BACHELORARBEIT

Stabilized $\mathcal{P}_1/\mathcal{P}_1$ elements for incompressible flow

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Bachelor of Science unter der Leitung von

Prof. Dr. Joachim Schöberl

E101 – Institut für Analysis und Scientific Computing

eingereicht an der Technischen Universität Wien
Fakultät für Mathematik und Geoinformation
von

Sebastian Hirnschall

Matrikelnummer: 11827170

Heinrich Pichler Gasse 4, 2700 Wr. Neustadt

December 10, 2022

Abstract

This thesis discusses both Stokes' and Navier-Stokes' equations to compute the flow of incompressible isothermal Newtonian fluids using different space discretization methods. Special attention is given to the usage of lowest order $\mathcal{P}_1/\mathcal{P}_1$ element pairings to reduce the number of degrees of freedom when working on fine meshes. The suggested elements are easy to implement and directly available in the open source FEM-library NGSolve [19]. A stabilization term is used to explicitly weaken the divergence-free condition and overcome the fact that a combination of lowest order elements for both the pressure and the velocity fe-space do not meet the LBB-condition.
To show the advantages and disadvantages of this method, we use both traditional $\mathcal{P}_2/\mathcal{P}_1$ Taylor-Hood elements, as well as $H(div)$-conforming elements, as a reference.
We combine this space discretization with a suitable, stiffly-accurate, IMEX-scheme to approximate turbulent flow.
Finally, numerical results are presented. They agree with our expectations.

Key words.

Stokes, Navier-Stokes, Finite element methods, Incompressible flow, Taylor-Hood, Stabilization, IMEX-scheme

AMS subject classifications (2020).

76M10, 35Q30, 65M60, 65N30, 65M12, 76D05, 35Q35

Zusammenfassung

Diese Arbeit beschäftigt sich mit Stokes und Navier-Stokes Gleichungen, um den Fluss von inkompressiblen Newtonschen Flüssigkeiten zu berechnen. Besonderes Augenmerk legen wir dabei auf lowest order $\mathcal{P}_1/\mathcal{P}_1$ Elemente, um die Anzahl der Unbekannten, vor allem auf sehr feinen Gittern, zu reduzieren. Die vorgestellten Elemente sind leicht zu implementieren und direkt in der open-source FEM-Bibliothek NGSolve [19] verfügbar. Ein Stabilisierungsterm wird dabei verwendet, um die Bedingung der Divergenzfreiheit explizit zu schwächen. Dies ist notwendig, da eine Kombination aus lowest-order FE-Räume für Druck und Geschwindigkeit nicht die LBB-Bedingung erfüllt.
Um Vor- und Nachteile dieser Methode aufzuzeigen, werden sowohl die klassischen $\mathcal{P}_2/\mathcal{P}_1$ Taylor-Hood-Elemente, als auch $H(div)$-konforme Elemente als Referenz verwendet.
Wir kombinieren diese Raumdiskretisierungen mit einem geeigneten, L-stabilen, IMEX-Verfahren, um auch turbulente Strömungen zu berechnen.
Schlussendlich präsentieren wir numerische Ergebnisse, die unsere Erwartungen bestätigen.

Schlagworte.

Stokes, Navier-Stokes, Finite Elemente Methode, inkompressible Strömung, Taylor-Hood, Stabilisierung, IMEX-Verfahren

Acknowledgments

The computational results presented have been achieved [in part] using the Vienna Scientific Cluster (VSC).

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

There are several stable finite element pairings for incompressible flow known. However, using a higher order space for the velocity as with Taylor-Hood elements [TH73] or additional degrees of freedom to achieve e.g. exact divergence free solutions like $H(div)$-conforming elements introduced in [Leh10] and [LS16] enlarges the size of the resulting system of equations, thus increasing the computational complexity.
We will therefore examine the possibility of using stabilized $\mathcal{P}_1/\mathcal{P}_1$ element pairings as suggested in [BF91] and [BP84]. When working with a fine mesh, as in e.g. chapter 6, the difference in computational complexity is significant.

1.2 Outline

This thesis focuses on solving Stokes' and Navier-Stokes' equations on a fine mesh using the finite element method. We therefore compare different element-pairings, namely Taylor-Hood elements, $H(div)$-conforming elements, and $\mathcal{P}_1/\mathcal{P}_1$ elements with stabilization (chapter 4). To cope with the time-derivative in Navier-Stokes, we use an L-stable IMEX time-stepping-method discussed in chapter 5.
In particular this thesis contributes the following approaches and results:

Chapter 4: We introduce and discuss different stable finite-element-pairings for Stokes, i.e. velocity- and pressure-space combinations for which the LBB-condition holds.

Chapter 5: We choose a suitable time-stepping method to approximate solutions to the time dependent Navier-Stokes' equations. In particular a stiffly

accurate IMEX scheme to ensure that the solution stays divergence free from time-step to time-step.

Chapter 6: We conclude with several numerical comparisons of the different element pairings introduced in chapter 4 to work out the benefits and drawbacks of each method. This includes not only the size of the resulting matrices (number of degrees of freedoms) but also accuracy and other limitations.

1.3 Implementations

The complete code used to obtain the numerical results presented in chapter 6 can be found in appendix A and on GitHub at

https://github.com/shirnschall/bachelorthesis.

The code uses the open-source finite element library *NGSolve* and mesh-handler *Netgen* [19; Sch97] available at

https://github.com/NGSolve.

The different element-types we want to use are directly available.

Chapter 2

Mathematical modeling

In this chapter we will derive both Stokes' and Navier-Stokes' equation as well as their weak formulations for use in the next chapters.
As we want to analyze the behavior of incompressible isothermal Newtonian fluids, we can use a model based on basic results from physics. Let u be the velocity-field, p the pressure, ρ the density and μ the viscosity of a fluid.
It is well known that

1. Mass is conserved, and that
2. Mass times acceleration is equal to force (Newton's second Law).

2.1 Navier-Stokes

Mass conservation and incompressibility of the fluid leads to the first Navier-Stokes' equation

$$\operatorname{div}(u) = 0. \tag{2.1}$$

Now, as we are considering each point individually, $ma = F$ simplifies to

$$\rho \frac{du}{dt} = F,$$

where F is the sum of forces acting on each point.
Splitting F into internal and external forces leads to the second Navier-Stokes' equation

$$\rho \frac{du}{dt} = -\nabla p + \mu \nabla^2 u + f, \tag{2.2}$$

where $-\nabla p$ is the force exerted by a difference in pressure, $\mu\nabla^2 u$ is based on viscosity and f are the remaining external forces. If the only external force is gravity, f can be rewritten as ρg. Furthermore, if the density ρ is constant eq. (2.2) can be divided by ρ on both sides to simplify it further.

If a pair $u \in C^2(\Omega, L_b(\Omega, \mathbb{R})) \cap C^0(\overline{\Omega})$ and $p \in C^1(\Omega) \cap C^0(\overline{\Omega})$ exists that satisfies eqs. (2.1) and (2.2), it is called a classical solution.

2.1.1 Weak Formulation

As we will later need the weak formulation of both equations we will construct them now.
Let $x \in C^1(\mathbb{R}_0^+, \mathbb{R}^n) : x(t) = (x^0, \ldots, x^{n-1}), t \in \mathbb{R}_0^+$. Using the chain rule the material derivative du/dt can be calculated as

$$\begin{aligned}\frac{du(x(t),t)}{dt} &= u_t + \sum_{i=0}^{n-1} u_{x^i} x_t^i \\ &= u_t + \sum_{i=0}^{n-1} u_{x^i} u^i(x,t) \\ &= u_t + u \cdot \nabla u.\end{aligned}$$

Thus, eq. (2.2) expands to

$$\rho\left(u_t + u \cdot \nabla u\right) = -\nabla p + \mu\nabla^2 u + f.$$

Multiplying both sides of eq. (2.1) with a test-function q and eq. (2.2) with a test-function v respectively ($q, v \in C_C^\infty$) leads to

$$\begin{aligned}\operatorname{div}(u)\, q &= 0 \\ v\rho\left(u_t + u \cdot \nabla u\right) &= -v\nabla p + v\mu\nabla^2 u + vf.\end{aligned}$$

Integrating both sides and using integration by parts results in the weak formulation [Che20]

$$\langle \operatorname{div}(u), q\rangle = 0 \quad \forall q \tag{2.3}$$

$$\langle u_t, v\rangle - \langle \operatorname{div}(v), p\rangle + \langle \mu\nabla u, \nabla v\rangle + \langle u \cdot \nabla u, v\rangle = \langle f, v\rangle \quad \forall v. \tag{2.4}$$

2.2 Stokes

A simplified version of eqs. (2.1) and (2.2) can be derived for small Reynolds-numbers Re [Chi08]. Thus, linearizing eq. (2.2) leads to Stokes' equation

$$\operatorname{div}(u) = 0 \tag{2.5}$$
$$\nabla p - \mu \nabla^2 u = f. \tag{2.6}$$

2.2.1 Weak Formulation

Similar to Navier-Stokes above, we multiply both equations with a test-function, which leads to

$$\operatorname{div}(u)\, q = 0$$
$$\nabla p v - \mu \nabla^2 u v = f v.$$

Again, integrating both sides and using integration by parts results in the weak formulation

$$\langle \operatorname{div}(u), q \rangle = 0 \quad \forall q \tag{2.7}$$
$$-\langle \operatorname{div}(v), p \rangle + \langle \mu \nabla u, \nabla v \rangle = \langle f, v \rangle \quad \forall v. \tag{2.8}$$

If we now define

$$a(\cdot,\cdot) : V \times V \to \mathbb{R}, (u,v) \mapsto \langle \mu \nabla u, \nabla v \rangle,$$
$$b(\cdot,\cdot) : V \times Q \to \mathbb{R}, (u,q) \mapsto \langle \operatorname{div}(u), q \rangle,$$
$$f(\cdot) : V \to \mathbb{R}, u \mapsto \langle f, u \rangle,$$

the variational formulation can be written as

$$\begin{array}{rcrcl} a(u,v) & + & b(v,p) & = & f(v) \quad \forall v \\ b(u,q) & & & = & 0 \quad \forall q. \end{array}$$

Chapter 3

Numerical method

As both Stokes' and Navier-Stokes' equations are hard to solve analytically we will use the *finite element method* (FEM) to approximate a solution.
We will start this chapter by going over the FEM basics using the Poisson-equation as an example. Then, since the discretization of Stokes' equations results in a saddle-point-problem, we will discuss conditions for the existence of a unique solution for this kind of problem (namely the inf-sup/LBB condition). Furthermore we will take a look at block preconditioners and different ways to enforce Dirichlet boundary conditions.

3.1 Finite Element Method (FEM)

To understand the idea of the *finite element method* (FEM), let us consider one of the most basic PDEs, the Poisson-Equation with homogeneous Dirichlet boundary conditions. Let Ω be a domain with $\Omega \subset \mathbb{R}^n$,

$$-\Delta u = f \quad \forall x \in \Omega \tag{3.1}$$
$$u = 0 \quad \forall x \in \partial\Omega. \tag{3.2}$$

We start by constructing the weak formulation. Multiplying both sides of eq. (3.1) with a test function v and integrating by parts leads to the weak formulation: Find $u \in V = H_0^1(\Omega)$ such that

$$\langle \nabla u, \nabla v \rangle = \langle f, v \rangle \quad \forall v \in V.$$

To make the next steps easier to read, we define

$$a(\cdot,\cdot) : V^2 \to \mathbb{R}, (u, v) \mapsto \langle \nabla u, \nabla v \rangle$$
$$f(\cdot) : V \to \mathbb{R}, v \mapsto \langle f, v \rangle.$$

Now, using Lax-Milgram, guaranties a unique solution $u \in V$ of

$$a(u, v) = f(v) \quad \forall v \in V. \tag{3.3}$$

We will now try to find an approximation u_h to the true solution u using a finite dimensional subspace $V_h \subset V, h \in \mathbb{R}^+$. As $\dim(V_h) = N, N \in \mathbb{N}$, we can find basis functions $(\varphi^i)_{i=1}^N$ such that $V_h = \text{span}\left((\varphi^i)_{i=1}^N\right)$. Thus, elements v_h of V_h can be represented as linear combinations

$$v_h = \sum_{i=1}^{N} v^i \varphi^i$$

with coefficient vector $\underline{v}_h \coloneqq (v^i)_{i=1}^N$. The coefficient vector $\underline{v}_h$ can be identified with the finite element function v_h using the Galerkin isomorphism

$$G : \mathbb{R}^n \to V_h : \underline{v}_h \mapsto \sum_{i=1}^{N} v^i \varphi^i. \tag{3.4}$$

Using the linearity of a and f we can define

$$A_h \coloneqq (a(\varphi^i, \varphi^j))_{i,j=1}^N$$
$$f_h \coloneqq (f(\varphi^i))_{i=1}^N.$$

We can therefore find the finite element approximation $u_h \in V_h$, with coefficient vector $\underline{u}_h$, by solving a big linear system

$$A_h \underline{u}_h = f_h.$$

Note that A_h is typically a sparse matrix, allowing for fast solvers to be used.

3.1.1 Approximation error

The natural next question is how good u_h approximates the true solution u. If u is not only in the infinite dimensional space V but also the dense subspace V^+ an upper bound of the approximation error is given by [Sch99]

$$\inf_{v_h \in V_h} \|u - v_h\|_V \leq \delta(h) \|u\|_{V^+}$$

where $\delta(\cdot) : \mathbb{R}^+ \to \mathbb{R}^+$ has to fulfill

$$\delta(h) \to 0 \quad as \quad h \to 0.$$

3.2 Saddle point problems

Let $A_h \in \mathbb{R}^{n\times n}$ be a *symmetric and positive definite* (SPD) matrix and $\underline{u}_h, f_h \in \mathbb{R}^n$. It is well known that [Bra92] solving the linear system

$$A_h \underline{u}_h = f_h$$

is equivalent to the minimization problem

$$F(\underline{u}_h) := \frac{1}{2}\langle \underline{u}_h, \underline{u}_h\rangle_{A_h} - f_h^T \underline{u}_h \longrightarrow \min\,!\,.$$

If we now add a constraint $G(\underline{u}_h) = 0$ where

$$G(\underline{u}_h) := B_h \underline{u}_h - g_h,$$

$B_h \in \mathbb{R}^{m\times n}, g_h \in \mathbb{R}^m$ and $m \leq n$, a necessary condition for the existence of a local minimum of $F(\underline{u}_h)$ under the constraint $G(\underline{u}_h) = 0$ at a point $\underline{u}_h^0$ where dG is of full rank is that $(\underline{u}_h^0, \lambda)$ is a critical point of the Lagrangian

$$L(\underline{u}_h, \lambda) := F(\underline{u}_h) + \lambda^T (G(\underline{u}_h))$$

with Lagrange parameter $\lambda \in \mathbb{R}^m$.
Thus, differentiation of L with respect to $\underline{u}_h$ and λ and zeroing leads to the *Karush-Kuhn-Tucker* (KKT) equations

$$\begin{array}{lcll} A_h\underline{u}_h & + & B_h^T\lambda & = & f_h \\ B_h\underline{u}_h & & & = & g_h. \end{array}$$

Furthermore, the matrix

$$H_L = \begin{pmatrix} A_h & B_h^T \\ B_h & 0 \end{pmatrix} \in \mathbb{R}^{(n+m)\times(n+m)}$$

has n positive and m negative eigenvalues. Therefore, the linear system $H_L x = y$, $x, y \in \mathbb{R}^{n+m}$ cannot be solved using the *conjugate gradient method* (CG). We will use the *minimal residual method* (MinRes) instead.

3.2.1 Stokes

An example of a saddle-point problem are Stokes' equations which we already know from section 2.2 in their weak form as: Find $u \in V$ and $p \in Q$ such that

$$\begin{array}{lcll} a(u,v) & + & b(v,p) & = & \langle f, v\rangle \quad \forall v \in V \\ b(u,q) & & & = & 0 \quad \forall q \in Q. \end{array} \tag{3.5}$$

By defining $c((u,p),(v,q)) \coloneqq a(u,v) + b(v,p) + b(u,q)$, the sum of eqs. (3.5) can be written as

$$c((u,p),(v,q)) = \langle f, v\rangle \quad \forall (v,q) \in V \times Q. \tag{3.6}$$

If both stability,

$$\sup_{(u,p)\in V\times Q} \frac{c((u,p),(v,q))}{\|(u,p)\|_{V\times Q}} \geq c_1 \|(v,q)\|_{V\times Q} \quad \forall (v,q) \in V \times Q, \tag{3.7}$$

and continuity,

$$c((u,p),(v,q)) \leq c_2 \|(u,p)\|_{V\times Q} \|(v,q)\|_{V\times Q} \quad \forall (v,q),(u,p) \in V \times Q, \tag{3.8}$$

hold, eq. (3.6) has a unique solution, [Sch99].
As [Bre74] has shown, eq. (3.7) can be replaced by V_0-ellipticity of a (eq. (3.9)) and the inf-sup/LBB-condition for b (eq. (3.10)). Let V_0 be the kernel of b, i.e.

$$V_0 \coloneqq \{v \in V : b(v,q) = 0 \quad \forall q \in Q\},$$

then V_0-ellipticity of a reads

$$a(u,u) \geq c_a \|u\|_V^2 \quad \forall u \in V_0 \tag{3.9}$$

and the inf-sup/LBB condition

$$\sup_{u\in V} \frac{b(u,q)}{\|u\|_V} \geq c_b \|q\|_Q \quad \forall q \in Q. \tag{3.10}$$

If both a and b are continuous, c is, as the composition of continuous functions, also continuous. Furthermore, if a is elliptical on V and $V_h \subset V$, a is V_0-elliptical. Thus, for conforming methods with elliptical a the existence of a solution is only dependent on the LBB-condition.

Using a discretization A_h of a, B_h of b and f_h of f on the finite element spaces $V_h \subset V$ and $Q_h \subset Q$ leads to the matrix equation

$$\begin{pmatrix} A_h & B_h^T \\ B_h & 0 \end{pmatrix} \begin{pmatrix} \underline{u}_h \\ \underline{p}_h \end{pmatrix} = \begin{pmatrix} f_h \\ 0 \end{pmatrix}.$$

3.3 Block Preconditioner

As we want to use the MinRes method to approximate a solution to the linear system $D_h x = y$ where

$$D_h \coloneqq \begin{pmatrix} A_h & B_h^T \\ B_h & G_h \end{pmatrix},$$

we will use a simple block Jacobi preconditioner to lower the number of iterations. A block LDU decomposition shows that

$$D_h^{-1} = \begin{pmatrix} I & 0 \\ -B_h A_h^{-1} & I \end{pmatrix} \begin{pmatrix} A_h^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & -A_h^{-1} B_h^T \\ 0 & I \end{pmatrix}$$

with the Schur compliment $S_h \coloneqq G_h - B_h A_h^{-1} B_h^T$. Thus, given a preconditioner $\tilde{A}_h$ for A_h and $\tilde{S}_h$ for S_h we can use the block matrix

$$\begin{pmatrix} \tilde{A}_h & 0 \\ 0 & \tilde{S}_h \end{pmatrix} \tag{3.11}$$

as a preconditioner for D_h.

3.4 Boundary Conditions

In the following, we will use two different methods to enforce Dirichlet *boundary conditions* (BCs). Consider the following problem:

$$\begin{aligned} a(u, v) &= \langle f, v \rangle \quad \forall v \in V \\ u &= g \quad \text{on} \quad \Gamma_{\mathrm{D}}. \end{aligned} \tag{3.12}$$

We will either directly resolve the boundary conditions on the mesh or add a penalty term on Γ_{D}.

3.4.1 Mesh

We reduce the problem with non-homogeneous Dirichlet boundary conditions to one with homogeneous BCs. Considering (3.12) we start by extending g to the inside of Ω. We can now split the true solution u in

$$u = u_0 + u_D$$

where u_D is the extension of g and $u_0 = 0$ on Γ_D.
Given discretizations A_h, f_h, of a, f and coefficient vectors $\underline{u}_0, \underline{u}_D$ this ansatz leads to

$$A_h(\underline{u}_0 + \underline{u}_D) = f_h.$$

Using the linearity of A_h yields

$$A_h(\underline{u}_0) = f_h - A_h(\underline{u}_D). \tag{3.13}$$

Now, splitting the *degrees of freedom* (dofs) into free (F) and Dirichlet (D) dofs, and pretending that the dofs are numbered accordingly, eq. (3.13) can be rewritten in block form as

$$\begin{pmatrix} A_h^{(FF)} & A_h^{(FD)} \\ A_h^{(DF)} & A_h^{(DD)} \end{pmatrix} \begin{pmatrix} \underline{u}_0^{(F)} \\ 0 \end{pmatrix} = \begin{pmatrix} f_h^{(F)} \\ f_h^{(D)} \end{pmatrix} - \begin{pmatrix} A_h^{(FF)} & A_h^{(FD)} \\ A_h^{(DF)} & A_h^{(DD)} \end{pmatrix} \begin{pmatrix} \underline{u}_D^{(F)} \\ \underline{u}_D^{(D)} \end{pmatrix}. \tag{3.14}$$

To compute u we need to solve for u_0. Using eq. (3.14) we know that

$$A_h^{(FF)} \underline{u}_0^{(F)} = f_h^{(F)} - \begin{pmatrix} A_h^{(FF)} & A_h^{(FD)} \end{pmatrix} \begin{pmatrix} \underline{u}_D^{(F)} \\ \underline{u}_D^{(D)} \end{pmatrix}.$$

Thus, defining

$$r \coloneqq f_h - A_h \underline{u}_D$$

allows us to calculate $\underline{u}$ as

$$\underline{u} = \underline{u}_D + \begin{pmatrix} {A_h^{(FF)}}^{-1} & 0 \\ 0 & 0 \end{pmatrix} r.$$

3.4.2 Penalty

As shown in [Bra92], solving (3.12) is equivalent to solving the minimization problem

$$F(u) \coloneqq \frac{1}{2} a(u,u) - \langle f, u \rangle \longrightarrow \min ! \tag{3.15}$$

under the constraint

$$u = g \quad \text{on} \quad \Gamma_\mathrm{D}. \tag{3.16}$$

As discussed in [UC82], eq. (3.16) can be enforced by adding a penalty term in eq. (3.15), namely

$$F(u) \coloneqq \frac{1}{2} a(u,u) - \langle f, u \rangle + \frac{\varepsilon}{2} \langle u - g, u - g \rangle_{\Gamma_\mathrm{D}} \longrightarrow \min ! \tag{3.17}$$

with penalty parameter ε. Similar to [BE86] we will choose $\varepsilon = h^2$ and only use low order fe-spaces.

Chapter 4

Space discretization

If both stability eq. (3.7) and continuity eq. (3.8) hold, the saddle point problem (3.5) has a unique solution. Similarly, discrete counterparts to eqs. (3.7) and (3.8) have to hold for a unique solution to exist in the finite element spaces $V_h \subset V$ and $Q_h \subset Q$. A commonly used stable finite element discretization for saddle point problems is the Taylor-Hood pair of order $k \in \mathbb{N}$.
Although, other methods exist, they differ not only in the size of the resulting linear system but also in mass conservation (divergence free on elements or on a global level).

4.1 Taylor-Hood elements

There are different ways to ensure the LBB-condition eq. (3.10) holds. Namely increasing the size of the velocity space V_h and decreasing the size of the pressure space Q_h.
Given a triangulation T_h of the domain $\Omega \subset \mathbb{R}^n$, a well known example of a stable finite element discretization for saddle point problems is the Taylor-Hood pair of order $k \in \mathbb{N}$

$$\begin{aligned} V_h &= \{v \in H_0^1(\Omega)^n : v|_T \in \mathcal{P}_{k+1}(T)^n \quad \forall T \in T_h\} \\ Q_h &= \{q \in L_0^2(\Omega) : q|_T \in \mathcal{P}_k(T) \quad \forall T \in T_h\}. \end{aligned}$$

Theorem 1. *[Neu17] The Taylor-Hood $\mathcal{P}_2/\mathcal{P}_1$ discretization has a quadratic convergence rate, assuming the solution is stable and smooth enough*

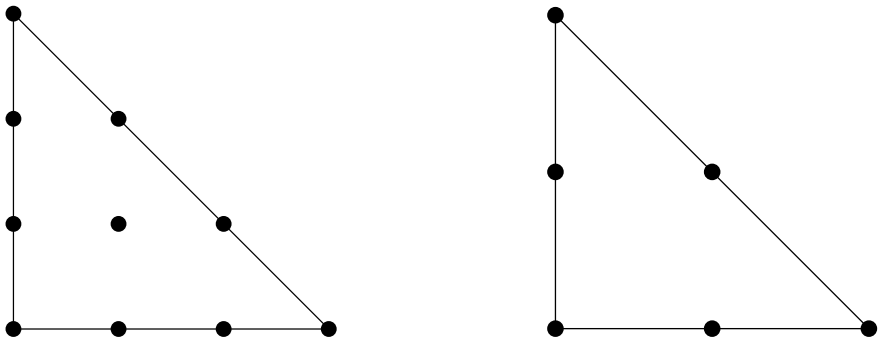
$$\|u - u_h\|_{H^1(\Omega)} + \|p - p_h\|_{L^2(\Omega)} \leq h^2|u|_{H^3(\Omega)} + h^2|p|_{H^2(\Omega)}. \tag{4.1}$$

Another type of pairing for the velocity and the pressure is to use a complete discontinuous pressure and two additional polynomial degrees for the velocity, $\mathcal{P}_2/\mathcal{P}_0^{dc}$.

For this pairing the LBB-condition can be proven in an easy way, but the convergence rate is only linear due to the poor approximation of the pressure

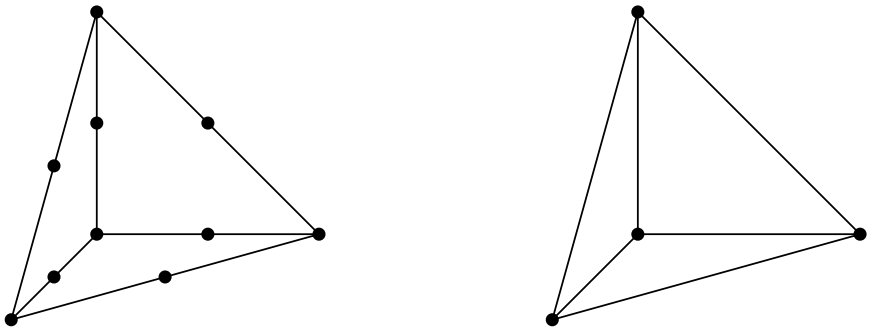
$$\|p - p_h\|_{L^2(\Omega)} \leq h|p|_{H^1(\Omega)}. \tag{4.2}$$

While a pairing $\mathcal{P}_k/\mathcal{P}_k$ is unstable, the pair $\mathcal{P}_{k+2}/\mathcal{P}_k$ is wasteful as the accuracy is limited by the low order pressure space. Stable Taylor-Hood elements in $2D$ and $3D$ are illustrated in figs. 4.1 and 4.2.



(a) $\mathcal{P}_3$ element for velocity (b) $\mathcal{P}_2$ element for pressure

Figure 4.1: $\mathcal{P}_3/\mathcal{P}_2$ Taylor-Hood elements in 2D



(a) $\mathcal{P}_2$ element for velocity (b) $\mathcal{P}_1$ element for pressure

Figure 4.2: $\mathcal{P}_2/\mathcal{P}_1$ Taylor-Hood elements in 3D

As the use of a Taylor-Hood pair of order k requires a higher order space for the velocity component, the computational complexity is increased compared to the pair $\mathcal{P}_k/\mathcal{P}_k$. A comparison of the $ndofs$ for $\mathcal{P}_2$ and $\mathcal{P}_1$ elements can be seen in table 4.1 below.

	ndof per $\mathcal{P}_2$ element (u)	ndof per $\mathcal{P}_1$ element (p)
2D	6	3
3D	10	4

Table 4.1: Comparison of $ndofs$ for $\mathcal{P}_2$ and $\mathcal{P}_1$ elements in $2D$ and $3D$.

4.2 Simple Elements with Stabilization

As suggested by [BF91], the key to successfully stabilizing incompressible elements is to weaken the discrete divergence-free condition. Increasing either the size of the velocity space V_h or decreasing the size of the pressure space Q_h results in a stable method, as discussed in the previous chapter. However, since the accuracy is limited by the lower order space Q_h, we want to consider the pairing $\mathcal{P}_1/\mathcal{P}_1$ to decrease the required computational cost compared to Taylor-Hood elements. To stabilize this pairing we will explicitly weaken the divergence-free condition

$$b(u, q) = 0 \quad \forall q \in Q \tag{4.3}$$

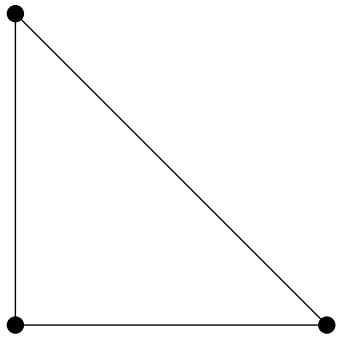
by adding a function $g := \alpha \sum_T h_T^2 \int_T (\nabla p_h, \nabla q_h)$ as in [BP84]:

$$b(u, q) - g(p, q) = 0 \quad \forall q \in Q. \tag{4.4}$$

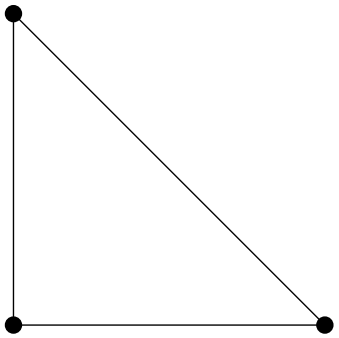
Given a discretization C_h of $-g$, the new problem can be expressed in matrix form as

$$\begin{pmatrix} A_h & B_h \\ B_h^T & C_h \end{pmatrix} \begin{pmatrix} \underline{u}_h \\ \underline{p}_h \end{pmatrix} = \begin{pmatrix} f_h \\ 0 \end{pmatrix}. \tag{4.5}$$

As shown in [BF91], this method is stable for all sufficiently small choices of $\alpha > 0$. Note that we have added a consistency error of order h.
Both $2D$ and $3D$ - $\mathcal{P}_1/\mathcal{P}_1$ elements can be seen in figs. 4.3 and 4.4 below.

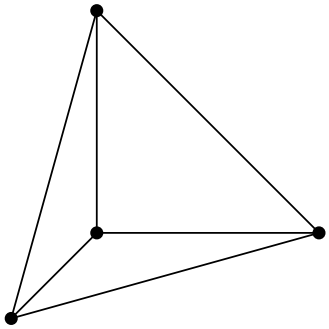


(a) $\mathcal{P}_1$ element for velocity

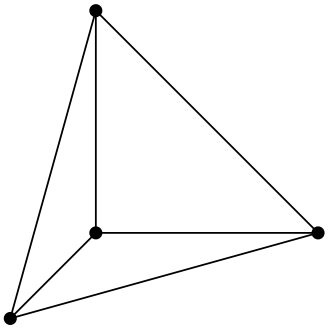


(b) $\mathcal{P}_1$ element for pressure

Figure 4.3: $\mathcal{P}_1/\mathcal{P}_1$ elements in 2D



(a) $\mathcal{P}_1$ element for velocity



(b) $\mathcal{P}_1$ element for pressure

Figure 4.4: $\mathcal{P}_1/\mathcal{P}_1$ elements in 3D

4.3 $H(div)$-conforming elements

As a comparison (i.e. to show limitations of stabilized low order element pairings) we will use $H(div)$-conforming elements which allow us to achieve a solution that is exact divergence-free. We will, without discussing it in too much detail, use the method introduced by in [Leh10] and [LS16].
Before taking a look at the fe-spaces used, we have to introduce several new notations. Given a triangulation $\mathcal{T}_h$ of a domain $\Omega \in \mathbb{R}^n$ we define the skeleton $\mathcal{F}_h$ of $\mathcal{T}_h$ in $2D/3D$ as the set of all edges/faces of $\mathcal{T}_h$. Furthermore, we define the normal-jump $[\![u]\!]_n$ as the difference of u_h from one element T to a neighboring element T' perpendicular to their interface E as

$$[\![u]\!]_n = [\![u_h \cdot n]\!]_E \coloneqq (u_h|_T - u_h|_{T'})\,|_E \cdot n_1. \tag{4.6}$$

While the Sobolev-space $H(div, \Omega)$ is defined as

$$H(div, \Omega) \coloneqq \left\{ u \in \left[L^2(\Omega) \right]^n : \operatorname{div}(u) \in L^2(\Omega \right\}, \tag{4.7}$$

we introduce two new spaces

$$W_h \coloneqq \left\{ u \in \left[\Pi^k(\mathcal{T}_h) \right]^2 : [\![u_h \cdot n]\!]_E, \forall E \in \mathcal{F}_h \right\} \tag{4.8}$$

$$F_h \coloneqq \left\{ \widehat{u}_h \in \left[\Pi^k(\mathcal{F}_h) \right]^n : \widehat{u}_h \cdot n = 0 \right\} \tag{4.9}$$

and define the velocity space V_h as

$$V_h \coloneqq W_h \times F_h. \tag{4.10}$$

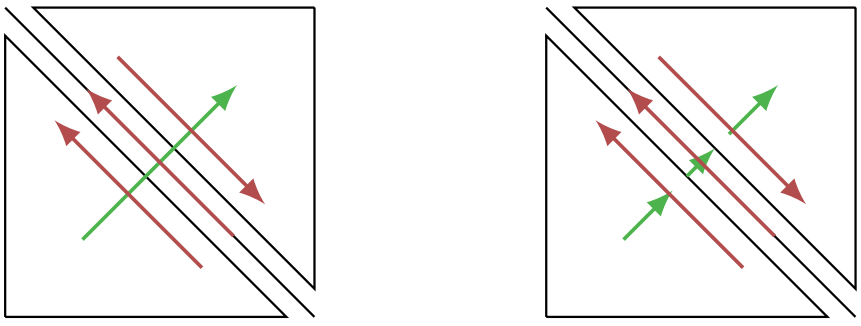
Thus, using polynomials of order up to $k-1$ as the pressure space

$$Q_h \coloneqq \Pi^{k-1}(\mathcal{T}_h), \tag{4.11}$$

we see that $\operatorname{div}(W_h) = Q_h$.
Therefore, as the divergence-free condition eq. (2.3) has to hold for all $q_h \in Q_h$, choosing $q_h = \operatorname{div}(u_h)$ leads to

$$0 = \langle \operatorname{div}(u_h), q_h \rangle = \langle \operatorname{div}(u_h), \operatorname{div}(u_h) \rangle \tag{4.12}$$

and therefore $\operatorname{div}(u_h) = 0$ point-wise.
This normal-continuity, together with the tangential-continuity, is illustrated in fig. 4.5 below.



(a) $H(div)$-conforming HDG (b) HDG

Figure 4.5: HDG elements in $2D$

Chapter 5

Time discretization

Until now we considered laminar flow i.e. the material derivative $du/dt = 0$. In order to cope with turbulent flow i.e. approximate a solution for Navier-Stokes' equations, we will now introduce explicit, implicit, and implicit-explicit time stepping methods.
It is well known that Runge-Kutta methods can be used to approximate a solution $u : \mathbb{R} \to \mathbb{R}^n$ given an initial value problem

$$\begin{aligned} \dot{u} &= f(t, u) \\ u(t_0) &= u_0. \end{aligned}$$

In this chapter we will use the notation $\dot{u} \coloneqq \partial u/\partial t$.

5.1 Implicit-Explicit Runge-Kutta Method

Applying any one of the space discretizations discussed in chapter 4 to the Navier-Stokes' equations (2.3) and (2.4) yields a large *ordinary differential equation* (ODE) system

$$M_h \underline{\dot{u}}_h + A_h^{(diff)} \underline{u}_h + A_h^{(conv)} \underline{u}_h = f_h, \tag{5.1}$$

where $A_h^{(diff)}$ corresponds to the diffusion term, $A_h^{(conv)}$ corresponds to the convection term, and the mass-matrix M_h is a discretization of the operator $M : V \to V : u \mapsto Mu : (Mu, v)_V \mapsto \langle u, v \rangle, \forall v \in V$. Replacing the time derivative $\dot{u}_h$ at time t_n with a finite difference approximation

$$\underline{\dot{u}}_h(t_n) \approx (u_h^{(n+1)} - u_h^{(n)})/\tau \tag{5.2}$$

for time-steps τ, discrete points in time $t_{n+1} \coloneqq t_n + \tau$, and $u_h^{(n)} \coloneqq u_h(t_n)$ leads to

$$M_h \frac{\underline{u}_h^{(n+1)} - \underline{u}_h^{(n)}}{\tau} + A_h^{(diff)} \underline{u}_h + A_h^{(conv)} \underline{u}_h = f_h. \tag{5.3}$$

Looking at the time-step restrictions provided in [Leh10] for both the convection term

$$\Delta t \leq c_C \frac{h}{p^2} \frac{1}{|u|} \tag{5.4}$$

as well as the diffusion term

$$\Delta t \leq c_B \frac{h^2}{p^4} \frac{1}{\nu} \tag{5.5}$$

with constants $c_C, c_B \in \mathbb{R}$, mesh size h, and polynomial order p, when using fully explicit methods, it is clear why we do not want to use an *explicit Runge-Kutta* (ERK) scheme. That is to avoid the time-step restriction of the diffusion term, as it scales with $(h/p^2)^2$ compared to the convection terms h/p^2.
We can now try to use an implicit time-stepping method for the diffusion term, avoiding time-step restrictions imposed by eq. (5.5), and an explicit one for the convection term. We will use compatible IMEX methods introduced in [ARS97].

5.1.1 Implicit-Explicit Euler

Using the stabilized $\mathcal{P}_1/\mathcal{P}_1$ elements from section 4.2, together with the notation used in the previous chapters yields the ODE:

$$\begin{pmatrix} M_h & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{\underline{u}_h^{(n+1)} - \underline{u}_h^{(n)}}{\tau} \\ \underline{p}_h^{(n+1)} \end{pmatrix} + \underbrace{\begin{pmatrix} A_h & B_h^T \\ B_h & G_h \end{pmatrix}}_{=:D_h} \begin{pmatrix} \underline{u}_h^{(n+1)} \\ \underline{p}_h^{(n+1)} \end{pmatrix} = \begin{pmatrix} f_h(\underline{u}_h^{(n)}) - C_h(\underline{u}_h^{(n)}) \\ 0 \end{pmatrix}. \tag{5.6}$$

Similarly to [Neu17], we will treat the left hand side of eq. (5.6) implicitly and the right hand side explicitly. Thus we use $\underline{u}_h^{(n+1)}$ and $\underline{p}_h^{(n+1)}$ for both the mass matrix as well as the Stokes block D_h, while we use $\underline{u}_h^{(n)}$ for the linear-form f_h and the convection term C_h.
Now, using the linearity of the operators, combining the Mass matrix with D_h and multiplying both sides of the equation with the step-size τ leads to

$$\underbrace{\begin{pmatrix} M_h + \tau A_h & \tau B_h^T \\ \tau B_h & \tau G_h \end{pmatrix}}_{=:M_h^*} \begin{pmatrix} \underline{u}_h^{(n+1)} - \underline{u}_h^{(n)} \\ \underline{p}_h^{(n+1)} \end{pmatrix} = \tau \begin{pmatrix} f_h(\underline{u}_h^{(n)}) - C_h(\underline{u}_h^{(n)}) - A_h(\underline{u}_h^{(n)}) - B_h^T(\underline{p}_h^{(n)}) \\ -B_h(\underline{u}_h^{(n)}) \end{pmatrix}. \tag{5.7}$$

As we prefer to have the equations in update form, we add $-\underline{p}_h^{(n)}$ on both sides and define $\tilde{f}_h \coloneqq (f_h \quad 0)^T$ and $\tilde{C}_h \coloneqq (C_h \quad 0)^T$:

$$M_h^* \begin{pmatrix} \underline{u}_h^{(n+1)} - \underline{u}_h^{(n)} \\ \underline{p}_h^{(n+1)} - \underline{p}_h^{(n)} \end{pmatrix} = -\tau \left(D_h(\underline{u}_h^{(n)}, \underline{p}_h^{(n)}) - \tilde{f}_h + \tilde{C}_h \right). \tag{5.8}$$

Thus, the new iterates $\underline{u}_h^{(n+1)}$ and $\underline{p}_h^{(n+1)}$ can be computed by solving

$$\begin{pmatrix} \underline{u}_h^{(n+1)} \\ \underline{p}_h^{(n+1)} \end{pmatrix} = \begin{pmatrix} \underline{u}_h^{(n)} \\ \underline{p}_h^{(n)} \end{pmatrix} - \tau (M_h^*)^{-1} \left(D_h(\underline{u}_h^{(n)}, \underline{p}_h^{(n)}) - \tilde{f}_h + \tilde{C}_h \right). \tag{5.9}$$

Note that although the block matrix containing M_h is not regular, the implicit-explicit Euler scheme can be used as it only requires the inverse of M_h^*. Thus, it can be used if M_h^* is regular.

5.1.2 L-stable, two-stage, second-order DIRK (2,3,2)

Similar to explicit Runge-Kutta methods, we can summarize implicit-explicit Runge-Kutta methods using Butcher-tableaus. In this case, we combine a two-stage, second-order, *singly diagonally implicit Runge-Kutta* (SDIRK) scheme with a corresponding three-stage, second-order *explicit Runge-Kutta* (ERK), resulting in a second-order *implicit-explicit* (IMEX) scheme, which can be represented by the following butcher tableau [ARS97]:

$$\begin{array}{c|cc} \gamma & \gamma & 0 \\ 1 & 1-\gamma & \gamma \\ \hline & 1-\gamma & \gamma \end{array} \qquad \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \gamma & \gamma & 0 & 0 \\ 1 & \delta & 1-\delta & 0 \\ \hline & 0 & 1-\gamma & \gamma \end{array} . \tag{5.10}$$

Given two matching Butcher-tableaus, one step of the IMEX scheme can be written as in algorithm 1.

Algorithm 1 One step from t_{n-1} to $t_n = t_{n-1} + \tau$ of the IMEX scheme [ARS97]

1: $\widehat{\underline{K}}_1 = f(u^{(n-1)})$
2: **for** $i = 1, \dots, s$ **do**
3: $\quad \underline{K}_i = g(u_i)$, where
4: $\quad u_i = u^{(n-1)} + \tau \sum_{j=1}^{i} a_{i,j} \underline{K}_j + \tau \sum_{j=1}^{i} \widehat{a}_{i+1,j} \widehat{\underline{K}}_j$
5: $\quad \widehat{\underline{K}}_{i+1} = f(u_i)$
6: $u^{(n)} = u^{(n-1)} + \tau \sum_{j=1}^{s} b_j \underline{K}_j + \tau \sum_{j=1}^{\sigma} \widehat{b}_j \widehat{\underline{K}}_j$

In our case $f \coloneqq -M^{-1}A_{conv}$ and $g \coloneqq -M^{-1}A_{diff}$.
To avoid repeated matrix-vector multiplications, we will adapt algorithm 1 using the linearity of both M^{-1} and A_{diff}:

$$\begin{aligned}
u_i &= u^{(n-1)} + \tau \sum_{j=1}^{i} a_{i,j}\underline{K}_j + \tau \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{\underline{K}}_j \\
\implies u_i &= u^{(n-1)} + \tau \sum_{j=1}^{i-1} a_{i,j}\underline{K}_j + \tau \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{\underline{K}}_j - \tau a_{i,i}M^{-1}A_{diff}u_i \\
\implies u_i + \tau a_{i,i}M^{-1}A_{diff}u_i &= u^{(n-1)} + \tau \sum_{j=1}^{i-1} a_{i,j}\underline{K}_j + \tau \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{\underline{K}}_j.
\end{aligned}$$

Multiplying both sides with the linear operator M from the left leads to

$$Mu_i + \tau a_{i,i}A_{diff}u_i = Mu^{(n-1)} + \tau M \sum_{j=1}^{i-1} a_{i,j}\underline{K}_j + \tau M \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{\underline{K}}_j.$$

Now, as $\underline{K}_j = -M^{-1}A_{diff}u_j$ and $\widehat{\underline{K}}_{j+1} = -M^{-1}A_{conv}u_j$, we first factor out the M^{-1} hidden inside $\underline{K}_j$ and $\widehat{\underline{K}}_j$ and then define $K_j \coloneqq -A_{diff}u_j$ and $\widehat{K}_{j+1} \coloneqq -A_{conv}u_j$. This yields

$$\begin{aligned}
(M + \tau a_{i,i}A_{diff})u_i &= Mu^{(n-1)} + \tau MM^{-1} \sum_{j=1}^{i-1} a_{i,j}K_j + \tau MM^{-1} \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{K}_j \\
\implies (M + \tau a_{i,i}A_{diff})u_i &= Mu^{(n-1)} + \tau \left(\sum_{j=1}^{i-1} a_{i,j}K_j + \sum_{j=1}^{i} \widehat{a}_{i+1,j}\widehat{K}_j \right).
\end{aligned}$$

Using K_j and $\widehat{K}_j$ instead of $\underline{K}_j$ and $\widehat{\underline{K}}_j$ we also obtain

$$u^{(n)} = u^{(n-1)} + \tau M^{-1} \left(\sum_{j=1}^{s} b_j K_j + \sum_{j=1}^{\sigma} \widehat{b}_j \widehat{K}_j \right).$$

These results lead to algorithm 2.

Algorithm 2 Modified version of Algorithm 1

1: $\widehat{K}_1 = -A_{conv}(u^{(n-1)})$
2: **for** $i = 1, \dots, s$ **do**
3: $\quad K_i = -A_{diff}(u_i)$, where
4: $\quad (M + \tau a_{i,i} A_{diff}) u_i = M u^{(n-1)} + \tau \left(\sum_{j=1}^{i-1} a_{i,j} K_j + \sum_{j=1}^{i} \widehat{a}_{i+1,j} \widehat{K}_j \right)$
5: $\quad \widehat{K}_{i+1} = -A_{conv}(u_i)$
6: $u^{(n)} = u^{(n-1)} + \tau M^{-1} \left(\sum_{j=1}^{s} b_j K_j + \sum_{j=1}^{\sigma} \widehat{b}_j \widehat{K}_j \right)$

Because we have chosen our Butcher-tableau such that $\widehat{b}_{s+1} = 0$, $b_j = a_{s,j}$, and $\widehat{b}_j = \widehat{a}_{s+1,j}$ for $j = 1, \dots, s$, we see that $u^{(n)} = u_s$. We can therefore omit line 6 of algorithm 2, eliminating the need to invert M. However, as we no longer compute the new iterate in update form, we will change line 4 to retain non-homogeneous Dirichlet BCs. Furthermore, as $a_{i,i} = a_{j,j}$ for $i, j = 1, \dots, s$, $M^* := (M + \tau a_{i,i} A_{diff})$ does not change between stages. We define $a^* := a_{i,i}$ and update algorithm 2 as can be seen in algorithm 3 below.

Algorithm 3 Modified version of Algorithm 1

1: $\widehat{K}_1 = -A_{conv}(u^{(n-1)})$
2: $(M^*)^{-1} = (M + \tau a^* A_{diff})^{-1}$
3: **for** $i = 1, \dots, s$ **do**
4: $\quad u_i = u^{(n-1)}$
5: $\quad u_i \mathrel{+}= (M^*)^{-1} \left(M u^{(n-1)} + \tau \left(\sum_{j=1}^{i-1} a_{i,j} K_j + \sum_{j=1}^{i} \widehat{a}_{i+1,j} \widehat{K}_j \right) - M^* u^{(n-1)} \right)$
6: $\quad K_i = -A_{diff}(u_i)$
7: $\quad \widehat{K}_{i+1} = -A_{conv}(u_i)$
8: $u^{(n)} = u_s$

Algorithm 3 can be implemented as seen in listing 5.1 below:

```
def imex_solver(currentApprox, timeStep, M, mstar, invMStar, Adiff
    , Aconv, butcher, f=0):
    k = [currentApprox.CreateVector() for i in range(butcher.
    stages)]
    kHat = [currentApprox.CreateVector() for i in range(butcher.
    stages+1)]

    ui = currentApprox.CreateVector()
    rhsi = currentApprox.CreateVector()
    Mun = currentApprox.CreateVector()

    M.Apply(currentApprox, Mun)
```

```
    Aconv.Apply(currentApprox, kHat[0])
    kHat[0] *= -1

    for i in range(butcher.stages):
        rhsi[:]=0
        for j in range(i):
            if(butcher.aImp[i][j] != 0):
                rhsi.data += butcher.aImp[i][j] * k[j]
        for j in range(i+1):
            #print("explicit i= ", i,"j= ",  j)
            if (butcher.aExp[i+1][j] != 0):
                rhsi.data += butcher.aExp[i+1][j] * kHat[j]

        rhsi.data *= timeStep
        rhsi.data += Mun

        #solve MStar * ui = rhsi for ui:
        #keep nonhomogenous Dirichlet bcs
        ui.data = currentApprox
        ui.data += invMStar * (rhsi - mstar.mat*currentApprox)

        Adiff.Apply(ui, k[i])
        k[i] *= -1
        Aconv.Apply(ui, kHat[i+1])
        kHat[i+1] *= -1
    # return un=us since cs=1 for implicit scheme (Asher 2.3)
    return ui
```

Listing 5.1: Implementation of algorithm 2.

Chapter 6

Numerical results

Now that we have acquired the necessary theory to approximate solutions for both Stokes' and Navier-Stokes' equations on an fe-space, we will solve several standard problems.We will use the open source finite element library *NGSolve* and mesh-handler *Netgen* [19; Sch97].
We start with the 2D Schäfer-Turek benchmark geometry described in [Sch+96],

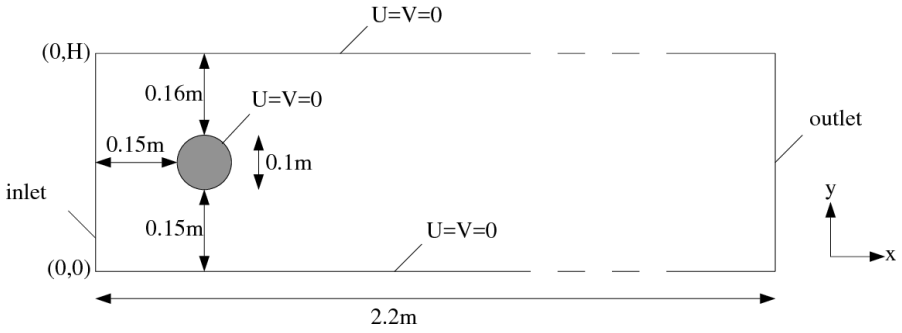


Figure 6.1: Geometry of 2D test cases with boundary conditions [Sch+96]

with the initial value problem

$$\begin{aligned} u(x,y) &= 0 && \text{on} \quad \Gamma_{\mathrm{D}} \\ \frac{\partial u}{\partial t}(x,y) &= 0 && \text{on} \quad \Gamma_{\mathrm{outlet}} \\ u(x,y) &= \left(\tfrac{4U_m y(0.41-y)}{0.41^2}, 0\right) && \text{on} \quad \Gamma_{\mathrm{inlet}}. \end{aligned}$$

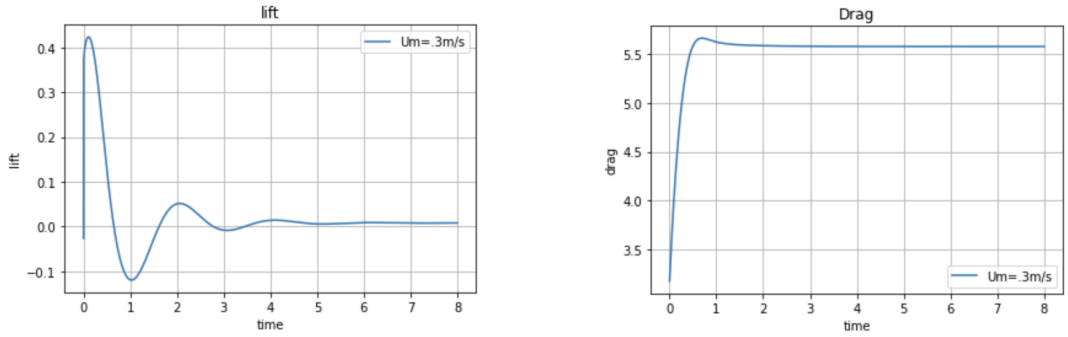
Problem A
Using $U_m = 0.3m/s$ results in a Reynolds number $Re = 20$. We therefore expect

a steady solution. We use the *IMEX*-scheme described in section 5.1.2 to calculate both the lift coefficient c_L and the drag coefficient c_D for each space discretization discussed in chapter 4. The results are shown in table 6.1.

Discretization	mesh-elements	$ndofs$	c_D	c_L
$\mathcal{P}_1/\mathcal{P}_1$ - with stabilization	416	747	6.4417	0.0759
$\mathcal{P}_2/\mathcal{P}_1$ Taylor-Hood	416	2077	5.6101	0.0383
$\mathcal{P}_3/\mathcal{P}_2$ Taylor-Hood	416	4904	5.5768	0.0078
$H(div)$-conforming	416	11144	3.6206	0.0025
$\mathcal{P}_1/\mathcal{P}_1$ - with stabilization	4526	7173	5.7491	0.0225
$\mathcal{P}_2/\mathcal{P}_1$ Taylor-Hood	4526	21007	5.5778	0.0102
$\mathcal{P}_3/\mathcal{P}_2$ Taylor-Hood	4526	50810	5.5790	0.0107
$H(div)$-conforming	4526	118544	3.6253	0.0110

Table 6.1: Results of the Schäfer-Turek benchmark with $U_m = 0.3m/s$

Looking at the drag and lift coefficients in fig. 6.2 we see the expected steady-state flow. Both lift and drag remain almost constant once the steady state is reached.



(a) Lift coefficient c_L (b) Drag coefficient c_D

Figure 6.2: Lift and drag coefficients using $\mathcal{P}_3/\mathcal{P}_2$ Taylor-Hood elements

The norm of the resulting velocity $\|u_h\|$ at time $t = 8s$ is illustrated in fig. 6.3, showing a laminar steady flow.

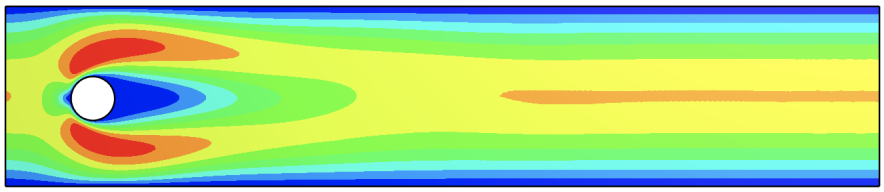


Figure 6.3: Resulting laminar steady flow ($\|u_h\|$) at $t = 8s$ for $U_m = 0.3m/s$

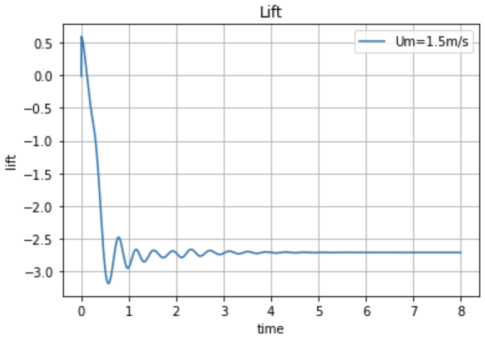
Problem B

We will now increase the inflow velocity U_m to $1.5m/s$ to achieve a Reynolds-number Re of 100. We expect turbulent flow. Again, the lift and drag coefficients for each space discretization discussed in chapter 4 can be seen in table 6.2 below.

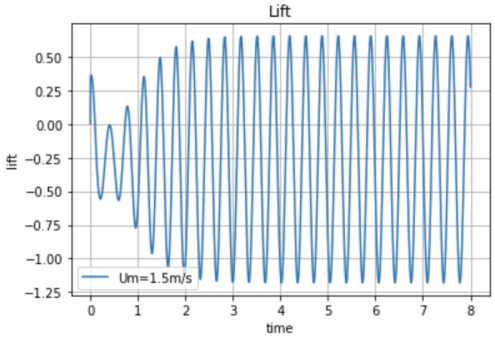
Discretization	mesh-elements	$ndofs$	c_{Dmax}	c_{Lmax}
$\mathcal{P}_1/\mathcal{P}_1$ - with stabilization	416	747	3.3862	0.5991
$\mathcal{P}_2/\mathcal{P}_1$ Taylor-Hood	416	2077	3.1325	0.4197
$\mathcal{P}_3/\mathcal{P}_2$ Taylor-Hood	416	4904	3.2806	1.0771
$\mathcal{P}_1/\mathcal{P}_1$ - with stabilization	4526	7173	3.5220	0.6091
$\mathcal{P}_2/\mathcal{P}_1$ Taylor-Hood	4526	21007	3.2199	0.5224
$\mathcal{P}_3/\mathcal{P}_2$ Taylor-Hood	4526	50810	3.2601	1.1003
$H(div)$-conforming	4526	118544	2.5347	0.9814

Table 6.2: Results of the Schäfer-Turek benchmark with $U_m = 1.5m/s$

If we plot the lift coefficient vs time, we see that we do not get the expected oscillations when using low order elements on a coarse mesh. This problem is alleviated by switching to a finer mesh as shown in fig. 6.4. As we want to use these elements specifically with fine meshes this is OK.



(a) Lift coefficient c_L on a coarse mesh



(b) Lift coefficient c_L on a fine mesh

Figure 6.4: Lift coefficient using $\mathcal{P}_1/\mathcal{P}_1$ elements with stabilization

The norm of the solution $\|u_h\|$ is depicted in fig. 6.5. We see the formation of a vortex-street as expected for Reynolds-numbers between around 45 and 150.

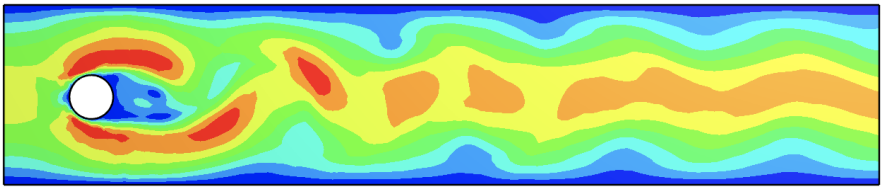
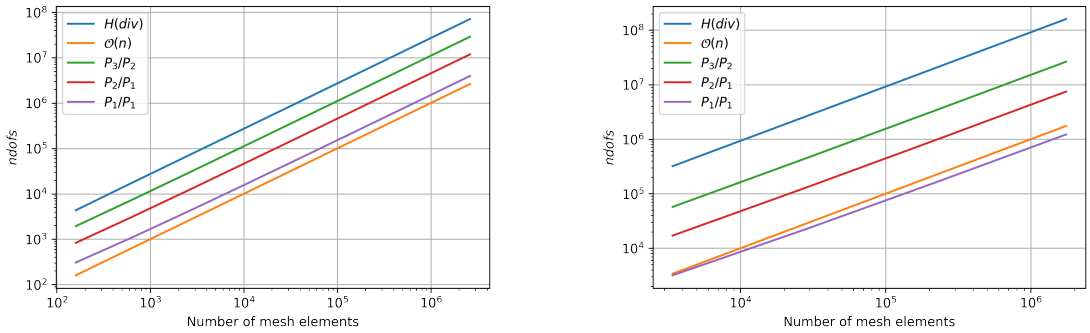


Figure 6.5: Resulting flow $\|u_h\|$ at $t = 8s$ for $U_m = 1.5m/s$

Problem C

Let us now examine how the $ndofs$ change with respect to the number of elements in our mesh. Figure 6.6 shows the behavior for each discretization discussed in both two and three spacial dimensions. The difference between lowest order element pairings and discretization approaches that increase the number of $ndofs$ to weaken the divergence-free condition is especially noticeable in $3D$.

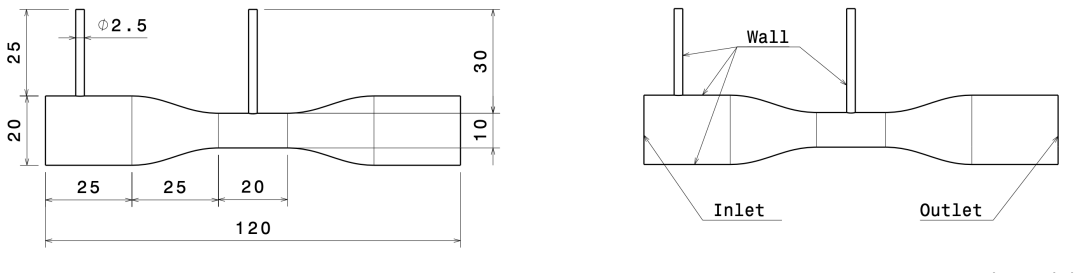


(a) 2-dimensions (b) 3-dimensions

Figure 6.6: Comparison of $ndofs$ per mesh elements for different fe-spaces

Problem D

Let us now consider the flow of a fluid through a pipe with a choke in 3D as depicted in fig. 6.7. We expect to see the Venturi effect, that is, a reduction in fluid-pressure as the fluid-velocity increases.



(a) Measurements (b) Boundary names

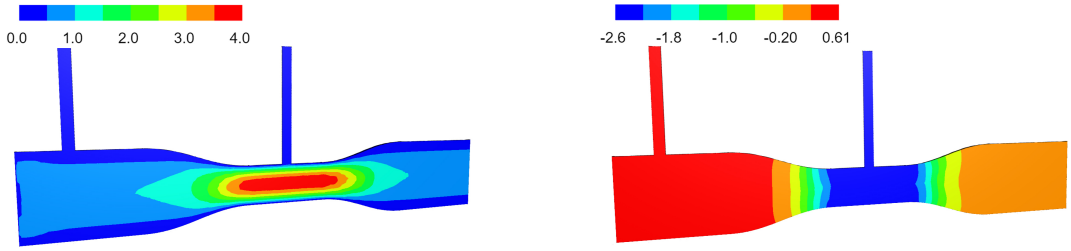
Figure 6.7: Pipe with a constricted section (2D projection)

We solve Navier-Stokes' equations for the initial values,

$$\begin{aligned} u &= 0 && \text{on} \quad \Gamma_{\text{wall}} \\ u_t &= 0 && \text{on} \quad \Gamma_{\text{outlet}} \\ u &= -n && \text{on} \quad \Gamma_{\text{inlet}}, \end{aligned}$$

with outward facing normal vector n.

The numerical results (fig. 6.8) agree with our expectations. Figure 6.8 clearly shows a negative pressure, as the fluid flows through the constricted section.

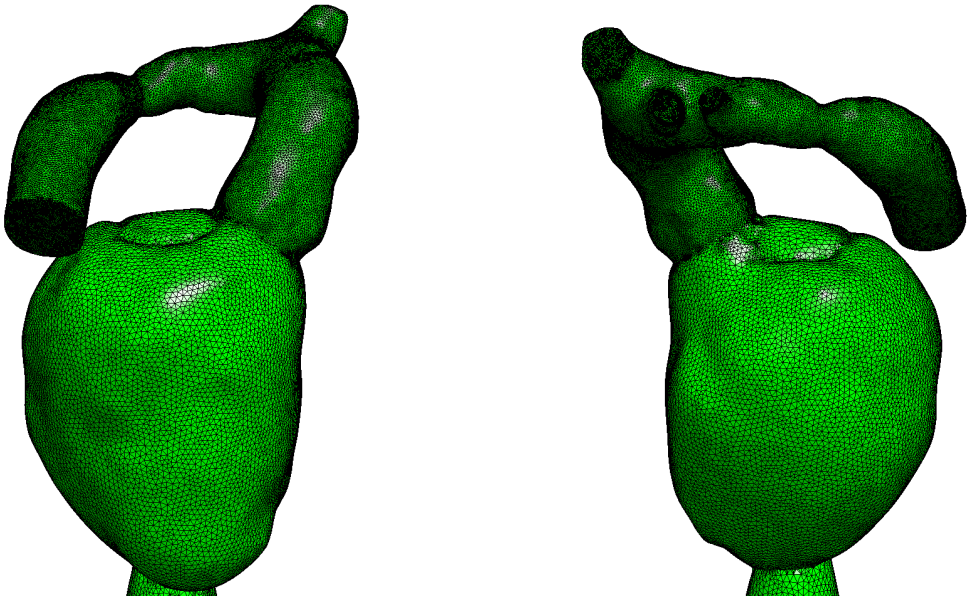


(a) Norm of the velocity $\|u\|$ (b) Pressure p

Figure 6.8: Resulting velocity and pressure for problem D at $t = 1$

Problem E

To conclude this chapter, we will now compute the flow of blood through the human heart depicted in fig. 6.9.



(a) Front (b) Back

Figure 6.9: Fine mesh of a human heart

This mesh has 27636514 elements and 4905775 vertices. We therefore combine the stabilized $\mathcal{P}_1/\mathcal{P}_1$ element pairing introduced in section 4.2 with a block-preconditioner from section 3.3, and a suitable iterative solver (MinRes or Bramble Pasciak CG). Recalling Stokes' equations from section 2.2, we choose the Sobolev space H^1 for

the pressure space Q and $(H^1)^3$ for the velocity space V, in three dimensions.
To reduce the computational complexity, we assemble a smaller, scalar matrix $A_h^{(scal)}$ on the space H^1 and use the algebraic multigrid preconditioner provided by petsc, $P_h^{(scal)}$, for the scalar matrix $A_h^{(scal)}$.
The matrix A_h and a corresponding preconditioner P_h can then be constructed using the scalar matrices $A_h^{(scal)}$ and $P_h^{(scal)}$, as

$$A_h \coloneqq \begin{pmatrix} A_h^{(scal)} & 0 & 0 \\ 0 & A_h^{(scal)} & 0 \\ 0 & 0 & A_h^{(scal)} \end{pmatrix} \quad \text{and} \quad P_h \coloneqq \begin{pmatrix} P_h^{(scal)} & 0 & 0 \\ 0 & P_h^{(scal)} & 0 \\ 0 & 0 & P_h^{(scal)} \end{pmatrix}.$$

Furthermore, we will use a Jacobi preconditioner, $P_h^{(jac)}$, for the mass-matrix on the pressure space Q, as a preconditioner for the Schur-compliment. This results in the Stokes-block

$$D_h \coloneqq \begin{pmatrix} A_h & B_h^T \\ B_h & -C_h \end{pmatrix},$$

and the block-preconditioner

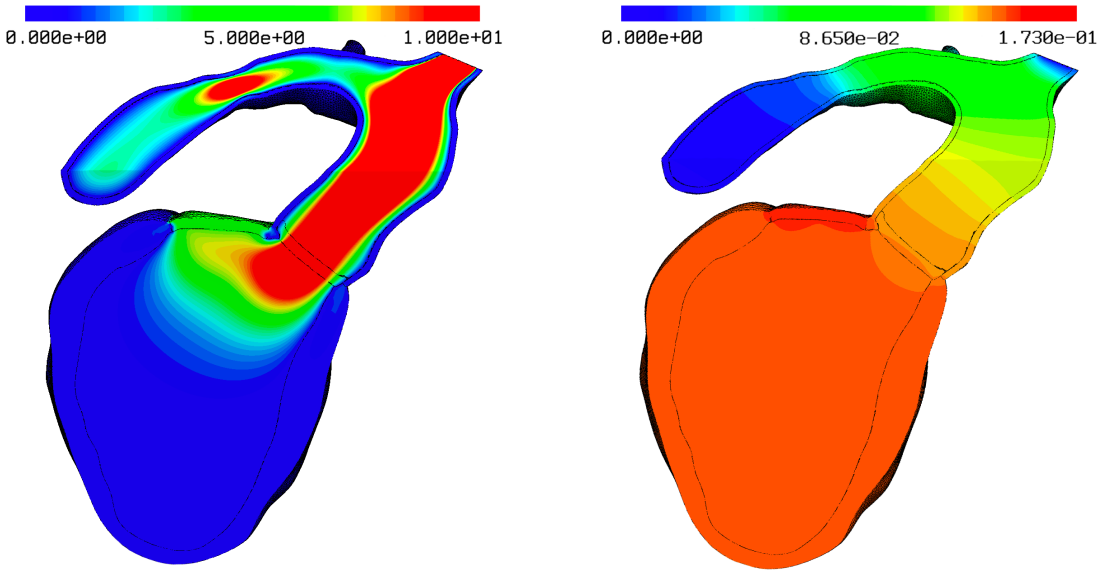
$$K_h \coloneqq \begin{pmatrix} P_h & 0 \\ 0 & P_h^{(jac)} \end{pmatrix}.$$

In addition to these optimization steps, we use MPI parallel computing with 48 CPU cores. This results in approx. 19.6 million degrees of freedom (approx. 600000 mesh elements per engine).
We solve Stokes' equations for the initial values,

$$\begin{aligned} u &= 0 && \text{on} \quad \Gamma_{\mathrm{D}} \\ u_t &= 0 && \text{on} \quad \Gamma_{\mathrm{outlet}} \\ u &= -n && \text{on} \quad \Gamma_{\mathrm{inlet}}, \end{aligned}$$

with outward facing normal vector n.

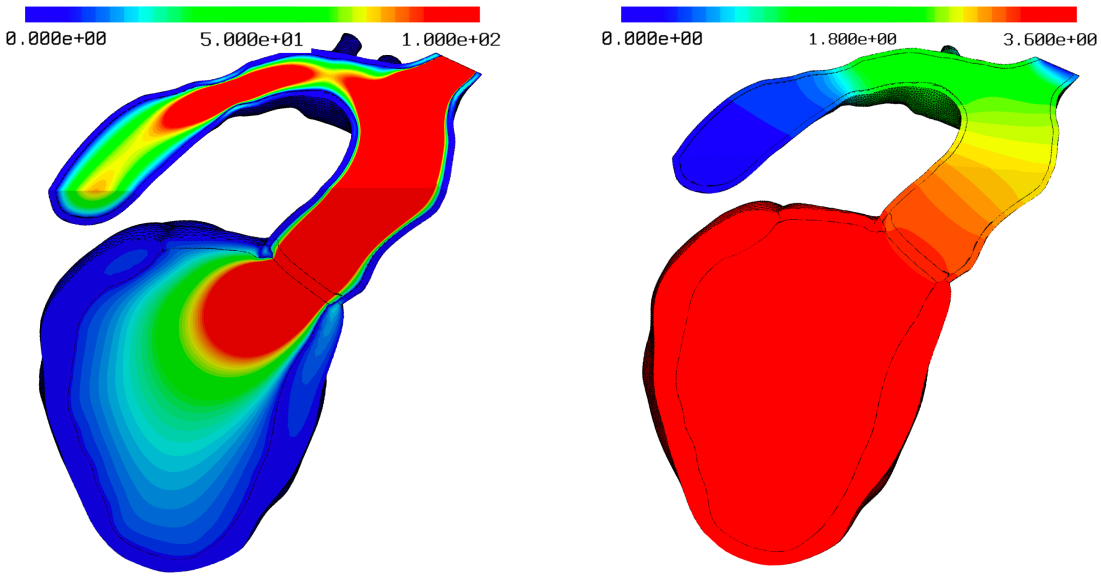
As a first example we choose the bicuspid valve as Γ_{inlet}, and the both the aortic wall and the heart muscle as Γ_{D}. The resulting velocity (fig. 6.10a) and pressure (fig. 6.10b) can be seen in fig. 6.10 below.



(a) Norm of the velocity $\|u\|$ (b) Pressure p

Figure 6.10: Solution to Stokes' equations on the heart domain fig. 6.9

With the contracting movement of a real heart in mind, we will now use the whole heart-muscle as Γ_{inlet}, and the aortic wall and the bicuspid valve as Γ_{D}. This results in the velocity and pressure shown in fig. 6.11.



(a) Norm of the velocity $\|u\|$ (b) Pressure p

Figure 6.11: Another solution to Stokes' equations on the heart domain fig. 6.9

Chapter 7

Conclusion

As expected, using low order $\mathcal{P}_1$ elements for both the pressure- and the velocity-space lowers the computational complexity compared to other methods with higher order spaces or other additional degrees of freedom. This is especially noticeable when working with fine meshes e.g. a large number of dofs, as discussed in chapter 6.
However, the apparent limitation of this approach is the limited accuracy, as only piece-wise linear velocities and pressures can be approximated using $\mathcal{P}_1/\mathcal{P}_1$ element pairings. Additionally, as this velocity/pressure-space combination does not meet the LBB-condition, a stabilization term is required, adding an unpreventable consistency error of order h.

Appendix A

Python implementation

A.1 Schäfer-Turek

```
from ngsolve import *
from ngsolve.webgui import Draw
from netgen.occ import *
from netgen.webgui import Draw as DrawGeo

shape = Rectangle(2,.41).Circle(0.2,0.2,0.05).Reverse().Face()
shape.edges.name="wall"
shape.edges.Min(X).name="inlet"
shape.edges.Max(X).name="outlet"
shape.edges.Nearest(gp_Pnt2d(0.2,0.2)).name="cyl"

DrawGeo (shape)
mesh = Mesh(OCCGeometry(shape, dim=2).GenerateMesh(maxh=0.2))#.
    Curve(3)
Draw (mesh);
print (mesh.GetBoundaries())
print("mesh elements: ", mesh.ne)


def runBenchmark(Um,tend,time,drag,lift,uquer):
    V = VectorH1(mesh,order=1, dirichlet="wall|inlet|cyl")
    Q = H1(mesh,order=1)
    X = V*Q


    e1 = GridFunction(X)
    e2 = GridFunction(X)
    e1.components[0].Set(CoefficientFunction((-2/(uquer**2*0.1),0)
   ), definedon=mesh.Boundaries("cyl"))
```

```
    e2.components[0].Set(CoefficientFunction((0,-2/(uquer**2*0.1))), definedon=mesh.Boundaries("cyl"))

    u,p = X.TrialFunction()
    v,q = X.TestFunction()

    nu = 0.001

    h = specialcf.mesh_size

    stokes = (nu*InnerProduct(grad(u), grad(v))+ \
        div(u)*q+div(v)*p -0.1*h*h*grad(p)*grad(q))*dx # -1e-10*p*q)*dx #

    a = BilinearForm(stokes).Assemble()

    print("ndofs: ", X.ndof)
    # nothing here ...
    f = LinearForm(X).Assemble()

    inv_stokes = a.mat.Inverse(X.FreeDofs())
    tau = 0.001 # timestep

    m = BilinearForm(u*v*dx).Assemble()
    mstar = BilinearForm(X)
    mstar += u*v*dx+tau*stokes
    mstar.Assemble()
    invmstar = mstar.mat.Inverse(X.FreeDofs())

    conv = BilinearForm(X, nonassemble = True)
    conv += (Grad(u) * u) * v * dx
    tmpTime = []
    tmpDrag = []
    tmpLift = []
    # gridfunction for the solution
    gfu = GridFunction(X)
    #http://www.mathematik.tu-dortmund.de/lsiii/cms/papers/SchaeferTurek1996.pdf
    uin = CoefficientFunction( (Um*4*y*(0.41-y)/(0.41*0.41), 0) )
    gfu.components[0].Set(uin, definedon=mesh.Boundaries("inlet"))
    #Draw (Norm(gfu.components[0]), mesh, "velocity", sd=3)
    #Draw (gfu.components[0], mesh, "vel");

    res = f.vec.CreateVector()
    res.data = f.vec - a.mat*gfu.vec
    gfu.vec.data += inv_stokes * res

    #Draw (gfu.components[0], mesh)
```

```
    t = 0; i = 0
    gfut = GridFunction(V, multidim=0)
    vel = gfu.components[0]
    scene = Draw ((gfu.components[0]), mesh)

    with TaskManager():
        while t < tend:
            conv.Apply (gfu.vec, res)
            res.data += a.mat*gfu.vec
            gfu.vec.data -= tau * invmstar * res

            t = t + tau; i = i + 1
            if i%10 == 0: scene.Redraw()
            if i%50 == 0: gfut.AddMultiDimComponent(vel.vec)

            tmpTime.append( t )
            tmpDrag.append(InnerProduct(res, e1.vec) )
            tmpLift.append(InnerProduct(res, e2.vec) )

    time.append(tmpTime)
    drag.append(tmpDrag)
    lift.append(tmpLift)

time = []
drag = []
lift = []
cl = []
ndofs = []

uquer = [0.2,1]
Um = [.3,1.5]
for i, um in enumerate(Um):
    runBenchmark(um,8,time,drag,lift,uquer[i])

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

plt.plot(time[0], drag[0],label="Um=0.3m/s")
plt.plot(time[1], drag[1],label="Um=1.5m/s")

plt.xlabel('time')
plt.ylabel('drag')
plt.title('Drag')
plt.grid(True)
plt.legend()
plt.show()

plt.plot(time[0], lift[0],label="Um=0.3m/s")
```

```
plt.plot(time[1], lift[1],label="Um=1.5m/s")

plt.xlabel('time')
plt.ylabel('lift')
plt.title('Lift')
plt.grid(True)
plt.legend()
plt.show()
```

Listing A.1: Schäfer-Turek benchmark.

```
order=3
VT = HDiv(mesh, order=order, dirichlet="wall|inlet|cyl")
VF = TangentialFacetFESpace(mesh, order=order, dirichlet="wall|
    inlet|cyl")
Q = L2(mesh, order=order-1)
X = VT*VF*Q

u, uhat, p = X.TrialFunction()
v, vhat, q = X.TestFunction()

n = specialcf.normal(mesh.dim)
h = specialcf.mesh_size
dS = dx(element_boundary=True)

nu = 1e-3

def tang(vec):
    return vec - (vec*n)*n

# Thesis Christoph Lehrenfeld, page 71
stokes = nu*InnerProduct(Grad(u), Grad(v)) * dx \
    + nu*InnerProduct(Grad(u)*n, tang(vhat-v)) * dS \
    + nu*InnerProduct(Grad(v)*n, tang(uhat-u)) * dS \
    + nu*4*order*order/h * InnerProduct(tang(vhat-v), tang(uhat-u)
    ) * dS \
    + div(u)*q*dx + div(v)*p*dx -1e-11/nu*p*q*dx

a = BilinearForm (stokes).Assemble()
```

Listing A.2: $H(div)$-block for Schäfer-Turek benchmark.

A.2 Heart (fine mesh in 3D)

```
from ipyparallel import Cluster
c = await Cluster(engines="mpi").start_and_connect(n=46, activate=
    True)
```

```
4
%%px
6 from ngsolve import *
from netgen.occ import *
8 from ngsolve.krylovspace import BramblePasciakCG
from mpi4py.MPI import COMM_WORLD as comm
10 import ngsolve.ngs2petsc as n2p
import petsc4py.PETSc as psc
12

14 %%px
if comm.rank == 0:
16     print(comm.rank, "loading Mesh...")

18 mesh = Mesh("B0305-28.vol.bin", comm)
print (mesh.GetNE(VOL))
20

22 %%px
# bcs
24 fluid_dom = "aortenklappe|mitralklappe|blut_linksventrikel|
    blut_aorta"
heart_dom = "herzmuskel|aortawand"
26 cushion = "cushion"# bnd-conditions
wall = "aortawand-blut_aorta|aortawand-aortenklappe|herzmuskel-
    blut_aorta|herzmuskel-aortenklappe|herzmuskel-mitralklappe|
    herzmuskel-blut_linksventrikel"
28 inlet = "mitralklappe"
outlet = "blut_aorta"
30 do_nothing = "aortenklappe-blut_aorta|mitralklappe-
    blut_linksventrikel|aortenklappe-blut_linksventrikel"


32
%%px
34 V = VectorH1(mesh, order=1, dirichlet=(wall + "|" + inlet + "|" +
    heart_dom))
V1 = H1(mesh, order=1, dirichlet=(wall + "|" + inlet + "|" +
    heart_dom))
36 Q = H1(mesh, order=1)
printmaster ("ndof = ", V.ndofglobal,'+',Q.ndofglobal,'=',
38         V.ndofglobal+Q.ndofglobal)

40 u,v = V.TnT()
u1,v1 = V1.TnT()
42 p,q = Q.TnT()

44 h = specialcf.mesh_size
```

```
bfa1 = BilinearForm(InnerProduct(grad(u1),grad(v1))*dx)
bfb = BilinearForm(div(u)*q*dx).Assemble()
bfc = BilinearForm(h*h*grad(p)*grad(q)*dx).Assemble()

prea1 = Preconditioner(bfa1, "gamg") # AMG precond from PETSc
bfa1.Assemble()

# make block-diagonal A matrix:
mata = sum( [Ri.T@bfa1.mat@Ri for Ri in V.restrictions] )
prea = sum( [Ei@prea1@Ei.T for Ei in V.embeddings])

bfschur = BilinearForm(p*q*dx, diagonal=True).Assemble()
preschur = bfschur.mat.Inverse()

%%px

gfu = GridFunction(V)
gfp = GridFunction(Q)

n = specialcf.normal(3)

gfu.Set(5*n, definedon=mesh.Boundaries(inlet))

resf = (-mata * gfu.vec).Evaluate()
resg = (-bfb.mat * gfu.vec).Evaluate()

sol = BramblePasciakCG (A=mata, B=bfb.mat, C=bfc.mat, f=resf, g=
    resg, \
                preA=prea, preS=preschur, maxit=500,
                printrates='\r' if comm.rank==0 else False)

gfu.vec.data += sol[0]
gfp.vec.data += sol[1]


gfu = c[:]["gfu"]


import pickle
print ("start pickling")
pickle.dump(gfu, open("heart-p48-it1000", "wb"))
```

Listing A.3: Code used for problem E

```
from ngsolve import *
import netgen.meshing
import pickle

with TaskManager (pajetrace=10**8):
```

```
    gfu = pickle.load (open("heart-p48-it1000.pickle", "rb"))

    #Draw ((gfu[0]))#, sd=0)
    Draw (Norm(gfu[0]), gfu[0].space.mesh, "|u|")#, order=1)
    Draw (gfu[0], gfu[0].space.mesh, "u")#, order=1)
    #Draw (gfu[1], gfu[1].space.mesh, "p")#, order=1)

input("Any key to continue..")
```

Listing A.4: Code to display pickled solution

Bibliography

[19] *Netgen/NGSolve*. 2019. URL: https://ngsolve.org/ (visited on 12/18/2021).

[ARS97] Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. "Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations". In: *Applied Numerical Mathematics* 25.2 (1997). Special Issue on Time Integration, pp. 151–167. DOI: https://doi.org/10.1016/S0168-9274(97)00056-1. URL: https://www.sciencedirect.com/science/article/pii/S0168927497000561.

[BE86] John Barrett and Charles Elliott. "Finite element approximation of the Dirichlet problem using the boundary penalty method". In: *Numerische Mathematik* 49 (07/1986), pp. 343–366. DOI: 10.1007/BF01389536.

[BF91] Franco Brezzi and Michel Fortin. *Mixed and Hybrid Finite Element Methods*. eng. Softcover reprint of the hardcover 1st edition 1991. Springer-Verlag New YorK Inc., 1991. 361 pp. DOI: 10.1007/978-1-4612-3172-1.

[BP84] F. Brezzi and J. Pitkäranta. "On the Stabilization of Finite Element Approximations of the Stokes Equations". In: *Efficient Solutions of Elliptic Systems: Proceedings of a GAMM-Seminar Kiel, January 27 to 29, 1984*. Ed. by Wolfgang Hackbusch. Wiesbaden: Vieweg+Teubner Verlag, 1984, pp. 11–19. DOI: 10.1007/978-3-663-14169-3_2. URL: https://doi.org/10.1007/978-3-663-14169-3_2.

[Bra92] Dietrich Braess. *Finite Elemente*. ger. 5. Auflage. Berlin Heidelberg: Springer Spektrum, 1992.

[Bre74] F. Brezzi. "On the existence, uniqueness and approximation of saddle-point problems arising from lagrangian multipliers". In: *R.A.I.R.O. Analyse Numérique* 8.R2 (1974), pp. 129–151. DOI: https://doi.org/10.1051/m2an/197408R201291. URL: https://www.esaim-m2an.org/articles/m2an/abs/1974/01/m2an197408R201291/m2an197408R201291.html.

[Che20] Long Chen. *FINITE ELEMENT METHODS FOR STOKES EQUATIONS*. 2020. URL: https://www.math.uci.edu/~chenlong/226/FEMStokes.pdf (visited on 06/21/2021).

[Chi08] Stephen Childress. *An Introduction to Theoretical Fluid Dynamics*. 2008. URL: https://www.math.nyu.edu/~childres/fluidsbook.pdf (visited on 06/21/2021).

[Leh10] Christoph Lehrenfeld. "Hybrid Discontinuous Galerkin Methods for Incompressible Flow Problems". MA thesis. RWTH Aachen, 05/2010. DOI: 10.25625/O4VBYH. URL: https://data.goettingen-research-online.de/file.xhtml?persistentId=doi:10.25625/O4VBYH/LNLFCJ&version=1.0.

[LS16] Christoph Lehrenfeld and Joachim Schoeberl. "High order exactly divergence-free Hybrid Discontinuous Galerkin Methods for unsteady incompressible flows". In: *Computer Methods in Applied Mechanics and Engineering* 307 (05/2016). DOI: 10.1016/j.cma.2016.04.025.

[Neu17] Michael Neunteufel. "Advanced numerical methods for fluid structure interaction". MA thesis. Vienna University of Technology, 2017. 99 pp. DOI: 10.34726/hss.2017.47691. URL: https://doi.org/10.34726/hss.2017.47691.

[Sch+96] Michael Schäfer, Stefan Turek, Franz Durst, Egon Krause, and Rolf Rannacher. "Benchmark Computations of Laminar Flow Around a Cylinder". In: 1996.

[Sch97] Joachim Schöberl. "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules". In: *Computing and Visualization in Science* 1.1 (07/1997), pp. 41–52. DOI: 10.1007/s007910050004. URL: https://doi.org/10.1007/s007910050004.

[Sch99] Joachim Schöberl. *Robust Multigrid Methods for Parameter Dependent Problems*. 1999. URL: https://www.asc.tuwien.ac.at/~schoeberl/wiki/publications/diss.pdf (visited on 12/18/2021).

[TH73] C. Taylor and P. Hood. "A numerical solution of the Navier-Stokes equations using the finite element technique". In: *Computers & Fluids* 1.1 (1973), pp. 73–100. DOI: https://doi.org/10.1016/0045-7930(73)90027-3.

[UC82] M. Utku and G.F Carey. "Boundary penalty techniques". In: *Computer Methods in Applied Mechanics and Engineering* 30.1 (1982), pp. 103–118. DOI: https://doi.org/10.1016/0045-7825(82)90057-3. URL: https://www.sciencedirect.com/science/article/pii/0045782582900573.