Newton's Method for a Finite Element Approach to the Incompressible Navier-Stokes Equations

Michael Brandl

June 10, 2016

Bachelor's Thesis in Applied Mathematics, 15 ECTS credits

Supervisor: Mats G. Larson
Examiner: Leif Persson

Umeå University
Department of Mathematics and Mathematical Statistics
SE-901 87 UMEÅ
SWEDEN

Abstract

The cG(1)cG(1)-method is a finite element method for solving the incompressible Navier-Stokes equations, using a splitting scheme and fixed-point iteration to resolve the nonlinear term $u \cdot \nabla u$.

In this thesis, Newton's method has been implemented on a formulation of the cG(1)cG(1)-method without splitting, resulting in equal results for the velocity and pressure computation, but higher computation times and slower convergence.

Sammanfattning

cG(1)cG(1)-metoden är en finita elementmetod för att lösa de inkompressibla Navier-Stokes ekvationerna genom att använda splittning och fixpunktiteration för att hantera den ickelinjära termen $u \cdot \nabla u$.

I detta examensarbete har Newtons metod implementerats på en formulering av cG(1)cG(1) utan splittning, vilket resulterar i lika resultat för hastighets- och tryckberäkningen, men högre beräkningstid och långsammare konvergens.

Contents

List of Figures

List of Tables

List of Algorithms

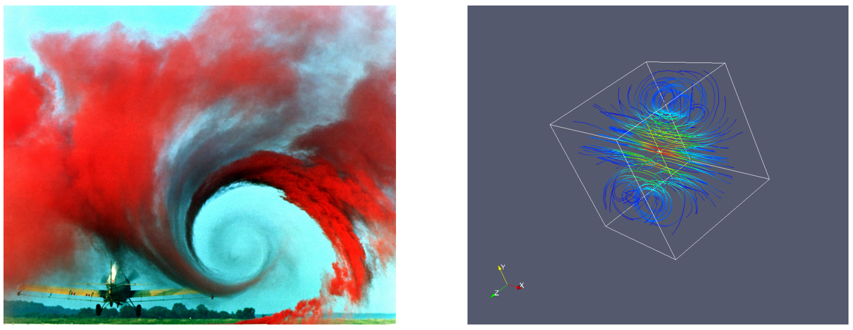
Chapter 1

Introduction

1.1 General Introduction

The field of *Fluid Dynamics* within physics deals with the flow of gases and liquids. The *Navier-Stokes equations* can be used to describe their motion, given boundary conditions, external forces and parameters such as viscosity of the fluid.

Examples of interesting use cases in science and engineering include flow of water around a ship hull, or flow of air around a car or airplane wing. Here, the Navier-Stokes equations capture the flow regime of turbulence, a seemingly chaotic flow pattern e.g. seen by smoke in the air, and also leading to phenomena such as eddies and vortices. Its effects are important in real-world cases such as wind-resistance of a vehicle or flight properties of an airplane (see figure 1.1 for an example).



(a) Landing agricultural airplane's wake vortex, visualized by colored smoke rising from the ground.

Source: [21]. Public Domain.

(b) Visualization of FEM-based computer simulation of turbulent flow, showing velocity streamlines.

Figure 1.1: Turbulence in real-world example and simulation.

This thesis will focus on the *incompressible Navier-Stokes equations* (introduced in more detail in section 2.1), which describe the motion of incompressible flows, such as of incompressible fluids such as water.

In most cases, the incompressible Navier-Stokes equations cannot be solved analytically. Instead, within the branch of *Computational Fluid Dynamics*, the problem is discretized in both space and time, and then solved numerically using computers. Computer simulation allows also for testing of virtual prototypes of machines and vehicles. Depending on the size of the problem and the desired accuracy, computations can involve many processors and take long time to complete.

One way to discretize the problem is to reformulate it in a weak formulation by using the *Finite Element Method*. Here, a non-linear term within the incompressible Navier-Stokes equations poses a challenge. One can approach the non-linearity using splitting schemes (see [27] for an example), or use techniques for solving non-linear equations, such as *fixed-point iteration* (also called *Picard iteration*) or *Newton's Method*. Also, stabilization methods are applied in order to make the discretized problem numerically stable.

One method involving stabilization is the cG(1)cG(1)-method introduced by Eriksson ([12, chapter 86.5]). This thesis will analyze an existing fixed-point iteration scheme for resolving the non-linear terms, and try to improve on it by introducing an application of Newton's method.

The goal is to be able to obtain higher accuracy in the solution, and to reduce computation time.

1.2 Summary of Main Results

Newton's method for cG(1)cG(1) has been derived and implemented as a modification to an existing C++ code for fixed-point iteration for cG(1)cG(1) with splitting, and experiments have been run on both variants of the code.

The experiments show that the results for velocity and pressure agree well in both methods.

However, for Newton's Method, quadratical convergence was not achieved, and it lead to a large increase in computation time compared to fixed-point iteration with splitting.

1.3 Previous Work

For an overview of the field of fluid dynamics, see [3]. For a derivation of the Navier-Stokes equations, see p. 147 there.

Larson and Bengzon [20] give an overview of the finite element method, also showing the use of Newton's method and treating the Navier-Stokes equations using the finite element method.

Eriksson et al. [12] provide an introduction of approaches to solve the Navier-Stokes equations using the finite element method. This thesis will follow their general approach.

Edwards et al. [8] apply Newton's method for computing the steady state in the Navier-Stokes equations.

Chapon applies in his PhD-thesis [6] Newton's method on the Navier-Stokes equations. Solver implementations, preconditioning and h- and r-refinement of the mesh are treated. The problem is only solved in 2D-space, though.

Tang et al. [31] solve the Navier-Stokes equations using Newton's method to linearize the non-linear terms in order to solve lid-driven cavity flows. For the same application, Zunic et al. [33] use a combination of the finite element and boundary element method.

More recently, Elman et. al.[10] summarize the current status of solving Navier-Stokes equations using finite element method and discuss strategies involving convergence of Newton's method and Picard iteration.

Several methods of improving run-time, and sometimes convergence, have been considered. Clift and Forsyth [7] investigate the possibility to avoid a full update of the Newton matrix when solving the Navier-Stokes equations by freezing coefficients in order to improve diagonal dominance. Orkwis [24] compares performance of Newton's and Quasi-Newton's method. Persson [25] uses line-based discontinuous Galerkin and Quasi-Newton's with an implicit stepping method in order to increase sparsity in the matrix and thus improve run-time performance. Shadid et al. [28] use an inexact Newton's method, while using backtracking for global convergence. Sheu and Lin [29] propose a variation of the Newton-based linearization, increasing both run-time and convergence. Selim et al. [27] investigate adaptive resolution for a method splitting pressure and velocity solve.

There has been work on finding good preconditioners for this problem. Kim et al. [18] show that a solution to Stokes equations can under certain circumstances be used as a preconditioner to the Navier-Stokes equations. Elman et al. [9] investigate a number of preconditioning techniques for Newton's method, developed earlier [11] for the Oseen solver, for the incompressible Navier-Stokes equations.

Pulliam [26] analyzes the time accuracy when using implicit time-stepping methods and Newton's method.

Bengzon introduces in his PhD-thesis [4] a priori and a posteriori error estimates that can be used for local mesh refinement for the finite element method, and applies this to several application domains, amongst others fluid dynamics. Parts of these results are used by Olofsson's master's thesis [23] for a parallel implementation of a finite element solution for a coupled fluid-solid system. Parts of Olofsson's code and work-flow were reused in the present thesis.

1.4 Structure

The remaining part of the thesis is structured as follows:

- Chapter 2 introduces theory and background by first defining the incompressible Navier-Stokes equations and then the general Finite Element Method. These concepts are then used to introduce the cG(1)dG(0)-method and its derivate, the cG(1)cG(1)-method, which will be used in the thesis. Then, the concepts of fixed-point iteration (with its application to the cG(1)cG(1)-method) and Newton's method will be introduced.

- Chapter 3 elaborates on the algorithms and implementations used, by applying Newton's method on the cG(1)cG(1)-method, and giving details on the technical implementation and hardware.

- Chapter 4 provides results of the implementation, which stem from applying the implementation on the benchmark test case of the lid-driven cavity. The velocity- and pressure-solutions resulting from the fixed-point and Newton's method will be compared, as well as computation time and convergence, and the results will be discussed.

- Chapter 5 summarizes achievements and shortcomings, and presents possible future work.

Chapter 2

Theory

2.1 The Incompressible Navier-Stokes Equations

The Navier-Stokes equations describe fluid dynamics due to convection and diffusion. An important paramters is the kinematic viscosity of the fluid, which describes the fluid's resistance to shear stress. Fluids with low viscosity under rapid movement tend to exhibit turbulence. The dimensionless *Reynolds number* is a measure for a fluid's tendency to develop turbulent flow, and is defined as $Re = UL/\nu$, with ν being the kinematic viscosity, U a representative velocity, and L a representative length scale.

This thesis will only consider the incompressible Navier-Stokes equations with constant temperature, density and kinematic viscosity, as in [12, page 1166]. They can be formulated in the following matter (see [20, Chapter 12.3]):

Let $\nu > 0$ be the constant kinematic viscosity of a fluid with unit density and constant temperature which is enclosed in a volume $\Omega \subset \mathbb{R}^3$ with boundary Γ.
The goal is then to determine the velocity $u = (u_1, u_2, u_3) : \Omega \times I \to \mathbb{R}^3$ and the pressure $p \colon \Omega \times I \to \mathbb{R}$ of the fluid for a given driving force $f \colon \Omega \times I \to \mathbb{R}^3$ with

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \Delta u + \nabla p = f \qquad \text{in} \quad \Omega \times I, \tag{2.1a}$$

$$\nabla \cdot u = 0 \qquad \text{in} \quad \Omega \times I, \tag{2.1b}$$

$$u = g_D \qquad \text{on} \quad \Gamma_D \times I, \tag{2.1c}$$

$$\nu n \cdot \nabla u - pn = 0 \qquad \text{on} \quad \Gamma_N \times I, \tag{2.1d}$$

$$u(\cdot, 0) = u^0 \qquad \text{in} \quad \Omega, \tag{2.1e}$$

where u^0 is the initial velocity and $I = (0, T)$ the time interval, and the following abbreviation is used: $u \cdot \nabla u = (u \cdot \nabla) u$. The boundary Γ is divided into the regions Γ_D with *no-slip* boundary conditions, where g_D is a constant velocity, and Γ_N with *natural* (also called *do-nothing*) boundary conditions, where n is the boundary normal.

Boundary Conditions

In **Partial Differential Equations** (PDEs), boundary conditions describe the value of the functions on the boundary of the domain. Within fluid dynamics, the following types of boundary conditions are common (see [20, Page 292] for a more thorough treatment):

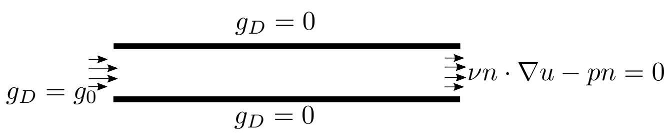


Figure 2.1: Channel with different boundary conditions on inflow, walls and outflow. Arrows indicate velocity.

- Dirichlet (also called **no-slip**) boundary conditions are defined as $u = g_D$ on Γ_D. They can be used to model interaction with solid geometry such as pipe walls. Non-zero Dirichlet boundary conditions can be used to model in-flow velocity.
- **Natural**, or **do-nothing** boundary conditions are used to model flow out of an unbounded domain, e.g. to truncate a long channel. When using variational methods such as the Finite Element Method, the do-nothing boundary condition is automatically derived if no other boundary conditions are enforced. However, there can be issues with stability (see [5] for more details).

Together with parameters such as viscosity and external forces, variations in the geometry and boundary conditions make it possible to use the incompressible Navier-Stokes equations to model fluid dynamics in various scenarios.

One example would be a channel section with $g_D = 0$ on the walls, $g_D = g_0$ for the given inflow velocity, and Γ_N being the outflow region (see picture 2.1). Another one is the so-called lid-driven cavity, which will be investigated in section 4.1.

Non-Linearity

Note that the term $u \cdot \nabla u$ introduces a non-linearity. This non-linearity makes it possible to model effects such as turbulent flow, but it makes it also hard to solve the incompressible Navier-Stokes equations. Analytical solutions are only known for some special cases. The question if an analytical solution exists for the general case is not answered yet and is one of the Millennium Prize Problems[13].

For practical purposes, numerical methods are usually applied for solving the incompressible Navier-Stokes equations, using computers.

One such numerical method is the **Finite Element Method** (FEM).

2.2 The Finite Element Method

Here, the Finite Element Method will be introduced very briefly by following the presentation by Bengzon [4, chapter 3]. For a more thorough introduction, see [20].

2.2.1 Weak Form

The Finite Element Method is a technique for numerically finding approximative solutions to differential equations. The general approach is to reformulate the differential equation, or *strong form*, in a so-called *weak form*, by choosing a space of test functions and integrating by parts, thus obtaining a variational equation.

Depending on the choice of the space of test functions, the weak form will be easier to solve, and if the domain Ω and the coefficients of the partial differential equation are sufficiently regular, it can usually be shown that a solution to the weak form is also a solution to the strong one [4, page 14].

Assume the differential equation is given as

$$Lu = f, \quad \text{in} \quad \Omega \tag{2.2a}$$

$$u = 0, \quad \text{on} \quad \delta\Omega \tag{2.2b}$$

with L a linear differential operator, and f a given function.

From this strong form, the weak form is obtained by multiplying equation (2.2a) with a test function v which is zero on the boundary $\delta\Omega$, and then integrating over Ω, which gives

$$\int_\Omega Lu \cdot v dx = \int_\Omega f \cdot v dx \tag{2.3}$$

This can be written in short-hand as

$$(Lu, v) = (f, v) \tag{2.4}$$

with a slight abuse of notation.

The equation holds as long as the involved integrals exist. Let V be the function space of all test functions v for which this is true. Note that also $u \in V$ due to the boundary conditions.

Introducing further the notation

$$a(u, v) = (Lu, v) \tag{2.5a}$$

$$l(u) = (f, v), \tag{2.5b}$$

the weak form can be formally stated as follows:

Find $u \in V$ such that

$$a(u, v) = l(v), \quad \forall v \in V. \tag{2.6}$$

The definition of $a(u, v$ and $l(u)$ depends on the original strong problem.

The existence and uniqueness of the solution u of the weak form is given under conditions such as coercivity, inf-sup stability, continuity of a and l via the Lax-Milgram-Lemma. See [20, chapter 7.3] for derivation and proof.

2.2.2 Tessellation

From a weak form, a finite element method is obtained by replacing the infinite vector space V by a finite dimensional subspace $V_h \subset V$, often the space of piecewise polynomials (up to a certain given degree).

The computational domain is tessellated into a mesh $\kappa = \{K\}$ which is defined as a set of geometrical simplices K such as triangles in 2D (see e.g. figure 2.2) or tetrahedra in 3D. The resolution of the mesh influences the quality of the approximation.

For measuring the size of a triangle or tetrahedron K, let h_K be its longest side. One way of measuring the quality of K is by defining its chunkiness parameter $c_K = \frac{s_K}{h_K}$, where s_K is K's shortest side. Another way to define the chunkiness parameter is $c_K = \frac{h_K}{d_K}$, where d_K is the diameter of the triangle's inscribed circle/the tetrahedron's inscribed sphere [20, page 46]. A tesselation κ is called *shape-regular* if $\exists c_0 : c_K > c_0 \forall K$. Shape-regularity is a quality measure of a tesselation, and can be used in several theorems and proofs with regard to error estimations (e.g. [20, chapter 3.3.1]).

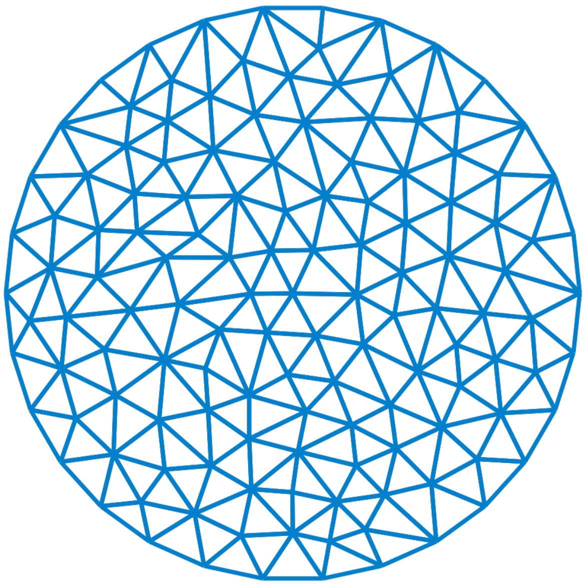


Figure 2.2: Triangulation of circle for FEM using Matlab.

Source: Oleg Alexandrov, via [1]. Public Domain.

Given a tessellation κ, on each $K \in \kappa$ a function space is defined together with a set of functionals, giving a certain degree of continuity at the borders between two adjacent Ks.

A finite element is then defined by the triplet of simplex, polynomial space, and functionals.

The problem is further reduced by letting the approximation u_h for the sought-for function u also lie in the constructed vector space V_h. It is then possible to find basis functions $\varphi_i, i = 1, \ldots, n$ for this function space due to its finite nature. As an example, let V_h be the space of continuous, piecewise linear functions on the 1D-mesh κ. Then, a basis can be formed by hat-functions which have value 1 on their center-node and 0 on all others, such as in figure 2.3.

2.2.3 Computing the Approximate Solution

Given the space V_h, and a basis $\{\varphi_i\}$.

A finite element version of equation (2.6) can be formulated as: Find $u_h \in V_h$ such that

$$a(u_h, v_h) = l(v_h), \quad \forall v_h \in V_h. \tag{2.7}$$

Since $\{\varphi_i\}$ is a basis of V_h, this is equivalent to

$$a(u_h, \varphi_i) = l(\varphi_i), \quad i = 1, \ldots, n. \tag{2.8}$$

Also, the approximate solution $u_h \in V_h$ can be represented as $u_h = \sum_{i=1}^{n} \varphi_i \xi_i$, with n unknowns $\xi_i, i = 1, \ldots, n$.

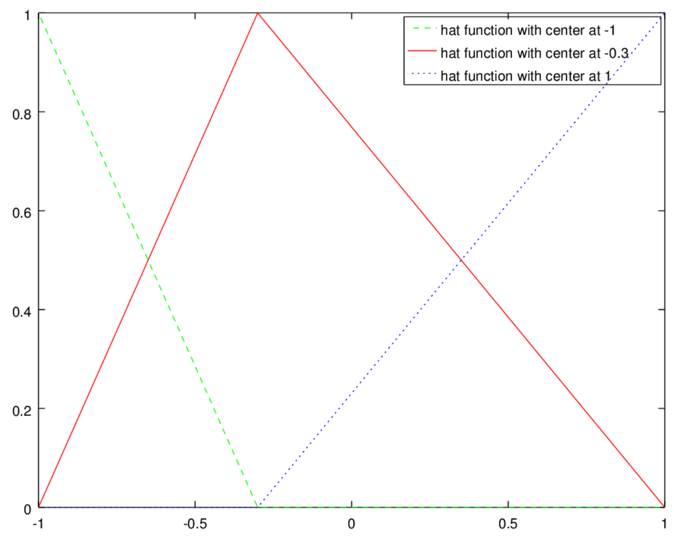


Figure 2.3: Hat functions on 1D domain, with center at -1, -0.3 and 1, respectively, which together form a basis for the space of continuous piecewise linear functions on the mesh {-1,-0.3,1}.

Inserting this into equation (2.8), one obtains

$$\sum_{j=1}^{n} \xi_j a(\varphi_j, \varphi_i) = l(\varphi_i), \quad i = 1, \ldots, n. \tag{2.9}$$

This can be written in matrix form as

$$A\xi = L, \tag{2.10}$$

where $A_{ij} = a(\varphi_i, \varphi_j)$ and $L_i = l(\varphi_i)$ are constant and can be computed exactly, or approximated by means of numerical integration.

Solving this linear system of equations gives the coefficients for constructing u_h, since $u_h = \sum_{i=1}^{n} \varphi_i \xi_i$.

The matrix A is usually large and sparse, which makes the use of dedicated sparse linear algebra software libraries advisable.

2.2.4 Further Notes

For numerical stability, it might be necessary to add stabilization terms to the weak form, such as least square-stabilization. However, these extra terms have to be chosen carefully in order not to compromise the correctness of the approximate solution.

The latter can also be influenced by the resolution of the tessellation, as well as the choice of the finite element.

2.3 The cG(1)dG(0) Method

One way of formulating a weak form of the incompressible Navier-Stokes equations using the Finite Element Method is the cG(1)dG(0)-method (see [12, Page 1168]), which will be derived below. Unless otherwise stated, the presentation will follow [12].

Consider the Navier-Stokes equations as defined in equations (2.1). Assume homogeneous Dirichlet boundary conditions (the case of Neumann boundary conditions is treated in [12, Page 1170]). Let the space Ω be discretized by a triangulation $T_h = \{K\}$ of Ω, with a mesh function $h(x)$. Let W_h be a finite element space of continuous piecewise linear functions on $\{K\}$. The continuous piecewise linearity of the space-approximation gives the method the first part of its name: *cG(1)*.

2.3.1 Time-discretization

The time-dependence in the term $\frac{\partial u}{\partial t}$ in equations (2.1) makes it necessary to also discretize time. Let $0 = t_0 < t_1 < \cdots < t_N = T$ be a sequence of discrete time levels, with time steps $k_n = t_n - t_{n-1}, 1 \leq n \leq N$. Let the approximations $U(x,t)$ of velocity u and $P(x,t)$ of pressure p be piecewise constant and discrete in t, which gives the method the second part of its name: *dG(0)*. Thus, for each $n = 1, \ldots, N$ one seeks $U^n \in V_h^0$, with $V_h^0 = W_h^0 \times W_h^0 \times W_h^0$, and $P^n \in W_h$. Define

$$\begin{aligned} & U(x,t)^n(x), \quad x \in \Omega, \quad t \in (t_{n-1}, t_n], \\ \text{and} \quad & P(x,t)P^n(x), \quad x \in \Omega, \quad t \in (t_{n-1}, t_n]. \end{aligned}$$

The term $\frac{\partial u}{\partial t}$ can be discretized as $\frac{U^n - U^{n-1}}{k_n}$. By choosing all other occurring u-terms to be U^n, the p-terms to be P^n, and modifying the equations (2.1a) and (2.1b) accordingly, multiplying with a test function $v \in V_h^0$ and $q \in W_h$ respectively, taking the integral over the domain Ω, and integrating the diffusion-term by parts, one obtains

$$\left(\frac{U^n - U^{n-1}}{k_n}, v \right) + (U^n \cdot \nabla U^n + \nabla P^n, v) + \nu(\nabla U^n, \nabla v) = (f^n, v) \qquad \forall v \in V_h^0, \tag{2.11a}$$

$$(\nabla \cdot U^n, q) = 0 \qquad \forall q \in W_h, \tag{2.11b}$$

where $U^0 = u^0$ and $f^n = f(x, t_n)$.

The cG(1)dG(0)-method without stabilization is defined thus: For each $n = 1, \ldots N$, find $(U^n, P^n) \in V_h^0 \times W_h$ which fulfill the system of equations (2.11a) and (2.11b).

2.3.2 Relation to Backward Euler

Note that the time discretization scheme used above corresponds to backward Euler (also called *implicit Euler*), an implicit numerical method for solving ordinary differential equations.

For a given ordinary differential equation

$$\frac{dy}{dt} = f(t, y), \tag{2.12}$$

backward Euler gives the following scheme

$$\frac{y^n - y^{n-1}}{h} = f(t^n, y^n). \tag{2.13}$$

Using the values of the new time step in the right-hand side gives better numerical stability than the obvious alternative of using the old ones (as in *forward Euler*), but introduces the necessity to solve for the unknown y^n, and introduces artificial damping. The truncation error is $O(h^2)$, which can be seen by deriving backward Euler from a Taylor-expansion and truncating at the second order term.

2.3.3 Stabilization

From the equations (2.11a) and (2.11b), the stabilized version of the cG(1)dG(0)-method is obtained by adding the two equations, and introducing stabilization by replacing v with $v + \delta(U^n \cdot \nabla v + \nabla q)$ in the terms $(U^n \cdot \nabla U^n + \nabla P^n, v)$ and (f^n, v):

$$\begin{aligned}\left(\frac{U^n - U^{n-1}}{k_n}, v\right) + (U^n \cdot \nabla U^n + \nabla P^n, v + \delta(U^n \cdot \nabla v + \nabla q)) + (\nabla \cdot U^n, q) + \nu(\nabla U^n, \nabla v) \\ = (f^n, v + \delta(U^n \cdot \nabla v + \nabla q)) \quad \forall (v,q) \in V_h^0 \times W_h,\end{aligned} \tag{2.14}$$

where δ is defined as

- $\delta(x) = h^2(x)$ if $\nu \geq Uh$ (*diffusion-dominated flow*), or
- $\delta = (\frac{1}{k} + \frac{U}{h})^{-1}$ otherwise (*convection-dominated flow*).

For more background on the reasoning for the introduction of the stabilization terms, see [12, page 1169].

2.4 The cG(1)cG(1) Method

In the cG(1)cG(1)-method, a higher-order approximation of the time-component is used. Replacing all U but the ones in the first term by $\hat{U}$ where $\hat{U} = \frac{1}{2}(U^n + U^{n-1})$, one obtains

$$\begin{aligned}\left(\frac{U^n - U^{n-1}}{k_n}, v\right) + (\hat{U}^n \cdot \nabla \hat{U}^n + \nabla P^n, v + \delta(\hat{U}^n \cdot \nabla v + \nabla q)) + (\nabla \cdot \hat{U}^n, q) + \nu(\nabla \hat{U}^n, \nabla v) \\ = (f^n, v + \delta(\hat{U}^n \cdot \nabla v + \nabla q)) \quad \forall (v,q) \in V_h^0 \times W_h.\end{aligned} \tag{2.15}$$

2.4.1 Relation to Implicit Midpoint

This time, the time discretization scheme does not correspond to backward Euler, but instead to the implicit Midpoint method, another implicit numerical method for solving ordinary differential equations.

For a given ordinary differential equation

$$\frac{dy}{dt} = f(t, y), \tag{2.16}$$

the implicit Midpoint Method gives the following scheme

$$\frac{y^n - y^{n-1}}{h} = f(t^{n-1} + \frac{h}{2}, \frac{1}{2}(y^n + y^{n-1})). \tag{2.17}$$

Implicit Midpoint makes it necessary to do slightly more computations than backward Euler. However, the symmetry lets even-ordered error terms cancel, leading to a truncation error of $O(h^3)$ and thus higher accuracy (similar derivation to backward Euler).

This method will be used for simulating the incompressible Navier-Stokes equations in this thesis. However, the difficulty of resolving the non-linear term remains.

2.4.2 Modified cG(1)cG(1) with Least Squares Stabilization

As a derivation of the cG(1)cG(1) method, a least squares stabilization term will be added to equation (2.15) for increased numerical stability:

$$\begin{aligned}\left(\frac{U^n - U^{n-1}}{k_n}, v\right) + (\hat{U}^n \cdot \nabla \hat{U}^n + \nabla P^n, v + \delta_1(\hat{U}^n \cdot \nabla v + \nabla q)) + (\nabla \cdot \hat{U}^n, q) + \nu(\nabla \hat{U}^n, \nabla v) \\ + \delta_2(\nabla \cdot \hat{U}^n, \nabla \cdot v) = (f^n, v + \delta_1(\hat{U}^n \cdot \nabla v + \nabla q)) \quad \forall (v, q) \in V_h^0 \times W_h.\end{aligned} \tag{2.18}$$

Note that the original stabilization factor δ is now called δ_1, and the new term has a factor δ_2.

2.5 Fixed-Point Iteration

2.5.1 General Introduction

One simple technique for solving non-linear equations is *Fixed-Point Iteration* (also called *Picard Iteration*). The presentation in this section follows [20, chapter 9.1].

Let

$$g \colon X \to X, \quad y = g(x) \tag{2.19}$$

be a non-linear function of the single variable $x \in X$.

One tries to find a fixed-point of g, i.e. a $\bar{x}$ for which $g(\bar{x}) = \bar{x}$ by starting with an initial guess x_0, and creating a sequence by letting

$$x_{i+1} = g(x_i). \tag{2.20}$$

The sequence is then hoped to converge to a fixed-point $\bar{x} = g(\bar{x})$.

This gives rise to the algorithm

Algorithm 1: Fixed-point iteration for a non-linear function

```
Choose initial starting guess $x_0$, and maximum desired increment size $\epsilon$;
for $k = 0, 1, 2, \dots$ do
    $x_{k+1} = g(x_k)$;
    if $|x_{k+1} - x_k| < \epsilon$ then
        Stop.
    end
end
```

Here, the size of the increment was chosen as a termination criteria; other ones might be considered, such as the residual size.

The Banach fixed-point theorem states that this approach converges if X is a metric space (with metric d) and g is a *contraction mapping*, i.e.

$$\exists L \in [0,1)\colon d(g(x), g(y)) \leq L d(x,y) \quad \forall x, y \in X. \tag{2.21}$$

Note that L is called a *Lipschitz constant* of g.

If X is a not only a metric space, but also a normed space, this simplifies to

$$\exists L \in [0,1)\colon \|g(x) - g(y)\| \leq L\|x - y\| \quad \forall x, y \in X. \tag{2.22}$$

As a proof for the latter version, note that $\bar{x} = g(\bar{x})$. Subtracting that from equation (2.20) and taking the norm gives

$$\|x_{i+1} - \bar{x}\| = \|g(x_i) - g(\bar{x})\| \leq L\|x_i - \bar{x}\| \leq L^{i+1}\|x_0 - \bar{x}\|. \tag{2.23}$$

The convergence follows from $0 \leq L < 1$.

This is a sufficient, but not necessary condition - fixed-point iteration can also converge in cases where the Banach fixed-point theorem does not hold.

Fixed-point iteration is easy to implement, but converges slowly - the speed of convergence is linear, with the factor of convergence being the smallest possible Lipschitz-constant of g.

2.5.2 Linear Convergence

A sequence x_k is defined to **converge linearly** to $\bar{x}$ if there exists a number $\mu \in (0,1)$ such that

$$\lim_{i\to\infty} \frac{|x_{i+1} - \bar{x}|}{|x_i - \bar{x}|} = \mu. \tag{2.24}$$

One can see directly that this holds for any Lipschitz-constant of g, thus giving linear convergence for fixed-point iteration.

2.5.3 Application to the cG(1)cG(1)-method

Eriksson et al. [12, page 1169] propose a splitting scheme for fixed-point iteration for cG(1)dG(0), which is here adapted to cG(1)cG(1).

In order to apply fixed-point iteration to the equation (2.18), one can split the equation by letting v vary and setting $q = 0$, which gives the *discrete momentum equation*

$$\begin{aligned}\left(\frac{U^n - U^{n-1}}{k_n}, v\right) + (\hat{U}^n \cdot \nabla\hat{U}^n + \nabla P^n, v + \delta_1(\hat{U}^n \cdot \nabla v)) + \nu(\nabla\hat{U}^n, \nabla v)\\ + \delta_2(\nabla \cdot \hat{U}^n, \nabla \cdot v) = (f^n, v + \delta_1(\hat{U}^n \cdot \nabla v)) \quad \forall v \in V_h^0.\end{aligned} \tag{2.25}$$

Similarly, letting q vary and setting $v = 0$, one obtains the *discrete pressure equation*

$$\delta_1(\nabla P^n, \nabla q) = -\delta_1(\hat{U}^n \cdot \nabla\hat{U}^n, \nabla q) - (\nabla \cdot \hat{U}^n, q) + \delta_1(f^n, \nabla q) \quad \forall q \in W_h, \tag{2.26}$$

where $\hat{U}^n = \frac{1}{2}(U^n + U^{n-1})$ as above.

In what follows, equations (2.25) and (2.26) are simplified without loss of generality by removing the external force-terms involving f, since the numerical examples tested in this thesis do not contain external forces ($f = 0$). The general arguments made are still valid, and since the external force-terms depend only linearly on $\hat{U}^n$, the division of terms for the fixed-point could be followed through on it in the same manner. Removing the f-terms yields

$$\begin{aligned}\left(\frac{U^n - U^{n-1}}{k_n}, v\right) + (\hat{U}^n \cdot \nabla\hat{U}^n + \nabla P^n, v + \delta_1(\hat{U}^n \cdot \nabla v)) + \nu(\nabla\hat{U}^n, \nabla v)\\ +\delta_2(\nabla \cdot \hat{U}^n, \nabla \cdot v) = 0 \quad \forall v \in V_h^0,\end{aligned} \tag{2.27}$$

and

$$\delta_1(\nabla P^n, \nabla q) = -\delta_1(\hat{U}^n \cdot \nabla\hat{U}^n, \nabla q) - (\nabla \cdot \hat{U}^n, q) \quad \forall q \in W_h. \tag{2.28}$$

When considering that $\hat{U}^n = \frac{1}{2}(U^n + U^{n-1})$, the term $(\hat{U}^n \cdot \nabla\hat{U}^n + \nabla P^n, v + \delta_1(\hat{U}^n \cdot \nabla v))$ in equation (2.27) is depending on U^n in a complex manner.

This complexity shows when considering to iterate over the equation by replacing all $\hat{U}^n$ with the k-th iterate $\hat{U}_k^n$, where $\hat{U}_k^n = \frac{1}{2}(U_k^n + U^{n-1})$. However, this can be simplified by replacing $\hat{U}_k^n \cdot \nabla\hat{U}_k^n$ with $\hat{U}_{k-1}^n \cdot \nabla\hat{U}_k^n$ and similarly $\hat{U}_k^n \cdot \nabla v$ with $\hat{U}_{k-1}^n \cdot \nabla v$, where $\hat{U}_k^n = \frac{1}{2}(U_k^n + U^{n-1})$ is the k-th iteration update.

This way, parts of the system are a little behind in the fixed-point-iteration, but the non-linear relationship is broken up.

Together with using the previous pressure iterate P_{k-1}^n in the momentum equation, this yields:

$$\begin{aligned}\left(\frac{U_k^n - U^{n-1}}{k_n}, v\right) + (\hat{U}_{k-1}^n \cdot \nabla\hat{U}_k^n + \nabla P_{k-1}^n, v + \delta_1(\hat{U}_{k-1}^n \cdot \nabla v)) + \nu(\nabla\hat{U}_k^n, \nabla v)\\ +\delta_2(\nabla \cdot \hat{U}_k^n, \nabla \cdot v) = 0 \quad \forall v \in V_h^0.\end{aligned} \tag{2.29}$$

In the pressure equation, one can use the newer velocity iterate since it has now already been computed:

$$\delta_1(\nabla P_k^n, \nabla q) = -\delta_1(\hat{U}_k^n \cdot \nabla\hat{U}_k^n, \nabla q) - (\nabla \cdot \hat{U}_k^n, q) \quad \forall q \in W_h. \tag{2.30}$$

Writing $\hat{U}_k^n$ as $\frac{1}{2}(U_k^n + U^{n-1})$ and collecting only terms involving U_k^n on the left hand side

$$\begin{aligned}&\left(\frac{U_k^n}{k_n}, v\right) + \frac{1}{2}(\hat{U}_{k-1}^n \cdot \nabla U_k^n, v + \delta_1(\hat{U}_{k-1}^n \cdot \nabla v)) + \frac{1}{2}\nu(\nabla U_k^n, \nabla v) + \frac{1}{2}\delta_2(\nabla \cdot U_k^n, \nabla \cdot v)\\ &= \left(\frac{U^{n-1}}{k_n}, v\right) - (\frac{1}{2}\hat{U}_{k-1}^n \cdot \nabla U^{n-1} + \nabla P_{k-1}^n, v + \delta_1(\hat{U}_{k-1}^n \cdot \nabla v)) - \frac{1}{2}\nu(\nabla U^{n-1}, \nabla v)\\ &\qquad -\frac{1}{2}\delta_2(\nabla \cdot U^{n-1}, \nabla \cdot v) \quad \forall v \in V_h^0.\end{aligned} \tag{2.31}$$

The pressure equation (2.30) remains the same, since there is only one term with the new iterate P_k^n which is already the only term on the left hand side.

Then, the system of equations can be solved using fixed-point iteration in the following manner:

Algorithm 2: Fixed-point iteration for cG(1)cG(1)

```
Choose initial starting guess $(U^0, P^0)$, and desired maximum increment size $\epsilon$;
for $n = 1, 2, \dots$ do
    /* Next time step.                                                   */
    Set $U_0^n = U^{n-1}$, $P_0^n = P^{n-1}$;
    for $k = 1, 2, \dots$ do
        /* Iteration steps within a time step.                           */
        Solve equation (2.31) for $U_k^n$;
        Solve equation (2.30) for $P_k^n$, treating $U_k^n$ as given;
        if $\|U_k^n - U_{k-1}^n\| + \|P_k^n - P_{k-1}^n\| < \epsilon$ then
            Exit For;
        end
    end
end
```

There are different ways to approach this iteration scheme.

One can devise other criteria for leaving the iteration than the sum of the norm of velocity and pressure - for the experiments run in this thesis, a fixed number of iterations was done.

Also, other splitting-methods with iteration for the incompressible Navier-Stokes equations exist, such as Chorin's method (see [20, pages 318f.]).

2.6 Newton's Method

2.6.1 General Introduction to Newton's Method

Another technique for solving non-linear equations is Newton's method (also known as the *Newton-Raphson method*). It is slightly more complex than fixed-point iteration, but has usually faster convergence.

The presentation here follows [20, chapter 9.2].

Let again

$$g\colon X \to X, \quad y = g(x) \tag{2.32}$$

be a non-linear function of the single variable $x \in X$.

Newton's method can then be used to find roots of g, i.e. solve the equation

$$g(x) = 0. \tag{2.33}$$

Let again $\bar{x}$ be the solution of the equation.

Newton's method can then be derived by representing $\bar{x}$ as the sum of a known initial guess x_0 and a correction term δx:

$$\bar{x} = x_0 + \delta x. \tag{2.34}$$

Assuming that δx is small (if the guess x_0 is close to $\bar{x}$), a Taylor expansion of $g(x)$ around $\bar{x}$ can be done:

$$g(\bar{x}) = g(x_0 + \delta x) = g(x_0) + g'(x_0)\delta x + \mathcal{O}(\delta x^2). \tag{2.35}$$

Since $g(\bar{x}) = 0$, and neglecting $\mathcal{O}(\delta x^2)$-terms,

$$0 \approx g(x_0) + g'(x_0)\delta x. \tag{2.36}$$

Rearranging gives

$$\delta x \approx -g(x_0)/g'(x_0). \tag{2.37}$$

Adding this δx to x_0 gives $\bar{x}$ due to equation (2.34).

By iterating this approach, algorithm 3 is derived.

Algorithm 3: Newton's method for a scalar non-linear function

```
Choose initial starting guess $x_0$, and desired maximum increment size $\epsilon$;
for $k = 0, 1, 2, \dots$ do
    Compute $\delta x_k = -g(x_k)/g'(x_k)$;
    Derive new guess $x_{k+1} = x_k + \delta x_k$;
    if $|\delta x_k| < \epsilon$ then
        Stop.
    end
end
```

For vector-valued functions, Newton's method can be derived quite similarly. The main difference is that instead of having to divide by the scalar derivate $g'(x_k)$, one has to multiply by the inverse of the Jacobian matrix (the matrix of all first-order partial derivatives of g at x_k) J^{-1}, or equivalently - and preferably! solve the linear system of equations $J_k \delta x_k = -g(x_k)$.

Newton's method is not guaranteed to converge - it needs a non-zero derivative at the iterates, and a smooth surface. For difficult cases, methods such as *line search* have been developed. See [22] for details.

Apart from its relative ease of implementation, Newton's method is popular for its convergence properties.

2.6.2 Quadratic Convergence

Similar to the linear convergence found for fixed-point iteration in subsection 2.5.2, a sequence x_k is defined to **converge quadratically** to $\bar{x}$ if there exists a number $\mu > 0$ such that

$$\lim_{i \to \infty} \frac{|x_{i+1} - \bar{x}|}{|x_i - \bar{x}|^2} = \mu. \tag{2.38}$$

Note the exponent of 2 in the denominator. A further difference to the definition of *linear convergence* is that the factor μ is allowed to be larger than 1.

Under certain given circumstances such as sufficient smoothness of the function around the guessed point, Newton's method will converge quadratically. For a proof, see [22, Theorem 3.5].

Otherwise, it will only converge linearly, or, in adverse cases as mentioned above, not at all.

2.6.3 Newton's Method in the Finite Element Method

Similarly to the derivation of Newton's method for a non-linear scalar equation in subsection2.6.1, it can be derived for a non-linear partial differential equation.

Let, similar to equations (2.2a) and (2.2b), a partial differential equation in its strong form be defined by

$$NL(u) = f, \quad \text{in} \quad \Omega \tag{2.39a}$$

$$u = 0, \quad \text{on} \quad \delta\Omega \tag{2.39b}$$

with $NL()$ a non-linear differential operator, and f a given function.

Then, u can be written as $u = u_0 + \delta u$, with u_0 being an approximation of u, and δu a small correction.

Inserting this into equation (2.39a) gives

$$NL(u_0 + \delta u) = f, \quad \text{in} \quad \Omega. \tag{2.40}$$

Doing a Taylor expansion of $NL(u_0 + \delta u)$ around u_0 gives

$$NL(u_0) + NL'(u_0)\delta u + \mathcal{O}(\delta u^2) = f, \quad \text{in} \quad \Omega. \tag{2.41}$$

Neglecting all terms that are quadratic in δu gives

$$NL(u_0) + NL'(u_0)\delta u = f, \quad \text{in} \quad \Omega, \tag{2.42}$$

which is a linear equation in δu.

From this and equation (2.39b), a weak form and a finite element method can be defined similarly to the steps in section 2.2.

Once the derived equations have been solved for δu, the iteration can continue with the new guess $u_1 = u_0 + \delta u$.

For a more thorough example, see [20, chapter 9.3].

Chapter 3

Method

Based on the theoretical background presented above, the following derivations and implementations have been made:

1. Newton's method has been applied to a modified cG(1)cG(1)-method.
2. The resulting equations have been implemented in a C++ framework.
3. Numerical experiments have been run on both this implementation, and an existing one using fixed-point iteration, and compared to each other.

3.1 Newton's Method for cG(1)cG(1)

In what follows, equation (2.18) for the modified cG(1)cG(1)-method is simplified by removing the external force-term involving f, with the same argumentation as earlier in subsection 2.5.3 when removing the force-terms in the fixed-point iteration scheme which lead from equations (2.25) and (2.26) to (2.27) and (2.28).

This yields

$$\begin{aligned}\left(\frac{U^n - U^{n-1}}{k_n}, v\right) + (\hat{U}^n \cdot \nabla \hat{U}^n + \nabla P^n, v + \delta_1(\hat{U}^n \cdot \nabla v + \nabla q))\\ +(\nabla \cdot \hat{U}^n, q) + \nu(\nabla \hat{U}^n, \nabla v) + \delta_2(\nabla \cdot \hat{U}^n, \nabla \cdot v) = 0 \quad \forall (v,q) \in V_h^0 \times W_h.\end{aligned} \tag{3.1}$$

Now, let the set $\{U^n, P^n\}$ be the true solution of the equation at time step n.

Since the equation depends in a non-linear manner on U^n, Newton's method will be derived by representing U^n as the sum of the guess U_0^n and a correction term. In subsection 2.6.1, this correction term for the variable x was called δx. Since the letter δ already occurs in the current equations, the correction term will instead be called ϵ, so that

$$U^n = U_0^n + \epsilon. \tag{3.2}$$

One can then write $\hat{U}^n$ as $\hat{U}^n = \frac{1}{2}(U^n + U^{n-1}) = \frac{1}{2}(U_0^n + \epsilon + U^{n-1}) = \hat{U}_0^n + \frac{1}{2}\epsilon$, where

$$\hat{U}_0^n = \frac{1}{2}(U_0^n + U^{n-1}). \tag{3.3}$$

This will then be used in an iteration scheme, where for the k-th iteration,

$$U_k^n = U_{k-1}^n + \epsilon_k, \tag{3.4}$$

and

$$\hat{U}^n_{k-1} = \frac{1}{2}(U^n_{k-1} + U^{n-1}). \tag{3.5}$$

Also, let P^n_k be the k-th pressure-iterate.

Inserting this and equations (3.4) and (3.5) into equation (3.1) gives

$$\begin{aligned}\left(\frac{U^n_{k-1} + \epsilon_k - U^{n-1}}{k_n}, v\right) + ((\hat{U}^n_{k-1} + \frac{1}{2}\epsilon_k) \cdot \nabla(\hat{U}^n_{k-1} + \frac{1}{2}\epsilon_k) + \nabla P^n_k, v + \delta_1(\hat{U}^n_{k-1} \cdot \nabla v + \nabla q)) \\ + (\nabla \cdot (\hat{U}^n_{k-1} + \frac{1}{2}\epsilon_k), q) + \nu(\nabla(\hat{U}^n_{k-1} + \frac{1}{2}\epsilon_k), \nabla v) + \delta_2(\nabla \cdot (\hat{U}^n_{k-1} + \frac{1}{2}\epsilon_k), \nabla \cdot v) \\ = 0 \quad \forall (v, q) \in V^0_h \times W_h,\end{aligned} \tag{3.6}$$

where the ϵ_k-term was dropped in the term $\delta_1(\hat{U}^n_{k-1} \cdot \nabla v + \nabla q)$ since this term would involve more complexity and is likely small due to the product with the small stabilization factor δ_1.

Similarly, when bringing all terms except for the ones involving ϵ_k or p^n onto the left hand side, products of the type $\epsilon_k \cdot \nabla \epsilon_k$ are dropped since they are likely small:

$$\begin{aligned}\left(\frac{\epsilon_k}{k_n}, v\right) + (\frac{1}{2}\hat{U}^n_{k-1} \cdot \nabla \epsilon_k + \frac{1}{2}\epsilon_k \cdot \nabla \hat{U}^n_{k-1} + \nabla P^n_k, v + \delta_1(\hat{U}^n_{k-1} \cdot \nabla v + \nabla q)) \\ + \frac{1}{2}(\nabla \cdot \epsilon_k, q) + \frac{1}{2}\nu(\nabla \epsilon_k, \nabla v) + \frac{1}{2}\delta_2(\nabla \cdot \epsilon_k, \nabla \cdot v) \\ = \left(\frac{U^{n-1} - U^n_{k-1}}{k_n}, v\right) - (\hat{U}^n_{k-1} \cdot \nabla \hat{U}^n_{k-1}, v + \delta_1(\hat{U}^n_{k-1} \cdot \nabla v + \nabla q)) \\ -(\nabla \cdot \hat{U}^n_{k-1}, q) - \nu(\nabla \hat{U}^n_{k-1}, \nabla v) - \delta_2(\nabla \cdot \hat{U}^n_{k-1}, \nabla \cdot v) \quad \forall (v, q) \in V^0_h \times W_h.\end{aligned} \tag{3.7}$$

This leads to the following algorithm for the cG(1)cG(1) with Newton's method:

Algorithm 4: Newton's Method for cG(1)cG(1)

```
Choose initial starting guess $(U^0, P^0)$, and maximum desired increment size $\gamma$;
for $n = 1, 2, \dots$ do
    /* Next time step.                                              */
    Set $U^n_0 = U^{n-1}$, $P^n_0 = P^{n-1}$;
    for $k = 1, 2, \dots$ do
        /* Iteration steps within a time step.                      */
        Solve equation (3.7) for the variables $\{\epsilon_k, P^n_k\}$;
        Set $U^n_k = U^n_{k-1} + \epsilon_k$;
        if $\|U^n_k - U^n_{k-1}\| + \|P^n_k - P^n_{k-1}\| < \gamma$ then
            Exit For;
        end
    end
end
```

Similar to the fixed-point method, one can devise other criteria for leaving the iteration than the sum of the norm of velocity and pressure - for the experiments run in this thesis, a fixed number of four iterations was done. The number was chosen by balancing run-time against accuracy.

Note that the difference between the method introduced in subsection 2.5.3 and in this current section is two-fold: One difference is the use of Newton's method instead of fixed-point iteration, the other difference is the solve of the complete system at each iteration instead of a splitting scheme.

3.2 Implementation

3.2.1 Code and Numerical Libraries

An existing C++ code for running the fixed-point method as derived in subsection 2.5.3 has been extended for running Newton's method.

The code was set up in a similar fashion to the related project **CMLFET**[17], which was developed in the **Computational mathematics**[32]-research group at Umeå University.

The following tools and libraries where used:

- **PETSc**[2] for parallel data structures and algorithms for scientific computations involving partial differential equations,
- **ParMETIS**[19] for parallel graph partitioning and fill reduction,
- and **tetgen**[30] for generating the 3D mesh.

As a linear solver, PETSc's GMRES-solver was chosen, with a Jacobi-preconditioner.

Most of this implementation was already in place. This thesis added mainly the use of Newton's method, and analysis of the results.

3.2.2 Hardware

All development and computations where done on a Intel(R) Core(TM) i7-4700HQ machine with 8 cores with 2.40GHz each (four cores with hyper-threading), with 16313964 kB RAM and running Ubuntu 15.10, using mpiexec with four cores. Only four cores where used since extra cores due to hyper-threading might not perform well on numerical tasks.

It had also been considered to run the simulation on the Medusa-cluster of the **Computational mathematics**[32]-research group at Umeå University in order to get higher speedup. However, since Johansson[17, page 14] found out when running **CMLFET** on this setup that using more than one node did not yield higher execution due to latency of the ethernet, this idea was not followed up.

3.2.3 Analysis of Results

The results of the simulation were exported as .vtu-files. This file type is the **Visualization Toolkit (VTK)**'s[15] xml-based file format for unstructured meshes[16]. The result files were analyzed with the help of **ParaView**[14], a software for scientific visualization which is built on top of VTK.

Chapter 4

Results and Discussions

Here, several results of the implementation will be presented and discussed.

4.1 Background - Lid-Driven Cavity

The Lid-Driven Cavity was chosen in order to test the implementation, as it is a common benchmark problem in fluid dynamics. It consists of a cubic cavity, whose lid (the top layer) has a uniform velocity, and whose interior is filled with a fluid undergoing incompressible flow. Here, the cavity of size $[0m, 1m]^3$ is considered.

The problem is easy to understand and set up due to its simple geometry and boundary conditions. See [31] for more background.

While the 2D-version of the problem is relatively simple, the 3D-problem which will be treated here leads to interesting phenomena. See figure 4.1 for an example. Given a specific geometry, the parameters to vary between different experiments are the lid's velocity at the top, as well as the fluid's viscosity.

4.1.1 Boundary Conditions

The lid-driven cavity has no-slip boundary conditions on all six walls. On the bottom wall and the side walls, $u = (0, 0, 0)^T$, while on the top wall (the lid), $u = (u_D, 0, 0)^T$, where the driving velocity u_D in x-direction is one of the input parameters of the simulation.

4.2 Implementation Details

4.2.1 Parameters

Starting from 0 at $t = 0$, the driving velocity u_D is ramped up to its final input value according to the curve in figure 4.2. The final input value is $1m/s$ in all experiments.

Apart from the driving velocity, other parameters include the kinematic viscosity of the fluid ν, as well as its density ρ, which is set to $1000kg/m^3$ (the density of water) in all experiments.

Parameters which do not change the physics-related setup of the experiment, but have influence on the numerical behavior, are the time step Δt and the spatial resolution h. Since the latter will in general not be uniform, several measures could be considered

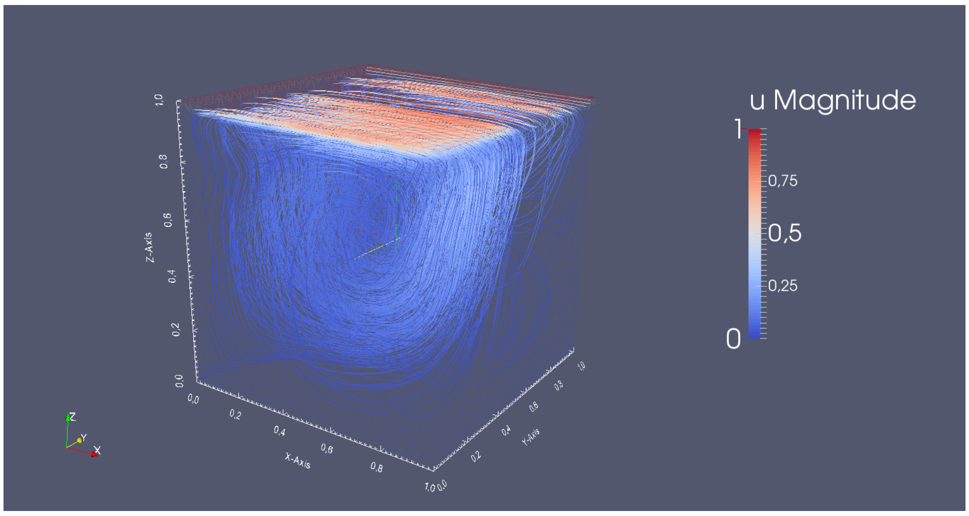


Figure 4.1: Streamline-visualization using ParaView of a simulation run of the lid-driven cavity with lid-velocity $u_D = 1m/s$ in x-direction, and $\nu = 0.001 Pa \cdot s$, at $t = 44.19s$, using fixed-point iteration. Magnitude of velocity u is shown both alongside the streamlines, and at the boundary using a wire-frame visualization.

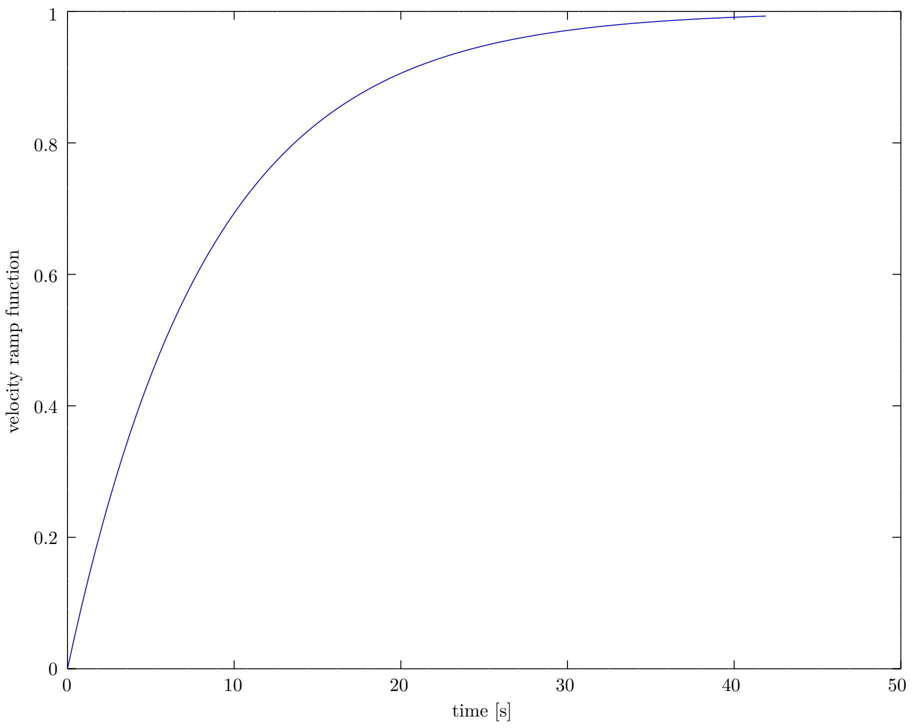


Figure 4.2: Ramping up of velocity from $t = 0$ to simulation end time.

- here, h will be defined as the longest edge length of the smallest tetrahedron (where the size of a tetrahedron shall be defined by its longest edge).

The stabilization factors introduced in equation (2.18) where set for each tetrahedron to $\delta_1 = 1.0 * h_i / u_D$ and $\delta_2 = 2.5 * h_i$ respectively, where h_i is the respective tetrahedron's largest edge.

4.2.2 Meshing

In all the numerical experiments run here, the same meshing was chosen using the library *tetgen*, with the following properties:

- Mesh points: 8328
- Mesh tetrahedra: 41258
- Mesh faces: 86040
- Mesh faces on facets: 7048
- Mesh edges on segments: 384
- Mesh minimum chunkiness parameter (defined as $c_K = \frac{s_K}{h_K}$, as in 2.2.2): 0.33288

The spatial resolution h is 0.0441942m, and the time step Δt was chosen as h/u_D, so for the case of $u_D = 1m/s$, $\Delta t = 0.0441942s$.

The finite element type used was order 1 Lagrange elements, which consist of simple hat functions as shown for the 2D-case in section 2.2. See [20, Chapter 8.12] for a definition and details.

4.2.3 Reynolds Number

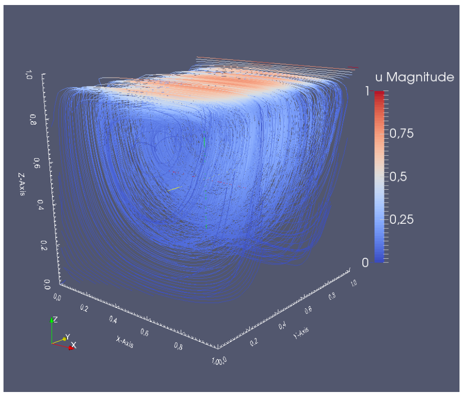
As noted earlier, the Reynolds number Re is a common heuristic for classifying fluid flow from *laminar* (at low Reynolds numbers) to *turbulent* (at high Reynolds numbers). It is defined as $Re = \frac{u \cdot L}{\nu}$, with u being a typical velocity for the scenario, and L a typical length scale. In this scenario, $u = u_D$, the driving velocity, and $L = 1m$, the edge length of the cubic cavity. Thus, $Re = \frac{1}{\nu}$.

Simulations were run for $\nu = 0.01$, $\nu = 0.001$ and $\nu = 0.0001$, leading to $Re = 100$, $Re = 1000$ and $Re = 10000$, respectively. This gave different results, which will be analyzed in the following sections.

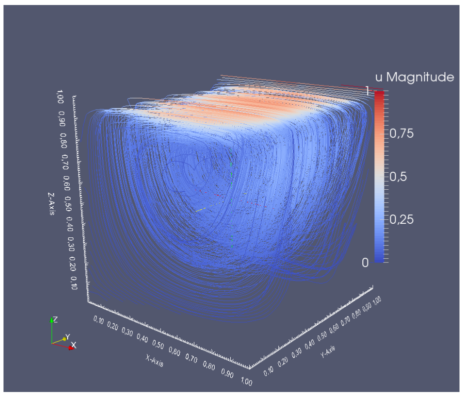
4.3 Velocity

The velocity results of the approach using the fixed-point method and Newton's method appear qualitatively similar, when considering figure 4.3 for streamline visualization plots of the velocity for simulations run with $\nu = 0.01$, $\nu = 0.001$ and $\nu = 0.0001$, and figure 4.4 for a slice of the xz-plane through point (0.5, 0.5, 0.5) of the same simulations.

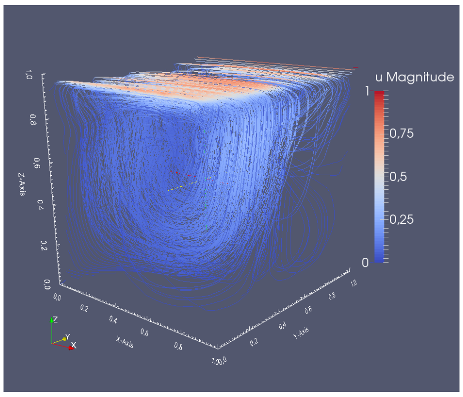
It can be seen that at $\nu = 0.01$, the laminar flow dominates, whereas at lower ν (and thus higher Reynolds number), the flow becomes more turbulent, as is to be expected.



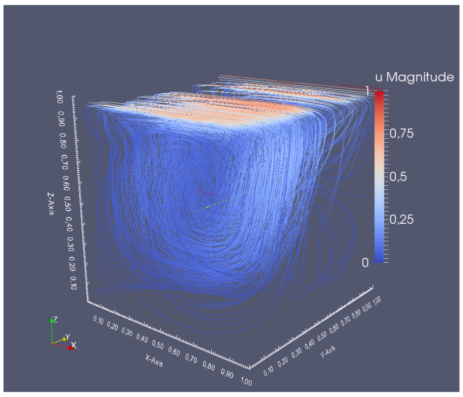
(a) Velocity from fixed-point method for $\nu = 0.01$.



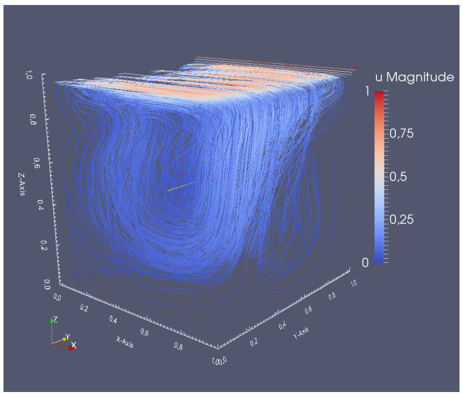
(b) Velocity from Newton's method for $\nu = 0.01$.



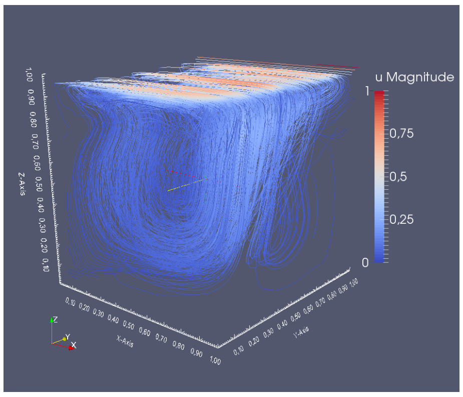
(c) Velocity from fixed-point method for $\nu = 0.001$.



(d) Velocity from Newton's method for $\nu = 0.001$.

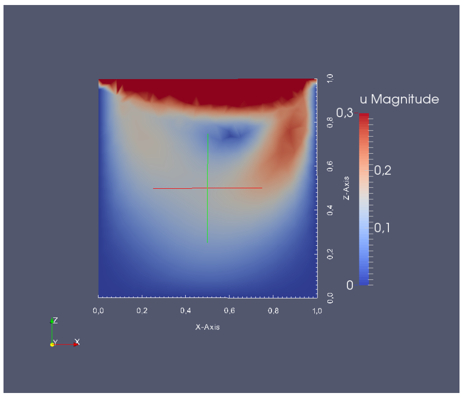


(e) Velocity from fixed-point method for $\nu = 0.0001$.

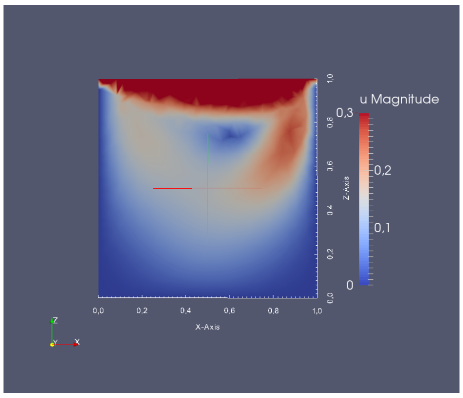


(f) Velocity from Newton's method for $\nu = 0.0001$.

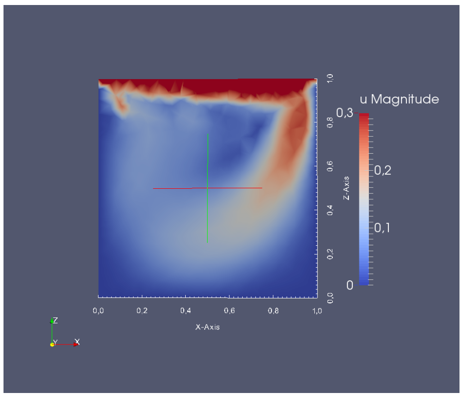
Figure 4.3: Streamline-visualization of velocity using ParaView. Simulations run of the lid-driven cavity with lid-velocity $u_D = 1m/s$ in x-direction, and several values of ν, at $t = 44.19s$.



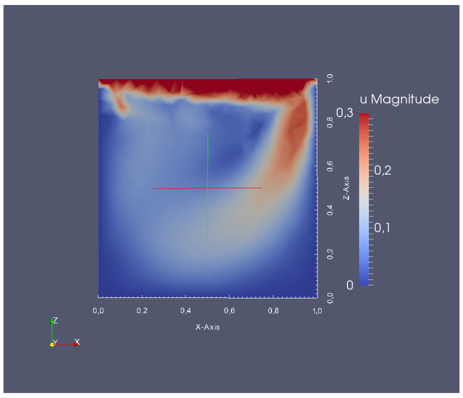
(a) Velocity from fixed-point method for $\nu = 0.01$.



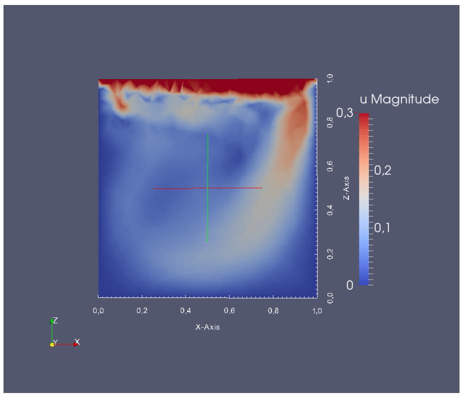
(b) Velocity from Newton's method for $\nu = 0.01$.



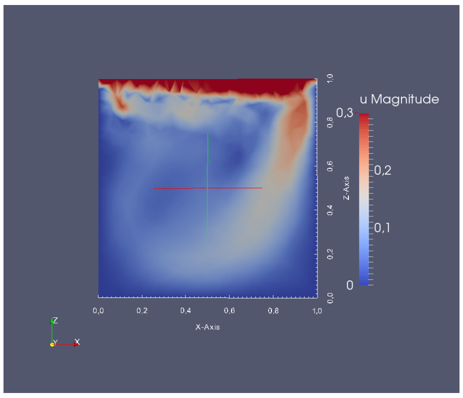
(c) Velocity from fixed-point method for $\nu = 0.001$.



(d) Velocity from Newton's method for $\nu = 0.001$.



(e) Velocity from fixed-point method for $\nu = 0.0001$.



(f) Velocity from Newton's method for $\nu = 0.0001$.

Figure 4.4: Velocity profile at the xz-plane through point (0.5, 0.5, 0.5) of velocity using ParaView. Simulations run of the lid-driven cavity with lid-velocity $u_D = 1m/s$ in x-direction, and several values of ν, at $t = 44.19s$. Note that the scale ends at 0.3m/s instead of 1.0m/s in order to highlight the flow inside the cavity.

4.4 Pressure

Like the velocity results mentioned above, the pressure results from Newton's method and the fixed-point method are qualitatively similar. See figure 4.5 for pressure plots for $\nu = 0.01$, $\nu = 0.001$ and $\nu = 0.0001$.

As a side note: Since the lid-driven cavity test case used here does not introduce any boundary conditions for the pressure, the pressure can only be determined up to an additive constant. When presenting the results in figure 4.5, the pressure results from the simulation had been averaged to zero first. This has been done by integrating over the pressure resulting directly from the simulation, and reducing the resulting additive constant from the pressure. The additive constants can be seen in table 4.1. The additive constants for Newton's Method are constant with regard to ν and quite close to zero, whereas the ones for the fixed-point method are farter from zero and differ slighly with ν.

	Fixed-Point Method	**Newton's Method**
$\nu = 0.01$	-0.242666	-0.00975585
$\nu = 0.001$	-0.243358	-0.00975585
$\nu = 0.0001$	-0.242837	-0.00975585

Table 4.1: Table additive constants in pressure results for fixed-point method and Newton's method, for several values of ν.

Given that both the velocity and the pressure results from fixed-point iteration and Newton's Method agree well, it is interesting to compare them with regards to their respective convergence properties and computation time.

4.5 Convergence

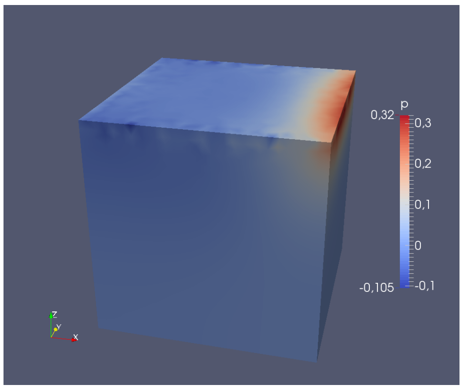
Both fixed-point iteration and Newton's method are iterative methods.

For comparing the convergence of the two approaches, their respective convergence rates can be considered. See subsection 2.5.2 for a definition of linear convergence, and subsection 2.6.2 for a definition of quadratic convergence.

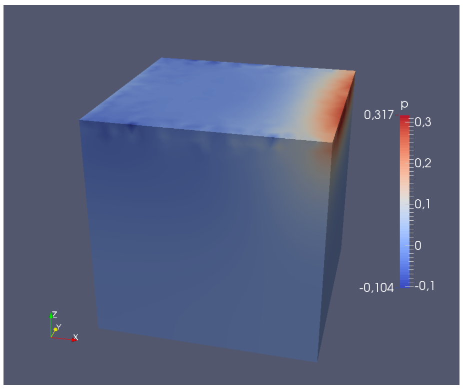
For simplicity, the i-th iteration update $|\Delta u_i|$ for the velocity will be used as approximation instead of $|x_i - \bar{x}|$ from equation (2.24) (and similarly, the pressure i-th iteration update $|\Delta p_i|$ will be used for measuring the convergence of the pressure).

The update errors for velocity and pressure for both methods are presented in the tables 4.2, 4.3, 4.4 and 4.5 for two points in time, together with the quotient between each iteration update. Note that for the both methods, four iterations were done per time step.

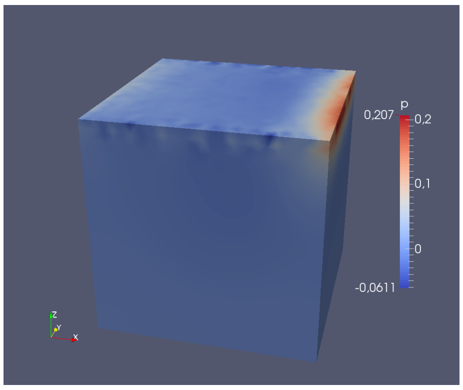
In table 4.6, a simulation was run with a time step of $\Delta t = 0.004419s$, 1/10th of the one in the other runs, and with 10 Newton iterations, in order to have a closer look on the convergence process. For practical purposes, this high resolution would be impractical, since simulations would take several days to complete. By considering the steady decrease in the iteration update size for both u and p, it can be seen that the Newton process converges. When reaching iteration update sizes of about 1E-11, the convergence rate decreases drastically - this is most likely due to round-off issues so close to machine precision. The following things can be noted:



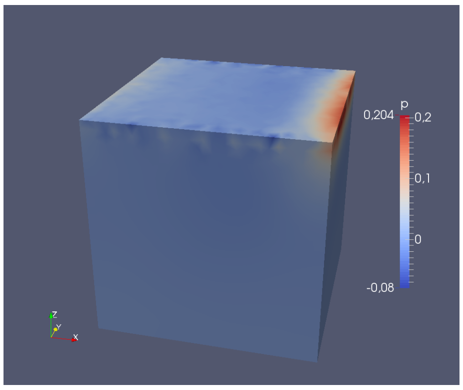
(a) Pressure from fixed-point method for $\nu = 0.01$.



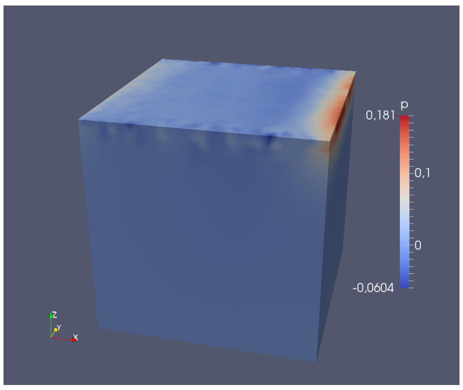
(b) Pressure from Newton's method for $\nu = 0.01$.



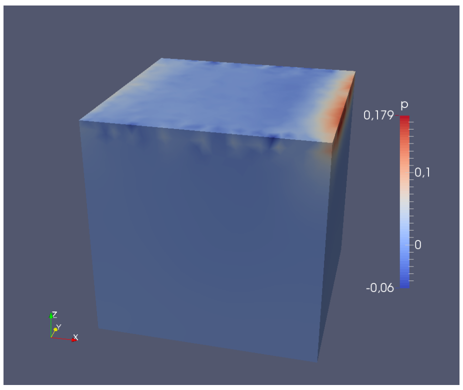
(c) Pressure from fixed-point method for $\nu = 0.001$.



(d) Pressure from Newton's method for $\nu = 0.001$.



(e) Pressure from fixed-point method for $\nu = 0.0001$.



(f) Pressure from Newton's method for $\nu = 0.0001$.

Figure 4.5: Pressure visualization using ParaView of simulations run of the lid-driven cavity with lid-velocity $u_D = 1m/s$ in x-direction, and several values of ν, at $t = 44.19s$. Note the different scales.

1. The fixed-point method converges after three steps for the velocity, and one step for the pressure, so that a convergence rate as above cannot be defined for the latter - however, the convergence is faster than can be hoped for when using the fixed-point method, which generally converges linearly (see section 2.5).
2. Newton's method converges here linearly or slightly less than that (similar to the fixed-point method with regards to velocity), since a μ as in equation (2.24) cannot be defined. This is not what is expected from section 2.6.
3. Since Newton's method converges worse than expected here, more iterations would be needed in some of the time steps in order to get a satisfactory result. Instead of a fixed number of iterations, a tolerance-based termination criterion would be preferable, but has not been tested due to lack of time - simulations of this size with a high number of iterations would take several days to complete.

However, it should be noted that Newton's method here considers the complete system, whereas the fixed-point method treats a split system, and its convergence numbers which we get here are only for the sub-parts of the system. In order to compare the two approaches, a different measure should be used, such as e.g. the residual when inserting the found solution into the complete equation.

	Fixed-Point Method		**Newton's Method**	
iteration i	$\lvert\Delta u_i\rvert$	$\frac{\lvert\Delta u_i\rvert}{\lvert\Delta u_{i-1}\rvert}$	$\lvert\Delta u_i\rvert$	$\frac{\lvert\Delta u_i\rvert}{\lvert\Delta u_{i-1}\rvert}$
1	8.367640E-02	-	1.048800E+01	-
2	8.232040E-04	9.837947E-03	2.906260E-02	2.771034E-03
3	4.700480E-05	5.709982E-02	8.427050E-05	2.899620E-03
4	0.000000E+00	0.000000E+00	4.478770E-07	5.314754E-03

Table 4.2: Table over convergence of u using fixed-point method and Newton's method, for $\nu = 0.001$ and $t = 4.419$ (after 100 time steps).

	Fixed-Point Method		**Newton's Method**	
iteration i	$\lvert\Delta u_i\rvert$	$\frac{\lvert\Delta u_i\rvert}{\lvert\Delta u_{i-1}\rvert}$	$\lvert\Delta u_i\rvert$	$\frac{\lvert\Delta u_i\rvert}{\lvert\Delta u_{i-1}\rvert}$
1	1.081610E-03	-	2.684680E+01	-
2	0.000000E+00	0.000000E+00	2.611480E-01	9.727342E-03
3	0.000000E+00	-	4.644280E-03	1.778409E-02
4	0.000000E+00	-	1.142960E-04	2.461006E-02

Table 4.3: Table over convergence of u using fixed-point method and Newton's method, for $\nu = 0.001$ and $t = 44.19$ (after 1000 time steps).

4.6 Computation Times

The computation times between the approach using Newton's method were much higher than when using fixed-point, as can be seen in table 4.7. They are approximately 16 times as high, independent on the value of ν.

This can be explained by the following considerations:

	Fixed-Point Method		**Newton's Method**	
iteration i	$\lvert\Delta p_i\rvert$	$\frac{\lvert\Delta p_i\rvert}{\lvert\Delta p_{i-1}\rvert}$	$\lvert\Delta p_i\rvert$	$\frac{\lvert\Delta p_i\rvert}{\lvert\Delta p_{i-1}\rvert}$
1	1.364060E-01	-	5.846910E-02	-
2	2.175190E-04	1.594644E-03	6.239610E-02	1.067164E+00
3	1.298340E-05	5.968858E-02	7.802060E-06	1.250408E-04
4	0.000000E+00	0.000000E+00	2.457430E-08	3.149719E-03

Table 4.4: Table over convergence of p using fixed-point method and Newton's method, for $\nu = 0.001$ and $t = 4.419$ (after 100 time steps).

	Fixed-Point Method		**Newton's Method**	
iteration i	$\lvert\Delta p_i\rvert$	$\frac{\lvert\Delta p_i\rvert}{\lvert\Delta p_{i-1}\rvert}$	$\lvert\Delta p_i\rvert$	$\frac{\lvert\Delta p_i\rvert}{\lvert\Delta p_{i-1}\rvert}$
1	1.092770E-04	-	4.032620E-01	-
2	0.000000E+00	0.000000E+00	4.036940E-01	1.001071E+00
3	0.000000E+00	-	7.581700E-04	1.878081E-03
4	0.000000E+00	-	9.585140E-06	1.264247E-02

Table 4.5: Table over convergence of p using fixed-point method and Newton's method, for $\nu = 0.001$ and $t = 44.19$ (after 1000 time steps).

	Newton's Method			
iteration i	$\lvert\Delta u_i\rvert$	$\frac{\lvert\Delta u_i\rvert}{\lvert\Delta u_{i-1}\rvert}$	$\lvert\Delta p_i\rvert$	$\frac{\lvert\Delta p_i\rvert}{\lvert\Delta p_{i-1}\rvert}$
0	2.5680E-01	-	5.3917E-04	-
1	7.0315E-07	2.7381E-06	1.1657E-05	2.1620E-02
2	2.0098E-12	2.8582E-06	5.6144E-11	4.8164E-06
3	7.7833E-13	3.8727E-01	9.4996E-13	1.6920E-02
4	...	...	...	...

Table 4.6: Table over convergence of u and p using Newton's method with a time step $\Delta t = 0.004419s$ being 1/10th of the one used in the other examples, and with 10 iterations, for $\nu = 0.001$ and at $t = 0.08838$ (after 20 out of 10000 time steps). Only the first 4 iterations are shown, since values oscillate around 1.0E-12 then.

The fixed-point method uses a splitting scheme, and computes thus only two smaller matrices, whereas the approach using Newton's method assembles one large matrix. Given the meshing described above with 8382 points, the matrix sizes are like this:

1. The split-velocity-matrix has a size of 24984 x 24984 entries.

2. The split-pressure-matrix has a size of 8328 x 8328 entries.

3. The complete system matrix (used in Newton's method) has 33312 x 33312 entries.

Assuming the same sparsity pattern, and a best-case linear scaling of the iterative solver, the ratio between the number matrix entries in matrix 3, and sum of the matrix entries in matrices 1 and 2 is $\frac{1109689344}{693555840} = 1.6$. On top of this factor, however, the complete matrix seems harder to solve: GMRES needs usually 50-800 Krylov subspace iterations on it (depending on time step and Newton iteration), whereas the two solves for the smaller matrices take usually between 10 and 100 Krylov subspace iterations.

This result makes the use of this approach based on Newton's method less attractive in the general case, it might however still be useful if correct pressure computations are of interest, as shown above.

While there was hope that using Newton's Method would lead to better convergence and thus optimization of the run-time, low-level optimization of the implementation for execution time has not been a focus of the thesis, which makes it likely that there is room for improvement on that end.

	Fixed-Point Method	**Newton's Method**
$\nu = 0.01$	0:48:53h	13:24:37h
$\nu = 0.001$	0:54:24h	15:57:54h
$\nu = 0.0001$	0:50:48h	13:41:52h

Table 4.7: Table over computation times for simulations using fixed-point method and Newton's method, for several values of ν.

4.7 Kinetic Energy

A sanity check for the quality of the numerical solution with regard to the real-world system being modeled is the kinetic energy. The simulation starts with the fluid being in a rest state. Due to the constant velocity being applied at the lid, one expects a steady state or something similar to have been reached after a certain amount of time.

If all the volume of the cube of $V = 1m^3$ with density of $\rho = 1000kg/m^3$ were to obtain the velocity of $u_D = 1m/s$, this would lead to a kinetic energy of $E_k = u_D^2 \rho V = 1000J$.

As is known from the results shown earlier, only parts of the fluid obtain this velocity, thus leading to a substantially lower kinetic energy. Since the motion (when disregarding the turbulent behavior) is close to circular along the y-axis acting like a cylinder axis, only fluid moving along the outermost radius will have a maximum velocity of 1m/s. Motion closer to the center however will have linearly lower velocity. Further causes for lower kinetic energy away from the lid include internal shear forces resisting the motion, as well as turbulence.

Regarding other energy forms: Note that due to the constant density, the potential energy in the system is constant. In the real system, internal friction and shear forces would lead to kinetic energy being converted to heat energy, but the effects of temperature are not modeled by the formulation of the incompressible Navier-Stokes equations considered here.

4.7.1 Comparison of Newton's Method and Fixed-Point Method

The approaches using Newton's method and the fixed-point method behave quite similar here, as can be seen in figure 4.6. Compare figure 4.2 for the corresponding velocity ramp-up.

Note that when generating this figure, the assumption was taken that all tetrahedra in the mesh have uniform size. The meshing process used strives for equally-sized tetrahedra, but there might be some derivations, so the correct kinetic energy is expected to differ slightly.

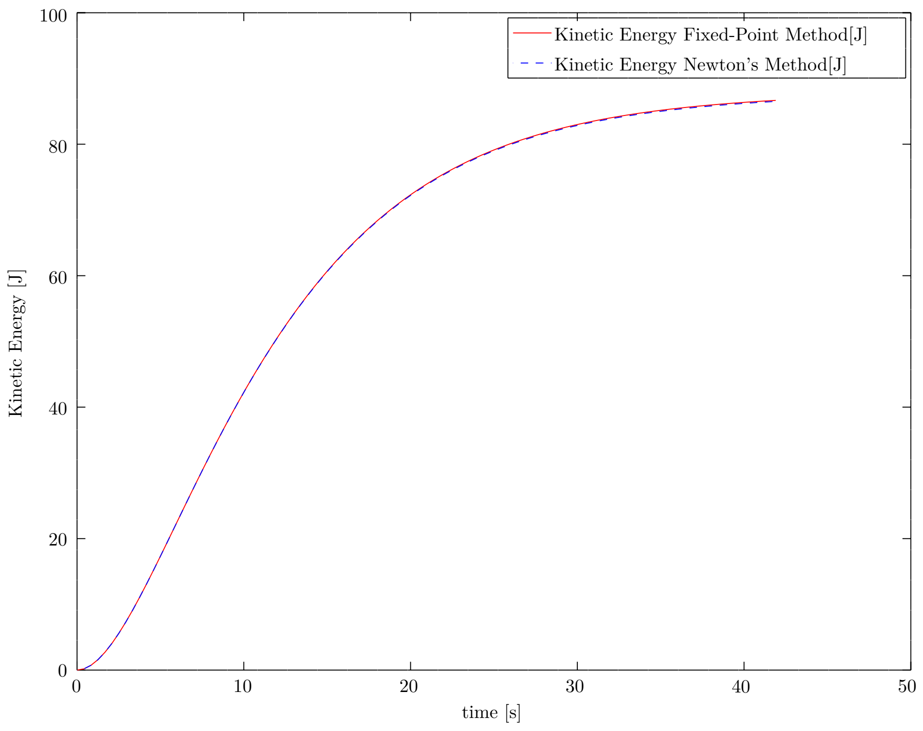


Figure 4.6: Kinetic Energy in fixed-point method vs. Newton's method, for $\nu = 0.001$ and u_D=1m/s.

Chapter 5

Conclusion

5.1 Achievements

This thesis introduced an implementation of Newton's method for the cG(1)cG(1)-method of numerically solving the incompressible Navier-Stokes equations, and compared it to an earlier version using fixed-point iteration.

The velocity and pressure results from both methods correspond closely, as investigated by qualitative analysis by considering streamlines and xz velocity profiles, as well as considering the kinetic energy as a quantitative measure for velocity. For the pressure, surface visualizations have been used for quantitative comparisons.

Newton's Method in this implementation does not converge quadratically as hoped for, and has a much higher computation time than the existing method using splitting and fixed-point iteration.

5.2 Shortcomings

The following shortcomings have been found and could be addressed in future work:

- **Tolerance-based termination criteria** - For both iterative methods, a tolerance-based termination criterion should be used, to make sure that the iteration processes converge. Note that this might increase computation times considerably.

- **Investigate Newton's method's bad convergence** - It is expected to converge quadratically when close to the solution, and linearly when further away. Here however, it converges linearly or less than that, which should be investigated. Approaches such as line search might help if this is an issue of global versus local convergence.

- **Investigate Newton's method's computation time** - In order to improve the computation time, optimizations and better use of parallelization should be investigated, e.g. by delaying communication points.

- **Low number of test cases** - More test cases should be investigated, both regarding geometry, boundary conditions, and parameters such as viscosity and external force.

- **Compare the two iterative methods by using residual** - To give a fairer quality measure of the two iterative processes than their convergence rate, the residual at each iteration step should be computed by inserting the found solution into the complete equation.

5.3 Future Work

Apart from addressing the shortcomings mentioned above, the following ideas could be followed up in future work:

- **Update to a newer version of PETSc** - The version used is quite old, and a newer version might bring improvements in performance and stability.
- **Examine Newton's method included in PETSc** - PETSc includes methods for non-linear solves, amongst others Newton's method. This would need a rewrite of parts of the framework.
- **Investigate use of GPGPU** - *Graphical Processing Units* (GPUs) and other dedicated hardware have outperformed more general processing units (CPUS) in a number of computational tasks within the last years, leading to the field of *General Purpose GPUs* (GPGPU). PETSc also supports GPU code. However, it is a large task to make efficient use of a given GPU, or even an array of different GPUs and other processors.

Chapter 6

Acknowledgments

I would like to thank my supervisor Mats G. Larson for his support and patience, Fredrik Bengzon for his support and help with implementation details, and Mats Johansson with his support with infrastructure and a work desk at Umit research lab.

I would also like to thank my wife Elena Brandl for her love and support.

References

[1] O. Alexandrov. *Finite element triangulation.* Accessed: 2016-05-23. URL: `https://commons.wikimedia.org/wiki/File:Finite_element_triangulation.svg`.

[2] S. Balay et al. *PETSc Web page.* `http://www.mcs.anl.gov/petsc`. Accessed: 2015-12-16. 2015. URL: `http://www.mcs.anl.gov/petsc`.

[3] G. K. Batchelor. *An Introduction to Fluid Dynamics.* Cambridge University Press, 2000. ISBN: 9781139175241. DOI: `10.1063/1.3060769`.

[4] F. Bengzon. "Adaptive Finite Element Methods for Multiphysics Problems". PhD thesis. 2009. ISBN: 9789172648999.

[5] M. Braack and P. B. Mucha. "Directional Do-Nothing Condition for the Navier-Stokes Equations". In: *Journal of Computational Mathematics* 32.5 (2014), pp. 507–521. ISSN: 02549409. DOI: `10.4208/jcm.1405-m4347`. URL: `http://www.global-sci.org/jcm/volumes/v32n5/pdf/325-507.pdf`.

[6] P. J. Capon. "Adaptive Stable Finite Element Methods for the Compressible Navier-Stokes Equations". PhD thesis. 1995.

[7] S. S. Clift and P. A. Forsyth. "Linear and Non-linear Iterative Methods for the Incompressible Navier-Stokes Equations". In: *...Journal for Numerical Methods in Fluids* (1994), pp. 1–40. URL: `http://onlinelibrary.wiley.com/doi/10.1002/fld.1650180302/abstract`.

[8] W. S. Edwards et al. "Krylov Methods for the Incompressible Navier-Stokes Equations". In: *Journal of Computational Physics* 110.1 (1994), pp. 82–102. ISSN: 00219991. DOI: `10.1006/jcph.1994.1007`. URL: `http://www.sciencedirect.com/science/article/B6WHY-45P0TV7-64/2/cad045b7eaecfacc4e6ac09fb79d14c8`.

[9] H. C. Elman, D. Loghin, and A. J. Wathen. "Preconditioning techniques for Newton's method for the incompressible Navier-Stokes equations". In: *BIT Numerical Mathematics* 43.October 2002 (2003), pp. 961–974. URL: `http://www.springerlink.com/index/N760573324Q00520.pdf`.

[10] H. C Elman, D. J. Silvester, and A. J. Wathen. *Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics.* 2014. ISBN: 9780199678792. DOI: `10.1093/acprof:oso/9780199678792.001.0001`.

[11] H. C. Elman et al. "A parallel block multi-level preconditioner for the 3D incompressible Navier-Stokes equations". In: *Journal of Computational Physics* 187.2 (2003), pp. 504–523. ISSN: 00219991. DOI: `10.1016/S0021-9991(03)00121-9`.

[12] K. Eriksson, D. Estep, and C. Johnson. “Incompressible Navier-Stokes: quick and easy”. In: *Applied Mathematics: Body and Soul: v. 3 Calculus in Several Dimensions*. 2003, pp. 1189–1204.

[13] C. L. Fefferman. “Existence and smoothness of the Navier-Stokes equation”. In: *The millennium prize problems* 1 (2000), pp. 1–5. URL: http://books.google.com/books?hl=en%5C&lr=%5C&id=7wJIPJ80RdUC%5C&oi=fnd%5C&pg=PA57%5C&dq=EXISTENCE+%5C&+SMOOTHNESS+OF+THE+NAVIER?STOKES+EQUATION%5C&ots=3GyvscOooI%5C&sig=35QIWVeNXQStYtPdUPLOSoTVu6I.

[14] Kitware Inc. *ParaView*. http://www.paraview.org/. Accessed: 2015-12-16.

[15] Kitware Inc. *Visualization Toolkit (VTK)*. http://www.vtk.org/. Accessed: 2015-12-16.

[16] Kitware Inc. *VTK File Formats*. http://www.vtk.org/wp-content/uploads/2015/04/file-formats.pdf. Accessed: 2015-12-16.

[17] M. Johansson. “Simulation of a multi phase flow in a rotating-lid driven cylinder”. MA thesis. UmeåUniversity, 2013.

[18] S. D. Kim, Y. H. Lee, and B. C. Shin. “Newton’s method for the Navier-Stokes equations with finite-element initial guess of stokes equations”. In: *Computers & Mathematics with Applications* 51.5 (2006-03), pp. 805–816. ISSN: 08981221. DOI: 10.1016/j.camwa.2006.03.007. URL: http://linkinghub.elsevier.com/retrieve/pii/S0898122106000174.

[19] Karypis Lab. *ParMETIS*. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview. Accessed: 2015-12-16.

[20] M. G. Larson and F. Bengzon. *The Finite Element Method: Theory, Implementation, and Applications*. 2013.

[21] NASA Langley Research Center (NASA-LaRC). *Airplane vortex*. Accessed: 2016-04-25. URL: https://en.wikipedia.org/wiki/File:Airplane_vortex_edit.jpg.

[22] J. Nocedal and S. J. Wright. *Numerical optimization*. 2006, xxii, 664 p. ISBN: 0387303030 (hd. bd.)\n9780387303031. DOI: 10.1007/978-0-387-40065-5. URL: http://www.loc.gov/catdir/enhancements/fy0818/2006923897-d.html$%5Cbackslash$nhttp://www.loc.gov/catdir/enhancements/fy0818/2006923897-t.html.

[23] A. Olofsson. “Theory and Parallel Implementation of an Adaptive Finite Element Method for the Coupled Stokes and Navier-Lamé Equations”. MA thesis. 2008.

[24] P. D. Orkwis. *Comparison of Newton’s and quasi-Newton’s method solvers for the Navier-Stokes equations*. 1993. DOI: 10.2514/3.11693.

[25] P.-O. Persson. “High-Order Navier-Stokes Simulations Using a Sparse Line-Based Discontinuous Galerkin Method”. In: *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition* (2012), pp. 1–12. DOI: 10.2514/6.2012-456. URL: http://arc.aiaa.org/doi/abs/10.2514/6.2012-456.

[26] T. H. Pulliam. “Time accuracy and the use of implicit methods”. In: 1 (1993), pp. 1–11. DOI: doi:10.2514/6.1993-3360.

[27] K. Selim, A. Logg, and M. G. Larson. “An Adaptive Finite Element Splitting Method for the Incompressible Navier–Stokes Equations”. In: *Computer Methods in Applied Mechanics and Engineering* 209-212 (2011), pp. 54–65. ISSN: 00457825. DOI: 10.1016/j.cma.2011.10.002. arXiv: 1205.3096. URL: http://linkinghub.elsevier.com/retrieve/pii/S0045782511003148.

[28] J. N. Shadid, R. S. Tuminaro, and H. F. Walker. “An inexact Newton method for fully coupled solution of the Navier-Stokes equations with heat and mass transport”. In: *Journal of Computational Physics* 137 (1997), pp. 155–185. ISSN: 00219991. DOI: 10.1006/jcph.1997.5798.

[29] T. W. H. Sheu and R. K. Lin. “Newton linearization of the incompressible Navier-Stokes equations”. In: *International Journal for Numerical Methods in Fluids* 44.3 (2004), pp. 297–312. ISSN: 02712091. DOI: 10.1002/fld.639.

[30] H. Si. “TetGen, a Quality Tetrahedral Mesh Generator”. In: *AMC Transactions on Mathematical Software* 41.2 (2015), p. 11. ISSN: 00983500. DOI: 10.1007/3-540-29090-7_9.

[31] L. Q. Tang, T. Cheng, and T. T. H. Tsang. “Transient solutions for three-dimensional lid-driven cavity flows by a least-squares finite element method”. In: *International Journal for Numerical Methods in Fluids* 21.5 (1995), pp. 413–432. ISSN: 0271-2091. DOI: 10.1002/fld.1650210505. URL: http://doi.wiley.com/10.1002/fld.1650210505.

[32] Computational Mathematics Research Group at Umit. *Computational mathematics*. Accessed: 2016-05-26. URL: http://www.org.umu.se/umit/english/about-umit/research-groups/computational-mathematics/.

[33] Z. Žunič et al. “3D Lid Driven Cavity Flow By Mixed Boundary and Finite Element Method”. In: *European Conference on Computational Fluid Dynamics* (2006), pp. 1–12.