to this physical problem will be referred to as the passive scalar equation for any scalar ϕ and is stated as

$$\rho c_p(\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla\phi) = \nabla \cdot (k_i \nabla\phi_i) + (q_{vol})_i. \tag{2.11}$$

The constants k, and ρc_p are interpreted depending on the scalar transported. For dispersion of neutral gas they resemble the viscosity and mass flux.

write more about this equation and how it is applied in this thesis

2.3 Resolving the turbulent term using LES

tydeliggjor filteret!, og bruk en annen τ, filtervidde, tydeligg diff antakelse When DNS is not feasible due to 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 independently of time and location 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 are modelled. Some source on the importance of SGS-model

2.3.1 Filter

A filter in its general form is given as

$$U^r(\mathbf{x},t) = \int_\Omega G_r(\mathbf{r},\mathbf{x})U(\mathbf{x}-\mathbf{r},t)d\mathbf{r} \tag{2.12}$$

In order to perform a LES a filter width a certain width Δ has to be chosen, the filtered N-S equations are given as show commutative properties and important simplifications of the filter?

$$\begin{aligned} \frac{\partial \mathbf{u}^r}{\partial t} + \mathbf{u}^r \cdot \nabla \mathbf{u}^r &= -\nabla p^r + \nu \Delta \mathbf{u}^r - \nabla \cdot \tau, \\ \nabla \cdot \mathbf{u}^r &= 0. \end{aligned} \tag{2.13}$$

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

$$\tau_{ij}(u_i, uj) = (u_i u_j)^r - u_i^r u_j^r \tag{2.14}$$

This tensor is a consequence of applying the filter on the non-linear advection term.

Should I show where this tensor comes from in a clearer way?

2.3.2 dynamic Smagorinsky-Lilly SGS-model

The problem is now reduced to modelling this tensor, the most common SGS-model is the dynamic Smagorinsky-Lilly model. The initial progress of this model was made by

Lilly in 1967 [2] suggesting the following model of the SGS-tensor

$$\tau_{ij} = -2C_d l^2 \mathcal{S}^r s^r_{ij}, \tag{2.15}$$

$$s^r_{ij} = \frac{1}{2}\left(\frac{\partial u^r_i}{\partial x_j} + \frac{\partial u^r_j}{\partial x_i}\right), \tag{2.16}$$

$$\mathcal{S}^r = \sqrt{2 s^r_{ij} s^r_{ij}}. \tag{2.17}$$

Where l denotes the filter width. The resolved strain rate is easily calculated and the problem is now reduced to determining the constant C_d. There were several attempts to determine this constant for the entire domain, but finally in 1991 [3] a dynamic subgrid-scale model was presented calculating a C_d locally in time and space. The general idea is that the C_d is independent of the filter width and from this assumption develop an expression for the dynamical constant.

Let a, b denote two distinct filters with corresponding filter widths l_a, l_b. Let τ_{ij} and T_{ij} denote the stresses based on single- and double filtering operations on the N-S equations

$$\begin{aligned} \tau_{ij} &= (u_i u_j)^a - u^a_i u^a_j, \\ T_{ij} &= ((u_i u_j)^a)^b - (u^a_i)^b (u^a_j)^b. \end{aligned} \tag{2.18}$$

Applying the b filter on the first tensor in eq 2.18 allows us to define a new tensor L_{ij} which depends only on the a-filtered variables. The identity is known as the Germano identity and was first introduced in 1991 [3],

$$L_{ij} = T_{ij} - (\tau_{ij})^b = (u^a_i u^a_j)^b - (u^a_i)^b (u^a_j)^b. \tag{2.19}$$

The fact that this tensor depends on the a-filtered solution and not the resolved one is very important and is what allows us to compute it exactly.

Substituting the stress-tensors with their corresponding expression from equation 2.17 and assuming a dynamic constant unaffected by the filter one obtains an approximation

for L_{ij} which is also computable,

$$\begin{aligned} L_{ij} &\approx 2C_d l_b^2 \mathcal{S}^{ab} s_{ij}^{ab} - 2(C_d l_a^2 \mathcal{S}^a s_{ij}^a)^b \\ &\approx 2C_d l_a^2 [\lambda \mathcal{S}^{ab} s_{ij}^{ab} - (\mathcal{S}^a s_{ij}^a)^b] \\ &= 2C_s M_{ij}. \end{aligned} \tag{2.20}$$

$$M_{ij} = \lambda^2 \mathcal{S}^{ab} s_{ij}^{ab} - (\mathcal{S}^a s_{ij}^a)^b \tag{2.21}$$

$$C_s = C_d l_a^2 \tag{2.22}$$

$$\lambda = l_b / l_a \tag{2.23}$$

Minimizing the mean-square error between the exact L_{ij} as expressed in equation 2.19 and the Boussinesq-based approximation in 2.20 yields the best approximation for the dynamic Smagorinsky constant

$$C_s = \frac{M_{ij} L_{ij}}{2 M_{kl} M_{kl}}. \tag{2.24}$$

Note that the double indices implies summing. This expression is however not a stable option and to deal with this most implementations apply some sort of mean in either time and/or space when calculating the constant. Another interesting property is that the constant C_s and the final SGS-tensor $\tau = -2C_s \mathcal{S} s_{ij}$ are independent of smallest resolution scale, the only necessary variable is the relationship between the two grid-sizes. The assumption made in this model is that turbulence behaves as diffusion, and the turbulent viscosity is defined for this case as $\nu_t = C_s \mathcal{S}$.

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

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

Notice that if v_t were to be a constant in the entire domain this equation would be equivalent to the one for a fluid with viscosity $\nu' = \nu + \nu_t$.

Clarify the effect on N-S eq.

2.4 Solution methods of incompressible N-S

Rewrite this introduction

Chapter 2.4 in the numerics part! time integration for NS

A non-linear set of equations requires a non-trivial solution method, and when the domain of the problem can be anything from a simple channel to a moving turbine there are many considerations that needs to be made. Although the equations have been known for over 200 years no one has been able to prove or disprove the well-posedness of the problem. Some of the most common algorithms will be discussed in the following subsections

2.4.1 Operator-splitting techniques

Let us consider a simplified transient problem

$$\frac{du}{dt} = f + g. \tag{2.26}$$

both f and g are functions of u and therefore of t as well. By integrating on both sides one obtains

$$u(s) - u(0) = \int_0^s f dt + \int_0^s g dt. \tag{2.27}$$

The idea is then to evaluate one of the integrals by an explicit scheme and the other one with an implicit scheme. A general formulation of this for a time step would be on the form

$$u^{n+1} - u^n = \sum_{k=0}^{N} a_k f^{n-k} + \sum_{k=0}^{N} b_k g^{n-k+1}. \tag{2.28}$$

In equation 2.28 the first integral in 2.27 is evaluated by an explicit scheme and the second one by an implicit scheme. The coefficients a_k, b_k corresponds to the elected scheme. When operator splitting is applied in this thesis an Adam-Bashford scheme is chosen for the implicitly evaluated integral while Runge-Kutta is applied for the explicit scheme. Operator splitting is a very convenient strategy for the N-S equations which consists of the easily invertible Laplacian and the demanding convection operator.

Do I need a chapter about Runge-Kutta and Adam-Bashford?

2.4.2 Operator integrating factor schemes (OIFS)

The operator-splitting method described in the previous chapter may lead to an unstable scheme, OIFS is a similar method but it offers a more stable scheme. 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 [4].

By considering the NS-equation in a general operational form

$$M\frac{d\mathbf{v}}{dt} + C\mathbf{v} = -A\mathbf{v} + D_i p + Mf_i. \tag{2.29}$$

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

$$\frac{dQ(t)M\mathbf{v}}{dt} = Q(t)M\frac{d\mathbf{v}}{dt} + \frac{d}{dt}(Q(t)M)\mathbf{v}, \tag{2.30}$$

$$= Q(t)M\frac{d\mathbf{v}}{dt} + Q(t)C\mathbf{v}. \tag{2.31}$$

This way equation 2.29 can be written as

$$\frac{dQ(t)M\mathbf{v}}{dt} = Q(t)(-A\mathbf{v} + Dp + M\mathbf{f}). \tag{2.32}$$

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

$$\sum_{j=0}^{k} \beta_j Q(t^{n+1-j})M\mathbf{v}^{n+1-j} = \Delta t\, Q(t^{n+1})(-A\mathbf{v}^{n+1} + Dp^{n+1} + M\mathbf{f}^{n+1}). \tag{2.33}$$

Applying the fact that $Q(t^{n+1}) = I$ enables us to write equation 2.34 as

$$\beta_0 M\mathbf{v}^{n+1} + \sum_{j=1}^{k} \beta_j Q(t^{n+1-j})M\mathbf{v}^{n+1-j} = \Delta t(-A\mathbf{v}^{n+1} + Dp^{n+1} + M\mathbf{f}^{n+1}). \tag{2.34}$$

Notice how all the easily invertible operators are evaluated implicitly, while the convective non-linear term is hidden in the BDFk scheme. Now the OIFS method allows us to find the terms in the sum in a rather elegant fashion. First of all the auxiliary variable $\tilde{\mathbf{v}}_j$ is defined such that $Q(t^{n+1-j})M\mathbf{v}^{n+1-j} = M\tilde{\mathbf{v}}_j$ thus enabling us to find the

summation expression by solving the initial value problem

$$\begin{aligned} M\frac{d\tilde{\mathbf{v}}_j}{ds} &= -C(\tilde{\mathbf{v}}_j(s))\tilde{\mathbf{v}}_j(s), \qquad t^{n+1-j} \leq s \leq t^{n+1} \\ \tilde{\mathbf{v}}_j(t^{n+1-j}) &= \mathbf{v}(t^{n+1-j}). \end{aligned} \tag{2.35}$$

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

The final scheme applied for solving equation 2.29 when applying OIFS consists of one scheme for solving equation 2.34 and another scheme for solving equation 2.35. When applied in this thesis the first scheme is an implicit BDFk-scheme while the second is an explicit kth order Runge-Kutta scheme. It is important to add that the error induced by this method is non-vanishing and is only recommended for assuring stability for very instable problem.

2.4.3 Fractional step

Fractional step is an algorithm that can be divided into four separate steps. For simplicity let us write the N-S equation as

$$\partial_t v = -Av + Dp - Cv. \tag{2.36}$$

Where ∂_t, A, D, C Denotes the time-derivate, linear, 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.

$$\begin{aligned} v^* &\leftarrow \frac{dv}{dt} = Cv, \\ p^{n+1} &\leftarrow \Delta\, p = \nabla \cdot \frac{v^*}{\Delta t}, \\ v^{**} &\leftarrow \frac{dv^*}{dt} = Dp^{n+1}, \\ v^{n+1} &\leftarrow \frac{dv^{**}}{dt} = -Av^{**}. \end{aligned} \tag{2.37}$$

As earlier mentioned this method is convenient because it allows us to handle the different terms with different solution techniques. So since the first equation in 2.37 involves the non-linear skew-symmetric advection operator this equation is solved using a explicit

Adam Bashford scheme. The second equation is the Possion pressure equation which assures a divergence free velocity field. 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 v^{n+1}, but since this solution is yet to be calculated the intermediate velocity field v^* is used to impose the boundary conditions. With p^{n+1} known the third equation is simply an update of the velocity in order to impose the divergence free condition. Now the last equation is solved implicitly due to its nice symmetric structure. This results in a system equivalent to the Helmholtz problem which will be discussed in detail in chapter 3. This can be easily shown by discretizing the equation,

$$\begin{aligned} (v^{n+1} - v^n)/\Delta t &= -Av^{n+1}, \\ Av^{n+1} + \frac{1}{\Delta t} v^{n+1} &= v^n. \end{aligned} \tag{2.38}$$

Knowing that A is the discrete Laplacian and v^n is a known variable this is similar to problem 3.9.

This method provides an efficient algorithm, but is known to produce errors of order O(1). The problem is the pressure Poisson equation which is solved with incorrect boundary conditions. The first step can also be evaluated in an OIFS-way to gain stability, by rewriting the initial equation as described in the previous chapter. This allows you to solve the unstable advection operator with multiple substeps. Explicitly the first step in 2.37 would be solved by applying the discretization used in 2.34 with an empty right hand side since the rest of the terms are take care of in the next steps. The IVP 2.35 is then solved to obtain v^*.

Chapter 3

Numerical algorithms

Ta med konkrete konvergensteoremer!

Rettskrivning!

Dropp hattfunksjoner, make ediff pretty! figur som viser energispekteret, ha et konkret eksempel med

3.1 Galerkin formulation

obs! sterk formulering, dette må rettes opp i Throughout this thesis all numerical methods will be based on the Galerkin formulation. Let us consider a general boundary value problem (BVP)

$$\begin{aligned} \mathcal{L}u &= f \quad \text{in } \Omega \\ \mathcal{B}u &= g \quad \text{on } \partial\Omega. \end{aligned} \tag{3.1}$$

The domain Ω is a closed subspace of $\mathbb{R}^d$, $\mathcal{L} : X(\Omega) \to Y(\Omega)$ and $\mathcal{B} : X(\partial\Omega) \to B(\partial\Omega)$ are two linear operators, $f \in Y(\Omega)$ and $g \in B(\partial\Omega)$ are known functions and $u \in X(\Omega)$ is the wanted solution. The space $X(\Omega)$ will be denoted as the search space. A weak formulation can now be obtained by multiplying the first equation in 3.1 by a test function $v \in X^t(\Omega)$ and integrating over the domain Ω. By choosing $X^t(\Omega) = X(\Omega)$ the Galerkin formulation is obtained. For more examples and information on this subject the first chapters in [5] are recommended.

By the Lax-Milgram theorem it is known that a BVP is well-posed if the Operator $\mathcal{L}$ is both bounded and coersive.

Solving the BVP numerically involves choosing discrete subsets of X, X^t, Y, B. These will be denoted X_h, X_h^t, Y_h, B_h. The discrete subspaces can be chosen in a number of ways and the defining basis functions vary from one numerical method to another. In this thesis the spectral and finite element basis will be shortly stated and the spectral element basis will be viewed in more detail.

vær presis med den svake formuleringen!!

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 connectivity properties of the elements it guaranties a sparse system. The decomposition of the geometrical domain into a finite amount of elements chosen according to the problem wanted to solve, 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 [5], but some of the key properties will be stated here in order to provide a thorough understanding of the spectral element method (SEM).

FEM provides an algorithm for solving any well-posed BVP 3.1 and the mathematical formulation is obtained by first finding the Galerkin formulation and choosing a discrete subset $X_h^p(\Omega) \subset X(\Omega)$ spanned by the finite element basis functions ϕ_i^p. p denotes the polynomial degree of the basis-functions, in 1D and for $p = 1$ the basis functions are defined as

$$\phi_i(x) = \begin{cases} \frac{x - x_i}{x_i - x_{i-1}} & \text{if } x_{i-1} \leq x \leq x_i, \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & \text{if } x_i \leq x \leq x_{i+1}, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that $\text{supp}(\phi_i) = [x_{i-1}, x_{i+1}]$ and as a consequence of this $(\phi_i, \phi_j)_\Omega = 0$ if $|i - j| > 1$. These qualities 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 key 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 $[-1,1]^d$ 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 $u_h \in X^h$ is a projection of the actual solution u of the BVP onto the discrete space X^h. Provided that the initial BVP is well-posed there exists to constants $M, \alpha > 0$ known as the bounded and coercivity constant such that the error of the solution can be reduced to a pure interpolation error. The result is known as Cea's lemma,

$$||u - u_h||_X \leq \frac{M}{\alpha}||u - I_h u||_X. \tag{3.2}$$

Where I_h is the projection operator.

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 let h denote the general size of the elements and p the order of the polynomial basis that defines X^h. Roughly speaking the error is given as $e = Ch^p$, C being some positive constant. This is not a formal truth but rather a simple expression to get some intuition on how the error behaves. Factors such as geometric complexity, condition-number,non-linear operators and the regularity of the solution will all provide slightly more complicated error estimates. However for a simpler BVP such as $u, f, g \in C^\infty(\Omega), \Omega = [-1,1]^d, \mathcal{L} = -\Delta, \mathcal{B} = 0$ the error estimate is valid. performing 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 spectral convergence, 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 [6] chapter 2.6.

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 (known 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 [7]. SM can be reduced to a interpolation problem such as FEM, and are very 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 draw-back of SM are the difficulties with applications to complex geometries. Although the system of equations surging from a BVP can be constructed in an elegant way 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

$$u_h(x) = \sum_{i=0}^{N} a_i \psi_i(x). \tag{3.3}$$

Where the coefficients a_i are called the expansion coefficients. There are many possible choices for the basis and the belonging coefficients, in this thesis and the algorithms used the 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 [8].

The GLL-nodes $\{\xi_i\}_{N+1}$ are given as the solutions of the equation

$$(1 - \xi^2) L'_N(\xi) = 0. \tag{3.4}$$

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

$$\frac{d}{dx}\left[(1-x^2)\frac{d}{dx}L_n(x)\right] + n(n+1)L_n(x) = 0. \tag{3.5}$$

With equations 3.4 and 3.5 the local spectral basis functions ψ_j can be stated as

$$\psi_j(x) = \prod_{i\neq j}^{N} \frac{x - x_i}{x_j - x_i}. \tag{3.6}$$

How much details on these polynomials and quadratures should be included?

$\{x_i\}$ being the solutions to 3.4. Note that $\psi_j(x_i) = \delta_{ij}$. The expansion coefficients in 3.3 are then chosen as $a_i = u_i := u(x_i)$. This definition of the expansion coefficients is very 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, for 3 dimensions the basis functions Ψ is given as

$$\Psi_l(\mathbf{x}) = \psi_i(x)\psi_j(y)\psi_k(z). \tag{3.7}$$

This expansion to multiple dimensions preserves the $\Psi_l(\mathbf{x}_m) = \delta_{lm}$ In order to clarify some of the concepts the SM approach will be applied on the Helmholtz equation

$$-\Delta u + \lambda u = f \quad \text{in } \Omega, \tag{3.8}$$

$$u = 0 \quad \text{on } \partial\Omega. \tag{3.9}$$

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 is the given as

Find $u \in V$ st.

$$\int_\Omega \nabla u \cdot \nabla v d\Omega + \lambda \int_\Omega uv d\Omega = \int_\Omega fv d\Omega \qquad \forall v \in V \tag{3.10}$$

In order 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 $u_h \in V_h$ st.

$$\sum_l \left(u_l \int_\Omega \nabla\Psi_l \cdot \nabla\Psi_m d\Omega + u_l\lambda \int_\Omega \Psi_l\Psi_m d\Omega \right) = \int_\Omega f\Psi_m d\Omega \qquad \forall\Psi_m \in V_h. \tag{3.11}$$

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

$$\sum_l \left(u_l \sum_n \rho_n \nabla\Psi_l(\mathbf{x}_n) \cdot \nabla\Psi_m(\mathbf{x}_n) + u_l\lambda \sum_n \rho_n \Psi_l(\mathbf{x}_n)\Psi_m(\mathbf{x}_n) \right) \tag{3.12}$$

$$= \sum_n \rho_n f\Psi_m(\mathbf{x}_n) \qquad \forall\Psi_m(\mathbf{x}_n) \in V_h. \tag{3.13}$$

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

$$(A + \lambda M)u_h = \tilde{f}. \tag{3.14}$$

Where the elements in the matrices and vectors are given as

$$\begin{aligned} A_{lm} &= \sum_n \rho_n \nabla\Psi_l(\mathbf{x}_n) \cdot \nabla\Psi_m(\mathbf{x}_n), \\ M_{lm} &= \sum_n \rho_n \Psi_l(\mathbf{x}_n)\Psi_m(\mathbf{x}_n) = \rho_l \delta_{lm}, \\ (u_h)_l &= u(\mathbf{x}_l), \\ \tilde{f}_m &= \sum_n \rho_n f(\mathbf{x}_n)\Psi_m(\mathbf{x}_n) = \rho_m f(\mathbf{x}_j). \end{aligned} \tag{3.15}$$

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.

3.4 Spectral element method

In the early 1980's the idea to combine FEM and SM came along in order to obtain the robustness and sparse properties of FEM combined with the spectral convergence rate provided by SM. The result was the Spectral element method. Several formulations was

investigated mainly by Patera, Maday and Rønqvist in the papers ... It is important to understand that when solving the N-S equations the efficiency of the solution method is extremely important. The algorithm has to be parallelizable and the development of the super computers and computational clusters has played an important role in deciding the method 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 $\mathcal{F} : \hat{\Omega} \rightarrow \Omega$. An example of this map is the Gordon-Hall procedure described in chapter 3.4.3. The Jacobian is then given as the transposed tensor derivative of $\mathcal{F}$

$$\mathbf{J} = (\nabla \otimes \mathcal{F})^T = \begin{bmatrix} \frac{\partial \mathcal{F}_1}{\partial x} & \frac{\partial \mathcal{F}_1}{\partial y} \\ \frac{\partial \mathcal{F}_2}{\partial x} & \frac{\partial \mathcal{F}_2}{\partial y} \end{bmatrix}, \tag{3.16}$$

$$J = \det(\mathbf{J}). \tag{3.17}$$

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

$$\begin{aligned} d\mathbf{x} &= \mathbf{J} d\boldsymbol{\xi} \\ \int_{\Omega} f(\mathbf{x}) d\mathbf{x} &= \int_{\hat{\Omega}} \hat{f} J d\boldsymbol{\xi} \\ \nabla u &= \mathbf{J}^{-T} \hat{\nabla} \hat{u}. \end{aligned} \tag{3.18}$$

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.

Although the whole domain Ω is deformed, the deformation of each element $\{\Omega_k\}$ is normally a lot less crucial.

Let us again consider the Helmholtz problem 3.9, 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 $\Omega = \bigcup_{k=1}^{K} \Omega_k$. The weak formulation in equation 3.10 can then be written according to the SEM formulation

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

$$\int_{\Omega_k} \nabla u \cdot \nabla v d\Omega + \lambda \int_{\Omega_k} uv d\Omega = \int_{\Omega_k} fv d\Omega \qquad \forall v \in X_k^N. \tag{3.19}$$

Where $X_k^N = H^1(\Omega_k) \bigcap \mathbb{P}^N(\Omega_k)$. The same discretization procedure as performed for the pure spectral case is now done for each of the sub domains Ω_k,

$$\sum_i \left(u_i \int_{\Omega_k} \nabla \psi_i \cdot \nabla \psi_j d\Omega + u_i \lambda \int_{\Omega_k} \psi_i \psi_j d\Omega \right) = \int_{\Omega_k} f \psi_j d\Omega \qquad \forall \psi_j \in V_h. \tag{3.20}$$

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 3.18 to 3.20 yields

$$\begin{aligned} \sum_i \left(u_i \int_{\hat{\Omega}_k} (\mathbf{J}^{-T} \hat{\nabla} \hat{\psi}_i)^T (\mathbf{J}^{-T} \hat{\nabla} \hat{\psi}_j) J d\hat{\Omega} + u_i \lambda \int_{\hat{\Omega}_k} \hat{\psi}_i \hat{\psi}_j J d\hat{\Omega} \right) &= \int_{\hat{\Omega}_k} \hat{f} \psi_j J d\hat{\Omega} \qquad \forall \psi_j \in V_h. \\ \sum_i \left(u_i \int_{\hat{\Omega}_k} \hat{\nabla}^T \hat{\psi}_i \mathbf{J}^{-1} \mathbf{J}^{-T} \hat{\nabla} \hat{\psi}_j J d\hat{\Omega} + u_i \lambda \int_{\hat{\Omega}_k} \hat{\psi}_i \hat{\psi}_j J d\hat{\Omega} \right) &= \int_{\hat{\Omega}_k} \hat{f} \psi_j J d\hat{\Omega} \qquad \forall \psi_j \in V_h. \end{aligned} \tag{3.21}$$

Notice how the integrals depend on the Jacobian $\mathbf{J}$ and its determinant J,

Do a more thorough analysis of the meaning of J??

Hence 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 sub domain. This procedure is general and can be performed on almost any deformed domain as oppose to SM.

Should direct stiffness summation be mentioned ?

3.4.1 Filtering

Although SEM provides spectral convergence in space and 2nd or 3rd order accuracy in time the stability of a straight forward implementation is not guaranteed, spurious nodes appear as shown in chapter 2.4.1.2 in [6]. In [9] a filter-based stabilization is introduced for SEM applied on Navier Stokes equation. The idea is to project a part $0 < \alpha < 1$ of the highest mode onto a polynomial space of lower order, explicitly they define the filter F_α as

$$F_\alpha = \alpha I_{N-1} + (1 - \alpha) I_d. \tag{3.22}$$

Where I_{N-1} is the projector from $\mathbb{P}_N$ to $\mathbb{P}_{N-1}$ and I_d is the identity operator. α is recommended to be somewhere in the interval $(0.05, 0.3)$.

The explication of why F_α has a filtering effect is best explained by considering the more general explication by Pasquetti and Xu in [10]. 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 the solution to some PDE, 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

$$F_\alpha u = (1 - \alpha)\hat{u}_N L_N + \hat{u}_{N-1} L_{N-1} + (\hat{u}_{N-2} + \alpha \hat{u}_N) L_{N-2} + \sum_{i=0}^{N-3} \hat{u}_i L_i. \tag{3.23}$$

From this identity the effect of the filter becomes clear, it is simply removing a part of the highest order 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 [6].

The filter is proved to be a very effective stabilization method and it preserves the spectral convergence rate. Another interesting property is that the filtering procedure does not imply dissipation of energy, let the energy norm be defined as $E(u) = ||u||_{L_2}^2$. By applying Parseval's identity [11] the difference in energy between the original solution

and the filtered solution is given as

$$\epsilon_{diff} = E(u) - E(F_\alpha u) \tag{3.24}$$

$$= 2\alpha\hat{u}_N(\hat{u}_N||L_N||^2 + \hat{u}_{N-2}||L_{N-2}||^2) - \alpha^2\hat{u}_N^2(||L_N||^2 + ||L_{N-2}||^2) \tag{3.25}$$

$$\approx \frac{2\alpha}{N}\left[(1-\frac{\alpha}{2})\hat{u}_N^2 + \hat{u}_N\hat{u}_{N-2}\right]. \tag{3.26}$$

Which can take both positive and negative values depending on the sign and size of $\hat{u}_N\hat{u}_{N-1}$. By applying the known norm of the Legendre polynomials the deduced absolute error ϵ_{diff} of the filtered energy is of order $\epsilon_{diff} \sim \alpha/N$. The approximation $||L_N||^2 \approx ||L_{N-2}||^2 \approx 1/N$ have been used to achieve the result in 3.26.

3.4.2 Aliasing

When evaluating the integral surging from the non-linear term in the N-S equation the polynomial to be integrated is order $2N + (N-1)$. The quadrature rule applied to solve the integrals are only exact up to an order $2N-1$, hence the error surging from this evaluation can be of significant size. Applying a non-sufficient quadrature to an integral like this is called a "variational crime". Applying a quadrature rule of a not sufficiently high order results in an aliasing effect 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 of order $3N$. The concept and illustrative examples are given in Chapter 2.4 in [6]. The effect of aliasing is made clear 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.

3.4.3 Gordon-Hall algorithm

Make figure!!

In order to work on complex geometries some elements require a certain deformation in order to be able to describe the entire domain. It is necessary to evaluate all the integrals surging from the weak formulation over a reference domain $\hat{\Omega} = [-1, 1]^d$ for sakes of 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

$$\tilde{\mathbf{x}} = \sum_i \sum_j \sum_k \mathbf{x}_{ijk} l_i(r) l_j(s) l_k(t). \tag{3.27}$$

l_i being the ith Lagrange polynomial. The full description of the algorithm with helpful figures can be found in [12] 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 $\Omega \in \mathbb{R}^2$, 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

$$\phi_0(\xi) = \frac{1-\xi}{2}, \qquad \phi_N(\xi) = \frac{1+\xi}{2}. \tag{3.28}$$

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 3.28 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.

$$\begin{aligned} \tilde{\mathbf{x}}_{i,j} = &\Gamma_{0,0}\phi_0(r_i)\phi_0(s_j) \\ &+ \Gamma_{0,N}\phi_0(r_i)\phi_N(s_j) \\ &+ \Gamma_{N,0}\phi_N(r_i)\phi_0(s_j) \\ &+ \Gamma_{N,N}\phi_N(r_i)\phi_N(s_j) \end{aligned} \tag{3.29}$$

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

$$\begin{aligned}\tilde{\mathbf{x}}_{i,j} = \tilde{\mathbf{x}}_{i,j} &+ (\Gamma_{i,0} - \tilde{\mathbf{x}}_{i,0})\phi_0(s_j) \\ &+ (\Gamma_{i,N} - \tilde{\mathbf{x}}_{i,N})\phi_N(s_j) \\ &+ (\Gamma_{0,j} - \tilde{\mathbf{x}}_{0,j})\phi_0(r_i) \\ &+ (\Gamma_{N,j} - \tilde{\mathbf{x}}_{N,j})\phi_N(r_i)\end{aligned} \tag{3.30}$$

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.

Chapter 4

Application of Nek5000

ta med flyttdiagram for l;sningen rydd opp i skjemaene! referer til en l;rerbok, beskriv vha numerisk Hver tydelig paa hva man itererer over. korte ned tekstene! There are many numerical solvers for turbulent flows 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 Differences, Finite Element Method, Spectral Element Method etc. The particular algorithm for resolving the Pressure-Velocity coupling, for instance Fractional Step, Poisson pressure and Uzawa. The type of simulation available also varies from solver to solver, whether they apply RANS, LES, DNS or something else. 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 section will be devoted to the handling of Nek5000 in particular, and can serve as a brief introduction to the code.

4.1 Nek5000 Basics

Nek5000 is a turbulent 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 great speedup. All the parallelization is accessed through subroutines and functions, enabling the user to make advanced functions without having to deal

directly with the functions from the MPI library. With SEM as the numerical method applied it is possible to obtain very accurate results.

Nek provides a basic tool for generation of mesh. For more complex geometries this tool cannot compare with more visualized-based softwares such as ICEM from ANSYS which exports mesh to several numerical solvers such as Fluent and Nastran. It is therefore very useful to have an automatic way of converting a mesh created in ICEM to the format required by Nek5000.

So far Nek5000 has supported three automatic routine 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 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

In order to work with Nek there are some practical information that needs to be clarified. Nek is recompiled for every case and the user specify all the case specific information in the three files `{.rea,.usr,SIZE,}`. These files along with the standard library that belongs to Nek are compiled using makenek which creates the executable file `nek5000`. The user guide [13] contains a tutorial which explains the necessary steps on how to get started with Nek. The next chapters will try to give some understanding on how the user is able to make the changes necessary for each case.

4.2.1 SIZE

Since Nek is mostly based on Fortran77 all memory allocations are done statically and needs to be specified explicitly before runtime. Most of these variables 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 defines the sizes of almost all the

arrays used in the program so it is important to define these variables as accurately as possible in order to optimize memory usage. The file is structured as a list of parameters, the first typical lines of a SIZE file are

```
parameter (ldim=3)                                          ! dimension
parameter (lx1=6,ly1=lx1,lz1=lx1,lelt=600,lelv=lelt) ! GLL-points,elements/processor
parameter (lxd=9,lyd=lxd,lzd=lxd)                           ! Order of De-aliasing
```

This file follows the standard Fortran77 syntax and parameters can be added as the user likes. Note that the number of GLL-nodes in each processor is `lx1*ly1*lz1*lelv`, and is a common length for many of the available arrays. It is important to be aware that `lelv` is an upper limit for the number of elements each processor can have, the actual number of elements may be smaller since the distribution of elements is uneven among the processors. It is however useful to make this upper bound as close as possible to the actual number in order to make the memory usage as efficient as possible. Another useful observation is that all the variables on these lines starts with `l`. This is not a rule, but rather a norm to enable a certain structure when working with the program. The variables defined here are common for all processors and is never changed, since the processors might have different number of elements a unique variable `nelv` (corresponding to `lelv`) is defined to be exactly the number of elements on that processor. Another such norm is the ending of the variable. When ending with v or 1, the variable is connected to the velocity. whereas t stands for temperature and 2 for pressure.

4.2.2 .REA

In `.rea` all the problem specific parameters are given. While the content in `SIZE` is an absolute necessity to even compile the program the `.rea` file 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. Example of the variables are DT,NSTEPS,IOSTEP which denotes the target time step, total number of steps and the frequency of writing data to file. There are also some physical variables such as VISCOS, RHOCP, DENSITY denoting the viscosity of the fluid, convectional constant and the density. In addition tolerances, target CFL-number, order of temporal discretization, preconditioner and de-aliasing info are also defined in this section. Of the remaining variables the user can

[h] 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	BCtype for all elements and their faces
E	THERMAL BC	Thermal BCtype for all elements and their faces
1	PRESOLVE/RESTART	Filename 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 is a predefined number depending on your problem which scales roughly as the number of elements, while $K \approx 1-25$ is user defined.*

define problem-specific parameters that can be applied in user-defined routines in `.usr`. It is important to edit this file with care, and pay attention to the implication of the variables. If for instance the Thermal BCs are included while `IFTEMP = F` an error will occur. Symmetric boundary conditions on surfaces that are not orthogonal to any of the Cartesian axes cannot be imposed unless `IFSTRS = T`. There are several of these subtle connections which the user has to be aware of when creating the environment for his simulation.

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 [13]. A list of those frequently used for this thesis are described below,

- `userbc` - Define the boundary conditions on the inflow-boundary, The routine is called once every time step for every node on the boundaries specified in the `.rea` with a lower-case 'v' or 't'.
- `uservp` - Define the eddy viscosity when applying LES. Only in use when `param(30)>0` in `.rea`, if this is the case then this routine is called every time step for all nodes in the domain.
- `userchk` - Read inflow-data, and specify the output. Called once by each processor after every time step and after the initialization.

- `usrdat2` - Project the geometry onto a deformed general surface. The details of how this routine is used will be specified further in chapter 5. The routine is called during the initialization, right after the geometry has been defined.

- `usrdat3` - Defines the interpolation algorithm that is applied to the inflow-data, the subroutine is called after `usrdat2` and the reading of `.rea`.

In addition to these routines all user-defined functions are specified in this file. The LES implementation in Nek is based on several subroutines specified in addition to those stated above. The project onto surface routine is also implemented in this file. These subroutines provide the user to do almost anything he would like, all variables can be accessed and transformed in whatever way desired.

One important thing to grasp when working with functions in `.usr` is how to access all the available variables. Almost all of the subroutines include `'TOTAL'` which is a file containing several other files where all the essential variables are defined. These variables are not always well-documented and a lot time has been spent searching through the routines of Nek in order to understand when the variables are defined and their purpose. In addition to all the variables defined there are roughly 1500 different subroutines and functions specified in different files. Some of these are well documented through commentaries in the code itself, but there are no list with useful functions. The majority of these routines are not of any use to the user itself, but the search for the useful routines is not any simpler of that reason. In absent if this information a list of useful functions and variables that are easily accessible from the subroutines in `.usr` are listed in Appendix B.

4.3 Steps in the solver

To best understand the work flow of Nek the subroutine `nek_advance` in `drive1.f` should be examined. This routine includes all the steps in one iteration of the solver. The routine is adapted so that it is functional for a number of different problems and user defined settings. The most important boolean switches used in this routine are described in the list below

- ifsplit - whether the $\mathbb{P}_N - \mathbb{P}_N$ or the $\mathbb{P}_N - \mathbb{P}_{N-2}$ is to be used.

- iftrans - transient or steady flow.
- ifheat - solving for heat.
- ifnav - natural convection for the scalar fields (Boolean array)
- param(103) - activation of filtering.

further the routines of interests are `fluid` and `heat`, the solvers for N-S equations and passive scalars respectively. Both subroutines are found in `drive2.f`. The understanding of these solvers are best achieved by studying equation 2.1 and 2.11. For the settings chosen in this thesis a fractional step procedure as described in chapter 2.4.3 is applied.

Talk about how the preconditioners work in Nek?

4.4 Nek5000 for complex geometries

For complex curved geometries such as bent cylinders, spheres, ellipsoidals etc. the user has to be able to express these surfaces analytically and write a routine in `usrdat2` that projects the points of interest onto the surface. 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 belongs 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.

Many turbulent solvers does 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 the initial grid is a lot coarser compared to equivalent settings in other solvers.

Comment on the MOAB possibility??

FIGURE 4.1: *Time-averaged concentration with a sample time of* 18.00 *s at* $z/H = 0.025$ *plotted horizontally and scaled with the free-stream velocity and emission rate. Compared against wind tunnel data. Two dashed lines on either side of the centerline represent the canyon.*

Chapter 5

Implementation

Some introduction that motivates the work done in this thesis

henvis til NS, nevn eksplisitt de ligningene som brukes

Ta med et turbulensbilde, og bilde av plumen.

5.1 Case 1: Gas dispersion in a simplified urban area

The scenario investigated in this work 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 5.1. In this figure h have been used as the length scale. The dotted lines indicates the positions where data is collected.

Scaling the 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 radius $r = 0.0515$. The grid used for the computations consists of 4425 elements and with a polynomial degree of 12 the total number of nodes $N \approx 7,6$mill.

The simulations are performed using Large Eddy Simulation (LES) with the dynamic Smagorinsky-Lilly subgrid-scale model. The release of gas will result in a plume that is

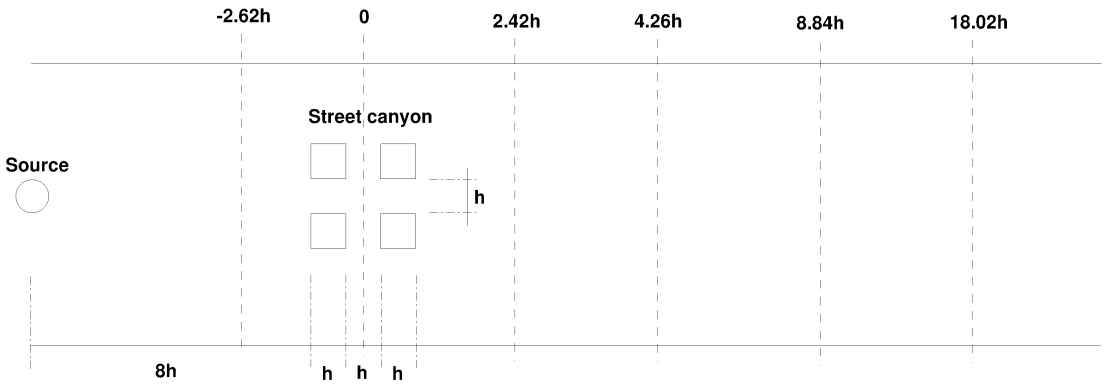

FIGURE 5.1: *Schematic overview of the domain from above. The data is collected along the dotted lines.*

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

TABLE 5.1: *Essential variables, *this value 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 a small amount from case to case.*

advected with the wind field. The size and shape of the plume at the indicated positions in figure 5.1 are compared with experimental data and simulations performed in Fluent and CDP. The wind-field in the tunnel is created by an inflow condition that is defined from previous simulations in CDP [14]. For clarification some of the variables repeatedly mentioned throughout this thesis will be stated explicitly in table 5.1.

The inflow conditions had to be extrapolated onto the domain at each time step. The velocity field on the inflow was generated in CDP, in order to create a boundary layer similar to the one found in the wind-tunnel. The inflow velocity was written to file every 0.0013s for a total of 28 seconds. An interpolation algorithm had to be implemented in order to adjust the inflow-data to each mesh.

5.2 Case 2: Drag and lift on a cylinder

A standard benchmark case for flow solvers is presented in [15]. 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 5.2. The constants applied in the description of the geometry and the coefficient scales are listed in table 5.2. The 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	$10^{-3}\text{m}^2/\text{s}$	Kinematic viscosity of the fluid
Re	20	Reynolds number

TABLE 5.2: *Constants for case 2*

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 case will yield a steady state solution at which point the coefficients are calculated and compared with the reference solutions. The spectral element method applied in Nek is known to be an accurate

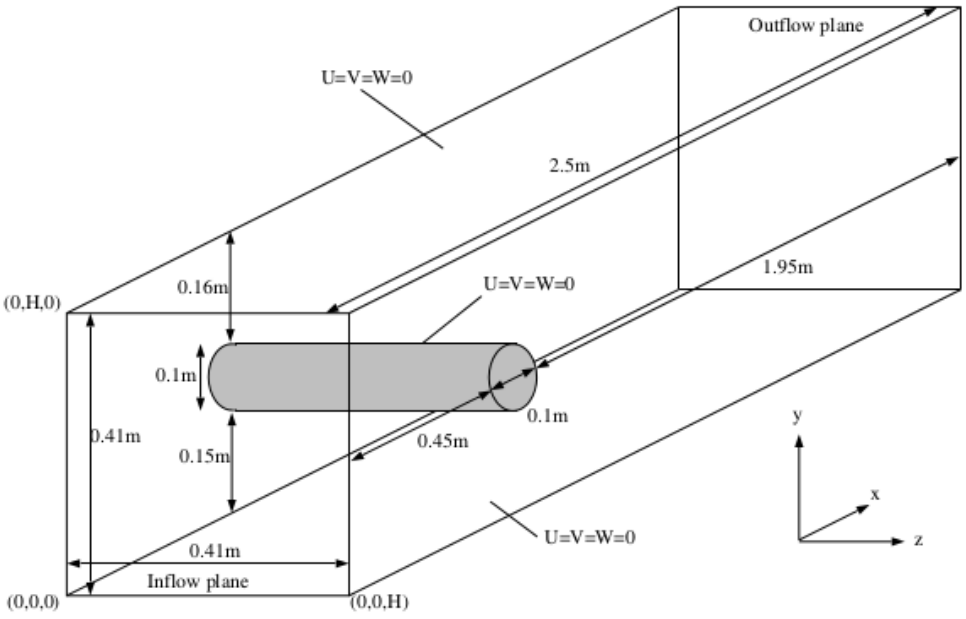

FIGURE 5.2: *Computational domain and boundary conditions.*

solution method and is expected to provide a very good result in a test-case like this. The drag and lift forces on an surface S are given as

$$F_D = \int_S (\rho\nu \frac{\partial v_t}{\partial n} n_y - p n_x) dS \quad , \quad F_L = -\int_S (\rho\nu \frac{\partial v_t}{\partial n} n_x + p n_y) dS. \tag{5.1}$$

Nevne hvordan integralet løses i Nek.

Dersom det var noen probmlemer med rammeverket, nevn det kort!

konsisten eq. bruk!

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

$$c_D = \frac{2F_D}{\rho U^2 DH} \quad , \quad c_L = \frac{2F_L}{\rho U^2 DH}. \tag{5.2}$$

Nek 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 in order to create an object which consists of all the faces on the cylinder. Let x, y be points in the computational domain, x_c, y_c be the coordinates to the center line in the cylinder and $0 < tol \ll 1$ be some user defined tolerance. The faces that belong to the cylinder can then be found by looping over all elements and their faces evaluating $\epsilon = \sqrt{(x - x_c)^2 + (y - y_c)}$. If $\epsilon < tol$ for an entire face then this face is known to belong to the cylinder and is added to the object. Nek also allows the user to specify multiple objects assigning the faces of interest to object 1, object 2 etc. The geometry and mesh for this case was generated in ICEM, and the total number of elements are 2070. For the final calculation polynomial degree $P = 11$ was applied leading to a total of $N = 2755170$.

Initially this case was solved using a second degree polynomial to describe the circle segments corresponding to each element. The mesh around the cylinder is illustrated in figure 5.3. Note that these elements was split in three, in order to obtain a finer mesh in the region of interest. Of the elements numbered in 5.3 only the first six contains edges on the cylinder.Hence the second order polynomials describing the curved edges describe approximately $\Theta = 360°/(6 \cdot 3) = 20°$ of the complete circle. There is no computational reason for not incorporating an order $n = P$ approximation of the circle segments eliminating this approximation error. An algorithm was therefore implemented as an extension to the currently available features regarding curvature on edges. The full description of the algorithm is presented in chapter 5.3. The importance of the error

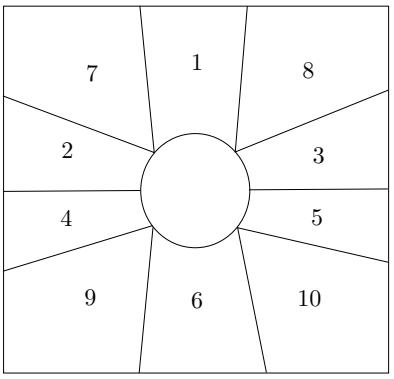

FIGURE 5.3: *Initial mesh around cylinder.*

resulting from the second degree approximation of the circle segments are presented in chapter 6.

5.3 Advances in the mesh-generation routine

The routine `xyzarc()`:

The Gordon Hall algorithm was already implemented as a function in Nek, with the GLL-points,polynomial degree and some initial coordinates to the element. The algorithm creates a distribution of the internal GLL-points in each element. If the element consists of linear edges the only necessary input are the vertices, but by specifying the points on edges and faces the algorithm creates a logical distribution of the internal GLL-points to a deformed element.

The curved edge is specified in the .rea file and the routine `genxyz()` processes the input of each edge. By specifying the radius and the circle center `genxyz()` calls the routine `xyzarc()` which performs the following algorithm;

a, b will be the two end nodes of the edge c will be the mid node, s will be the arc length, θ will be the full angle of the circle sector, cc is the center coordinates, g will be 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
s = r*θ                          ! arc length
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
```

These lines creates the wanted edge curved as a circle sector corresponding to the radius and circle center given. The remaining operation is to call the Gordon Hall algorithm and create the internal GLL-points defined by the edges provided. The figure 5.4 illustrates the geometry on which the algorithm is performed. Notice that the algorithm assumes that the center c is somewhere on the plane defined as all the points with equal distance to both a and b.

- initial script
- changes and modifications
- performance testing
- pitfalls

5.4 Additional projection algorithm

The routine `xyzarc()` enables the user to represent circular edges with the same order as the Lagrange polynomials of each element. For more complex geometry such as actual terrain and other surfaces without any analytic expression the large element sizes makes the geometrical representation difficult. There is however no reason why the GLL-points on a face can be projected onto a non-analytical surface. Although there are no support for automatically reading additional information for distributing the GLL-points exactly

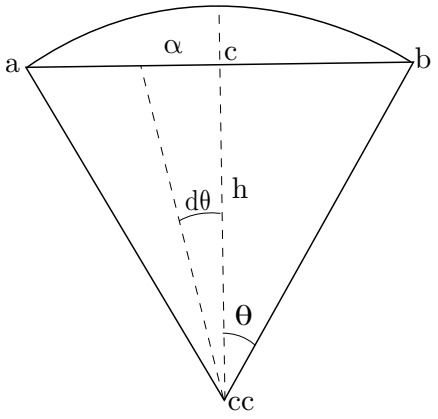

FIGURE 5.4: *A sketch of the curved edge and the variables necessary to calculate the projection*

on a given surface a routine was made which demands a minimum of manual work. The method can be summed up in these simple steps

1. Create initial Mesh and convert to .rea applying nmshconvert.
2. Create refined surface mesh on the non-regular surface.
3. Enable projection by setting param(33) = 1.
4. Choose number of interpolation points by modifying param(34)= (1,2,3)

In addition to the standard Nek library the file surfpro.f needs to be added to the /trunk/nek/ folder along with all the other scripts applied by Nek.

The algorithm relies on two external files generated by the modified nmshconvert script. `surf.i` contains all the coordinated to the points on the refined surface. `bdry.i` contains the element, and face number to all the faces to be projected onto the surface. When generated they automatically alter the local `SIZE` file to contain some key variables applied in the algorithm in surfpro.f. The algorithm is best explained through a simple

box with a non-regular floor. As an initial test-problem the hill of Ekeberg was applied. 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 somewhat larger than EN^2 in order to describe the surface for all the GLL-points that belong to the boundary. The pseudo code for the algorithm is listed below.

```
do e,f in bdry.i
  wrk = create_working_surface(e,f)
  do i in GLL-nodes
    interp = init_interpolation_array()
    do j in wrk
       update_int_array(interp,wrk(j))
    enddo
    set_new_point(interp,wrk,i,e,f)
  enddo
  fix_GLL()
enddo
fix_geom()
```

In order to understand the algorithm a short description of the auxiliary functions is given in the list below

- create_working_surface(e,f) – Loops through all the nodes in surf.i and adds the surface-coordinates within a certain radius to the center GLL-node 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 current surface point to the already existing interpolation points and adds it to the list if it is found to be closer to the initial 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. fix_GLL() distributes all internal GLL-points correctly between the newly projected face and the opposite.
- fix_geom() – An already existing Nek routine which redistributes the GLL-points to assure that the distance between them on the new surface are correct.

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 $O(EN^2(E+N^2))$ and the amount of additional storage space will be of order $O(EN^2+E+N^2) = O(EN^2)$.

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

```
>> /nmshconvert --mesh mesh_myFlow
    --reafile init.rea --outfile myFlow.rea
    --tol 1e3 --temperature True --curvetype A

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

5.4.1 Application on the hill of Ekeberg

As an initial test of the algorithm a mesh was created based on the hill of Ekeberg. The surface was loaded as a `.tin` file in ICEM and a simple box was created with the described terrain as floor. A simple sketch of the domain with the initial mesh in given in figure 5.5. This geometry was chosen because it resembles 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. It is however no theoretical problem to reconstruct the surface with with a polynomial of order P. With the routine described

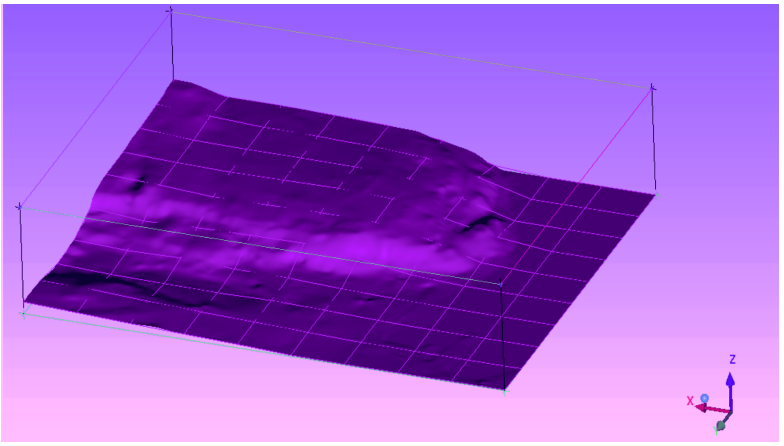

FIGURE 5.5: *The initial mesh of the hill*

in chapter 5.4 The surface was approximated accurately by higher order polynomials. The algorithm restricts itself to relatively smooth surfaces.

5.5 SPATIAL AVERAGING ROUTINE

A dynamic Smagorinsky model has previously been implemented in Nek5000 for flow in a channel. The method as described in chapter 2 depends on an averaging routine to calculate the dynamic Smagorinsky constant. The implementation in Nek applies an average routine in the plane, assuming that the Smagorinsky constant is the same for all points with equal distance to the walls of the channel. This is a rather specific averaging routine only applicable to channel flows.

When applying dynamical Smagorinsky to Case 1 a new spatial mean routine had to be applied. It was first attempted to average only in time, but this proved not to be sufficient. It was therefore implemented a simple routine for taking the average within each element, let c^e_{num}, c^e_{den} denote the numerator and the denominator in equation 2.24. The means are then calculated as

$$c^e_{num} = \frac{1}{V}\int_{\Omega_e} c^e_{num}\, d\,\Omega = \frac{1}{V}\sum_{i=1}^{N^3} \rho_{i,e} c^e_{num,i}. \tag{5.3}$$

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

Results

6.1 Drag and lift on a cylinder

The effect of the implemented algorithm in Nek5000 explained in Chapter 5 is illustrated by solving a laminar flow test problem. The solution is compared with previously benchmark computations performed by a number of contributors [15]. The test problem considered is steady flow with Re=20 in a rectangular channel past a cylinder. The drag and lift coefficients on the cylinder are calculated and compared to a pair of reference values. The results can be found in table 6.1. As the results clearly show the treatment of the geometry is crucial, both coefficients are computed with significantly better accuracy. The polynomial degree used in the calculations with Nek5000 was chosen as $p = 11$ in all directions. The explicit number of cells is therefore $2070 \cdot 11^3 = 2755170$, approximately 88% of the number of cells used for the benchmark simulations. Compared with the results from the other softwares applied in [15] Nek5000 performs just as well or better in most cases. It should be mentioned that the division of the grid is

# of Cells	Software	c_D	c_L	%**Err** c_D	%**Err** c_L
2070	Nek5000 (mid)	6.18349	0.008939	0.030	4.19
2070	Nek5000 (arc)	6.18498	0.009413	0.006	0.13
3145728	CFX	6.18287	0.009387	0.04	0.15
3145728	OF	6.18931	0.00973	0.06	3.5
3145728	FEATFLOW	6.18465	0.009397	0.01	0.05

Table 6.1: *Results for the drag and lift coefficients with reference values* $c_D = 6.18533$ *and* $c_L = 0.009401$.

done differently for Nek5000 so the comparison is not as direct as it may seem from the table.

add more info about the mesh for this case??

6.1.1 Parameter adjustments in Nek5000

As discussed in chapter 4 there are many adjustments available in Nek. In order to enlighten the actual effect on the results, several different settings have been investigated and the results are presented. All simulations are performed as transient flows and the data collected in table 6.2 are gathered when the flow has reached a steady state solution.

Perform the different tests and show the difference.

	Settings			% Error		Data		
#	Aliasing	IOFS	Filter	c_D	c_L	DT	CFL	T/Tstep
1	No	No	No	1.0	2.3	1e-04	2.03	2.1e-02

Table 6.2: *Performance data for different settings in Nek5000*

6.2 Gas dispersion in a simplified urban area

This case is a part of a larger project designed to evaluate different solvers ability to perform simulations of gas dispersion. An initial attempt to simulate the dispersion of gas was performed in Nek5000 with fairly coarse element sizes, but without any subgrid-scale model. Only applying filtering on the last 3 nodes made the solution sufficiently stable and the results can be seen in figures 6.1 and 6.1. Discuss the plot...

redo these simulation in case they were started to early.

6.2.1 Dynamic Smagorinsky

Applying a subgrid scale model to the problem should imply a more accurate result since the grid is not sufficiently fine to resolve the finest eddies.

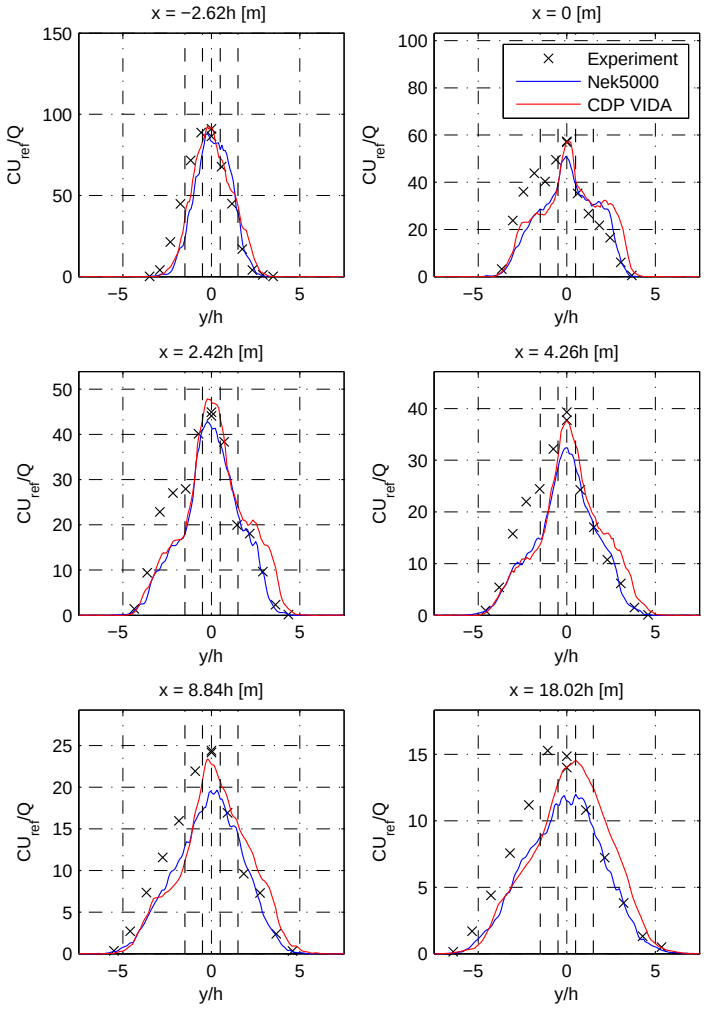

FIGURE 6.1: *Time-averaged concentration with a sample time of* 18.00 *s at* $z/H = 0.025$ *plotted horizontally and scaled with the free-stream velocity and emission rate. Compared against wind tunnel data. Two dashed lines on either side of the centerline represent the canyon.*

6.3 Discussion and Conclusion

How did Nek perform overall, user-friendly ?,correctness,speed etc.

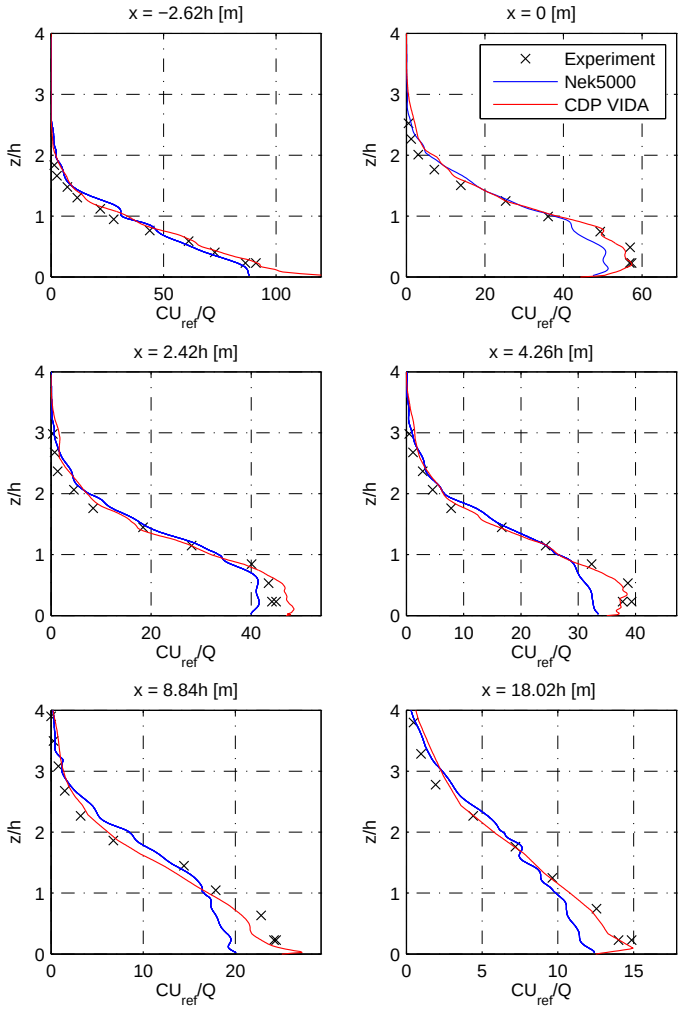

FIGURE 6.2: *Time-averaged concentration with a sample time of* 18.00 *s at* $y = 0$ *plotted vertically and scaled with the free-stream velocity and emission rate. Compared against wind tunnel data. Two dashed lines on either side of the centerline represent the canyon.*

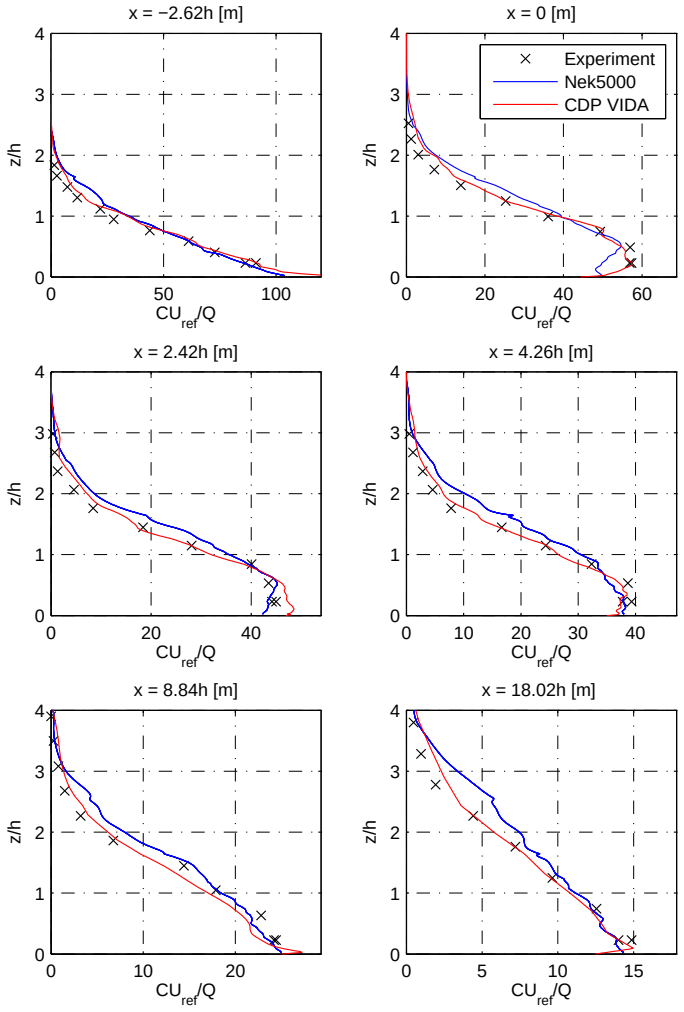

FIGURE 6.3: *Time-averaged concentration with a sample time of* 22.00 *s at* $y = 0$ *plotted vertically and scaled with the free-stream velocity and emission rate. Compared against wind tunnel data. Two dashed lines on either side of the centerline represent the canyon.*

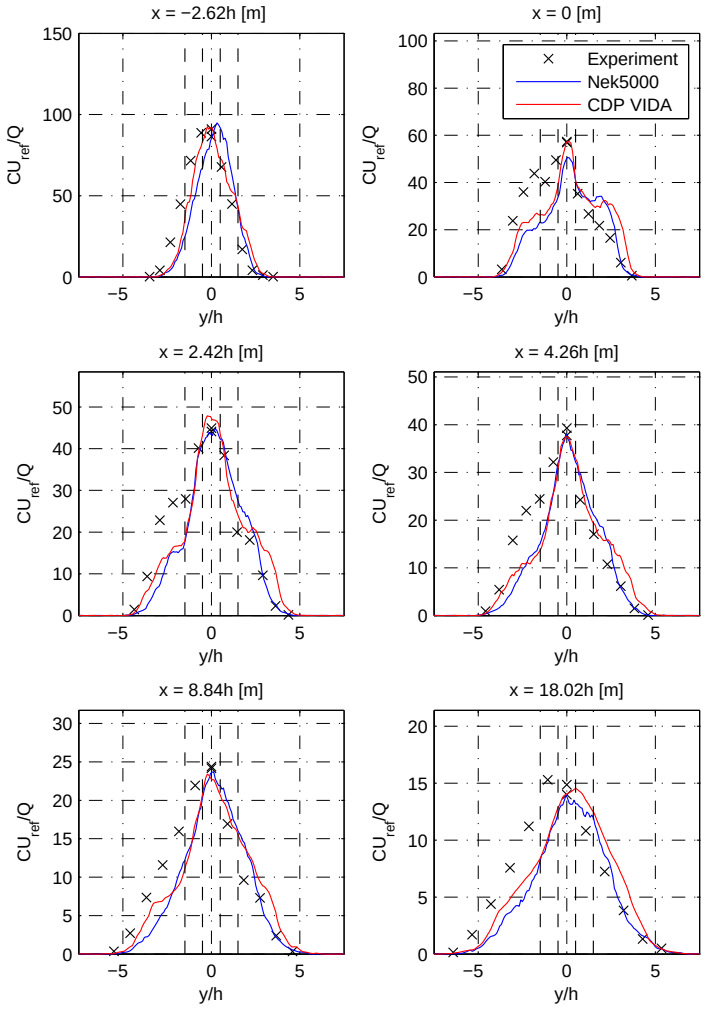

FIGURE 6.4: *Time-averaged concentration with a sample time of* 22.00 *s at* $y = 0$ *plotted vertically and scaled with the free-stream velocity and emission rate. Compared against wind tunnel data. Two dashed lines on either side of the centerline represent the canyon.*

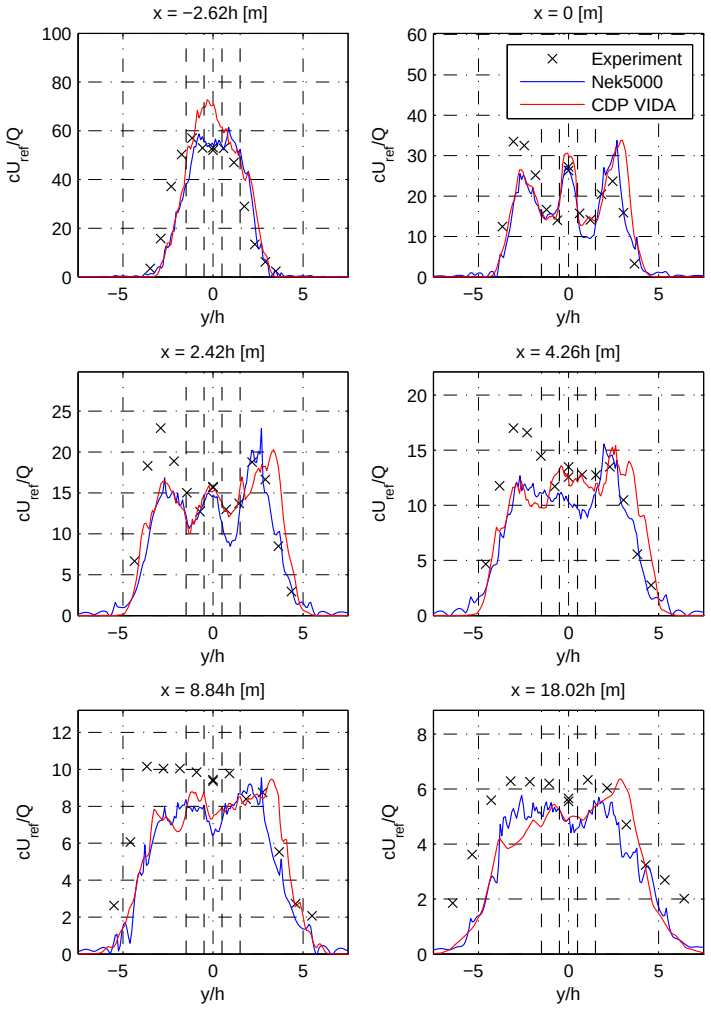

FIGURE 6.5: *Time-averaged concentration with a sample time of* 22.00 *s at* $y = 0$ *plotted vertically and scaled with the free-stream velocity and emission rate. Compared against wind tunnel data. Two dashed lines on either side of the centerline represent the canyon.*

Appendix A

Fundamental basics of numerical analysis

A.1 GLL-quadrature

A.2 Essential polynomials

1. Legendre polynomials
2. Lagrange basis

A.3 Preliminary concepts

1. coersiveness
2. Bounded

A.4 Lax-Milgram theorem

Stating the theroem without proof

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. Below is a list of some of the variables that have been frequently used in .usr subroutines which initially are defined outside of both 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
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 frecuency
time	Current time
tstep	Current timestep
dt	Timestep
dtlag(10)	The preevious 10 timesteps
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
WZF	
zgl(lx1)	Gauss lobatto points
wgl(lx1)	Gauss lobatto weights
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 seperate 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 lenght 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 lenght nx1.

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

gh_face_extend(x,zg,n,type,e,v) The Gordon hall algorithm described in chapter **??**, 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,e,ifaxl) Generate bi- or trilinear mesh.

fix_geom() Routine for re distributing the gll-points corrctly on the updated geometry.

B.2.3 Additional auxiliary functions implemented for this thesis

fix_gll(e,f) Redistribute the gll-points between the given face and the opposite to make sure that all points lie within the element.

getface(kx1,kx2,ky1,ky2,kz1,kz2,wrk,n,e) assigning the values of the face in element e corresponding to the index limits kx1,kx2... to the array wrk of size 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 from the initial gll-point to a given point on the surface.

interp_up(iinterp,rinterp,error,k) Update interpolation points

set_new_pt(iinterp,rinterp,ix,iy,iz,e) 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

Bibliography

[1] Frank 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. 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.

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

[4] Y. Maday, AnthonyT. Patera, and EinarM. 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.

[5] Alfio Quarteroni. *Numerical Models for Differential Problems, 2. edition*. Springer, 2014.

[6] George Karniadakis and Spencer Sherwin. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press, 2013.

[7] C. Canuto, M.Y Hussaini, and A. Quarteroni et al. *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.

[8] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*. Cambridge, 2006.

[9] Paul Fischer and Julia 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.

[10] 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.

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

[12] 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.

[13] *NEK5000 documentation*, 2015.

[14] Daniel Eriksson. Neutrally buoyant gas dispersion within an urban street canyon. *MekIT'15.*, 2015.

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