Software and Numerical Methods for the Incompressible Navier-Stokes Equations

Kent-Andre Mardal

May 7, 2003

Acknowledgements

This is my thesis for the degree Doctor Scientiarum at the Department of Informatics, University of Oslo. The work has been conducted in the period from May 2000 to May 2003 and was financed by the BeMatA program of the Research Council of Norway. I would like to thank the Simula Research Laboratory and the Department of Informatics for excellent working conditions and financial support.

I want to thank my two supervisors, Professor Hans Petter Langtangen and Professor Aslak Tveito for an interesting project, the time and effort they have invested, and their visions. Hans Petter has shaped much of my view and appreciation of software development. This thesis would not have been possible without Diffpack. Professor Ragnar Winther has also been an important person with his knowledge on multigrid and finite element methods. Many thanks for several good discussions, pool games, etc.: Ola Skavhaug, Trygve Kastberg Nilssen, Joakim Sundnes, Ariel Almendral, Glen Terje Lines, Audrey Huerta, Tom Thorvalsen and the others from Scientific Computing.

My friends and family, mum and dad, have always been supportive. I will also like to thank the best things that have ever happened to me: Nancy, Natalie and Niklas.

Oslo, April 2003 *Kent-Andre Mardal*

Contents

This thesis contains five individual papers and a short introduction.

- **Paper I** *Numerical Methods for Incompressible Viscous Flow*, Hans Petter Langtangen, Kent-Andre Mardal and Ragnar Winther. Selected review paper for the 25th Anniversary Issue of the journal *Advances in Water Resources*, vol 25, 1125-1146, 2002.
- **Paper II** *Mixed Finite Elements*, Kent-Andre Mardal and Hans Petter Langtangen. In Langtangen and Tveito (eds): *Advanced Topics in Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*, Lecture Notes in Computational Science and Engineering, Springer, 2003.
- **Paper III** *A Robust Finite Element Method for Darcy-Stokes Flow*, Kent-Andre Mardal, Xue-Cheng Tai, and Ragnar Winther. *SIAM Journal on Numerical Analysis*, vol 40, 1605-1631, 2002.
- **Paper IV** *Systems of PDEs and Block Preconditionering*, Kent-Andre Mardal, Joakim Sundnes, Hans Petter Langtangen and Aslak Tveito. In Langtangen and Tveito (eds): *Advanced Topics in Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*, Lecture Notes in Computational Science and Engineering, Springer, 2003.
- **Paper V** *Uniform Preconditioners for the Time Dependent Stokes Problem*, Kent-Andre Mardal and Ragnar Winther. Submitted to *Numerische Mathematik*.

The five papers are self-contained. Therefore, some information is repeated. There are also some differences in the notation used in the papers.

Introduction

This thesis is concerned with numerical methods for partial differential equations (PDEs) and their realization in software. The focus is on the incompressible Navier-Stokes equations and both the mathematics and the implementation are covered.

This introduction is organized as follows. First, some general remarks on the modeling of our physical reality by PDEs are made. In particular, we try to motivate the study of the Navier-Stokes equations and associated numerical methods. Then, we give a short outline of the papers included in this thesis. Paper I is a review of numerical methods for the incompressible Navier-Stokes equations and serves as an introduction to and overview of the topic. This introduction is therefore rather short.

Many of the laws in nature can be expressed by PDEs. We have the diffusion equation in thermal physics, the wave equation in, e.g., acoustics, the equation of elasticity in solid mechanics, the Schrödinger equation in quantum mechanics, the Maxwell's equations in electromagnetism, the Einstein's equation in general relativity and the Navier-Stokes equations in fluid dynamics. Solutions by pencil and paper are normally only feasible in very simplified applications. Instead, approximations are computed with numerical methods on computers. The efficiency and accuracy of a given numerical method vary among the equations and it is often necessary to design the method for the purpose. In some cases it is possible to buy or download reliable software. Diffpack is one example of software for solving PDEs [11] and it is used extensively in this thesis. One goal with this thesis is to extend Diffpack with state-of-the-art discretization techniques and solution algorithms for computational fluid dynamics (CFD).

In this thesis we will mostly be concerned with the Navier-Stokes equations, describing the motion of fluids, gases and in some cases solids. These equations are of fundamental importance in science and engineering. For instance, solving these equations is necessary to find an optimal shape design of aircrafts, cars and ships. Another application is the simulation of heating systems. Around 25% of the total production of electrical energy in Norway is used for heating. Alternative or more efficient heating of houses may therefore save a lot of money. In [10] we describe a simulator for certain low temperature heating systems. Another application appears in medicine, where the computation of blood flow in heart and body may give a better understanding of arterial diseases [21]. Also, solids like metal and rock can be described by the Navier-Stokes equations.

Some applications are aluminum extrusion [20] and the evolution of mountain belts like Himalaya as a result of tectonic plate drifting [6].

Needless to say, the examples are different in many aspects, e.g., the viscosity parameter μ vary from $\sim 10^{-5}\frac{\text{kg}}{\text{ms}}$ in air to $\sim 10^{20}\frac{\text{kg}}{\text{ms}}$ in rock. Many of the applications need additional features like turbulence models, coupling to a heat equation, moving geometries, free surfaces, or nonlinear viscosities. Nevertheless, a fast and accurate solver for the Navier-Stokes equations is needed as a core in the numerical engine for all these problems.

We will mostly be concerned with the incompressible Navier-Stokes equations. All materials are compressible, but incompressibillity is often a good approximation. Even air flow is incompressible up to 1/3 of the speed of sound. Many authors use the Navier-Stokes equations as a short term for the incompressible Navier-Stokes equations and we will adopt to this habit. The equations are on the form:

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v} = -\frac{1}{\varrho}\nabla p + \mu\nabla^2\mathbf{v} + \mathbf{g}, \quad \text{in } \Omega, \tag{1}$$

$$\nabla\cdot\mathbf{v} = 0, \quad \text{in } \Omega, \tag{2}$$

where $\mathbf{v}$ and p are the unknown velocity and pressure, respectively, μ is the viscosity, ϱ is the density, and $\mathbf{g}$ denotes the body forces. Additionally, boundary and initial conditions are needed. Equation (1) comes from Newton's second law of motion, while equation (2) states that the fluid is incompressible.

These equations are hard to understand, although they have been studied extensively for at least a century. In fact, even proving the existence of a solution to these equations (in a certain sense) has been ranked by the Clay Mathematics Institute of Cambridge as one of the seven hardest problems in this millennium. A solution of this problem gives a \$1 million price [5].

Despite of the theoretical difficulties, these equations are routinely solved numerically by today's scientists and engineers. The advantage and danger with numerical methods are that they will "always" compute an answer. The question is how accurate and efficient the computations are. A variety of methods and software packages exists, each with its strengths and weaknesses. The problem is that different methods compute significantly different answers. Additionally, it is often not known what the answer should be. A quote given in Turek [25] describes what practitioners from automotive industry (unofficially) say:

> "There is no software available which can provide a guaranteed lift and drag coefficient on a car-body with an error tolerance of less than 20%; often the sign of the lift cannot even be predicted. Hence, we stopped flow around objects and use simulation tools for interior flow problems only, for instance for modeling heating devices or acoustic behavior in car cabins. Here, we are content with a qualitatively good prediction!"

It should be noted that this quote is valid even for flow with low Reynolds numbers. In fact, Schäfer et al. [23] conducted a benchmark for flow around

a cylinder with Reynolds number 20 and 100. They compared the simulation results from various implementations of discretization techniques and solution methods. The solutions computed by 17 research groups differed by 20% in the computation of the lift coefficient. This problem is "simple", compared to applications in industry. Still, it is not satisfactory solved, at least not with all the methods.

There is no simple answer to why common methods computed different answers in the benchmark. Many discretization techniques were used in space and time, with both operator splitting and fully coupled solvers. Some of the techniques have a more sound mathematical foundation than others. Mixed finite elements, with streamwise upwinding or Petrov-Galerkin stabilization for high Reynolds numbers, are key components in the mathematical understanding of robust discretizations[1]. However, it is far from the Babuska-Brezzi condition and the Stokes problem, for which most of the theory for mixed elements has been developed, to the computation of the lift coefficient in flow around a cylinder using a limited amount of computer resources. This general uncertainty, summarized by the quote from Turek [25], is our motivation behind making the CFD tools in Diffpack flexible, such that different methods can be tested and compared. We try to have particular focus on the robustness of the methods.

A part of the explanation of the differences in the computation of the lift coefficient could be that the resolution of the meshes was not sufficiently fine. All, or at least most, of the methods should compute the proper lift coefficient, given enough unknowns. Today, we may solve linear systems with $10^6 - 10^7$ unknowns on a normal workstation. Given an efficient solution algorithm, such a system may be solved in less than a minute. Multigrid is a very promising and general strategy for solving such systems and is often an (order) optimal algorithm[2]. Although efficient multigrid methods for the Navier-Stokes equations have been implemented in many cases, it is still a problem to construct such solvers in general. This remains a hot topic, even though efficient algorithms in special cases have been demonstrated almost since multigrid was developed in the 1960-1970s. As a general rule of the thumb, multigrid is more efficient than the classical Conjugate-Gradient like methods (or Krylov methods) with standard algebraic preconditioners [4], if the number of unknowns is larger than 10^4 [16, 25]. In the benchmark [23], the number of unknowns was $10^4 - 10^7$ and the results clearly favored multigrid techniques. Moreover, in the nonstationary 3D case, they were not able to compute reliable reference solutions. More unknowns or better methods are needed. Applications in industry require accurate and efficient computations of far more advanced CFD problems. Hence, it is clear that efficient methods like multigrid will play a critical role in the future. Also, it must be combined properly with adaptivity, domain decomposition, etc.

[1] This does not mean that mixed elements are needed. Pressure stabilization techniques satisfy a slightly more complicated Babuska-Brezzi condition and the penalty method uses reduced integration to mimic mixed elements. More details can be found in Paper I and the references therein.

[2] An optimal algorithm solves the problem in $\mathcal{O}(n)$ operations, where n is the number of unknowns. A more detailed discussion of optimal algorithms can be found in Paper IV.

Multigrid methods for CFD can be found in, e.g., Turek [25]. An overview of multigrid methods in general can be found in Brandt [2]. Iterative methods are dealt with in [9].

The focus in this thesis is finite element methods and iterative solution techniques for the Navier-Stokes equations, and simplified versions of these equations. Particular emphasis is placed on the robustness of the methods and on implementational issues. Many aspects related to the Navier-Stokes equations are not covered, such as adaptivity, flow with high Reynolds numbers, streamwise upwinding/Petrov-Galerkin methods, turbulence models, parallelization, boundary conditions, etc.

Summary of the Papers

Paper I: Numerical Methods for Incompressible Viscous Flow. The purpose of this paper is to give an overview of numerical methods for the incompressible Navier-Stokes equations. This is a vast field and the paper is by no means complete. The focus is to show the relations between seemingly different methods. First, we cover mixed element methods and their relations with common stabilization techniques such as pressure stabilization, the penalty method and artificial compressibility. Then, various operator splitting techniques, both algebraic and in time, are discussed. The framework called the basic iteration on the pressure Schur complement [25] shows the relations between different methods such as projection methods, pressure correction schemes, the Uzawa algorithm, and the Vanka smoother. Finally, the connection between operator splitting and block preconditioning is presented. A more comprehensive review of methods can be found in the 1200 pages textbook [8].

This paper is an overview and not much new is described. However, we extend the framework in [25] such that it includes the block preconditioning in, e.g., Paper IV, V, and [22].

The efficiency of the block preconditioner has only been tested in the academic examples in Paper V. The next step is the fully nonlinear Navier-Stokes equations. The block preconditioning should also be compared with the basic iteration on the Schur complement, which is easy to implement in the framework described in Paper IV. Moreover, standard operator splitting schemes such as the projection method can be reused as preconditioners by using the auxiliary space method as in Paper IV and Paper V. Finally, it would be interesting to implement the problem in the benchmark [23].

Paper II: Mixed Finite Elements. This paper describes the basic mathematical concepts of mixed finite element methods and the usage of such methods in the Diffpack programming environment. The following two model problems are considered: the Stokes problem,

$$-\mu\Delta\mathbf{v} + \nabla p = \mathbf{g}, \tag{3}$$

$$\nabla \cdot \mathbf{v} = 0, \tag{4}$$

and the mixed formulation of the Poisson equation,

$$\mathbf{v} - \lambda\nabla p = 0, \tag{5}$$

$$\nabla \cdot \mathbf{v} = g. \tag{6}$$

Several elements have been implemented: Crouzeix-Raviart, Mini, $P_2 - P_0$, Rannacher-Turek, Raviart-Thomas and Taylor-Hood (see, e.g., [3, 7, 25] for mathematical descriptions). The new element in Paper III has also been implemented. These elements can be used on general unstructured grids, within the framework of multigrid methods.

Most of the applications of the elements in this thesis have been academic, in the sense that simple geometries are used. Further work can be to compare the elements with standard methods in real-life applications. It would, for instance, be interesting to check whether mixed elements actually improve the accuracy compared to standard finite elements, with a given amount of computer resources. We are also currently extending Diffpack with the Nedelec element [19] which is popular in electro-magnetism.

Paper III: A Robust Finite Element Method for Darcy-Stokes Flow. This paper introduces a new element, applicable for Darcy-Stokes flow, and compares it with several other mixed elements. The equations for Darcy-Stokes flow read

$$\mathbf{v} - \epsilon^2 \Delta \mathbf{v} + \nabla p = \mathbf{g}, \tag{7}$$

$$\nabla \cdot \mathbf{v} = 0, \tag{8}$$

where ϵ is a physical parameter in $[0, 1]$. For ϵ close to 1, this problem is similar to the Stokes problem (3)-(4), but with an lower order term, $\mathbf{v}$. On the other hand, if $\epsilon \to 0$ the problem reduces to the mixed formulation of the Poisson equation (5)-(6). If $\epsilon > 0$, standard Stokes elements can be used. However, the numerical experiments in this paper show that the accuracy decreases as ϵ decreases. The new element is robust with respect to ϵ. Some implications for time dependent flow are described in [15].

The element made here is new. There are also some new experiments showing that common Stokes elements handle low viscosity (e.g., 10^{-6}) differently. Mixed elements with continuous pressure seem more robust, although not as robust as the new element.

A natural extension of this work is to let ϵ vary spatially, such that the Darcy character of the flow dominates in some parts of the domain, while Stokes flow governs the rest. It is not clear that the element handles this situation.

Paper IV: Systems of PDEs and block preconditioning. This paper describes the block preconditioning tools in Diffpack. Block preconditioning is a simple, but general, idea. Let us assume that we have the system of equations,

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}. \tag{9}$$

A diagonal block preconditioner is then on the form,

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{N} \end{bmatrix}. \tag{10}$$

An important point is that if the matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ come form the discretization of PDEs, we might know how to construct preconditioners for $\mathbf{A}$ and $\mathbf{D}$, while a preconditioner for the fully coupled system is not known. If $\mathbf{A}$ and $\mathbf{D}$ "dominate" the matrix system, it is reasonable to assume that they "dominate" the preconditioner. We would then make $\mathbf{M} \sim \mathbf{A}^{-1}$ and $\mathbf{N} \sim \mathbf{D}^{-1}$. A more rigorous mathematical theory for block preconditioning can be found in [1].

The software described here is new, and together with the multigrid software in [16], several fast preconditioners have been implemented. Optimal preconditioners are implemented for the Stokes problem (3)-(4) and the mixed Poisson equation (5)-(6). The preconditioner for the mixed Poisson problem uses the auxiliary space technique of Xu [26], which enables multigrid methods on standard elements to be reused for mixed elements[3]. Additionally, an optimal block preconditioner for a simplified version of the equations for the electrical activity in the heart [24] is described. These equations read,

$$\begin{aligned} \frac{\partial v}{\partial t} &= \nabla \cdot (\sigma_i \nabla v) + \nabla \cdot (\sigma_i \nabla u_e), \\ 0 &= \nabla \cdot (\sigma_i \nabla v) + \nabla \cdot ((\sigma_i + \sigma_e) \nabla u_e), \end{aligned}$$

where v is the transmembrane potential, u_e is the extracellular potential, and σ_i and σ_e are the intra- and extracellular conductivities, respectively.

Future work here is an implementation of the framework described in Paper I. We are also currently working on extending the block preconditioner to thermally driven flow. The optimality of the preconditioner in the case of the equations for the electrical activity in the heart and the mixed Poisson equations should also be proven theoretically.

Paper V: Uniform Preconditioners for the Time Dependent Stokes Problem. In this paper we study preconditioners for the system (7)-(8). However, the ϵ^2 is not a physical parameter, it equals the time stepping parameter, Δt. The optimal preconditioner, robust with respect to Δt, is on the form,

$$\begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} + \Delta t \mathbf{N} \end{bmatrix}. \tag{11}$$

The preconditioners $\mathbf{L}$ and $\mathbf{M}$ are made as multigrid sweeps on second order elliptic equations and $\mathbf{N}$ may be a lumped mass matrix. Hence, the building blocks are standard preconditioners. This preconditioner resembles the preconditioner made for the iteration on the pressure Schur complement in [25], although this was discovered later. The preconditioner can also be (and was originally) motivated by the numerical error estimates with the Mini element in Paper III.

The uniformity of the preconditioner in Δt is new. There are also some new Δt-uniform inf-sup conditions.

[3] Some properties must be satisfied.

In future work we would like to extend this preconditioner to the fully non-linear and non-symmetric Navier-Stokes equations as described in Paper I. It would also be interesting to check whether a similar preconditioner can be made for the new element in Paper III. The numerical results in Paper IV suggest that this might be possible.

During the work of this thesis some more papers were written. The paper [16] is about the multigrid software in Diffpack, which was used in Paper IV, V and [24]. The paper [24] is about the block preconditioner for the heart problem described in Paper IV. The paper [13] is about making Python interfaces to Diffpack programs. The motivation behind using Python is to ease the debugging and verification of complex C++ programs such as a Diffpack simulator. Several of the results in this thesis and related results have been presented as talks and proceedings at conferences [10, 12, 14, 15, 17, 18] .

Bibliography

[1] D. N. Arnold, R. S. Falk, and R. Winther. Preconditioning discrete approximation of the Reissner Mindlin plate problem. *Matematical Modeling and Numerical Analysis*, 1996.

[2] A. Brandt. Multiscale scientific computation review 2001. In T. J. Barth, T. F. Chan, and R. Haimes, editors, *Multiscale and Multiresolution Methods: Theory and Applications*. Springer, 2001.

[3] F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Methods*. Springer-Verlag, 1991.

[4] A. M. Bruaset. *A Survey of Preconditioned Iterative Methods*. Addison-Wesley Pitman, 1995.

[5] C. L. Fefferman. Existence & smoothness of the Navier-Stokes equations. See URL http://www.claymath.org/Millennium_Prize_Problems/.

[6] P. Fullsack. An arbitrary Lagrangian-Eulerian formulation for creeping flows and its application in tectonic models. *Geophys. J. Int.*, 120:1–23, 1995.

[7] V. Girault and P. A. Raviart. *Finite Element Methods for Navier-Stokes Equations*. Springer-Verlag, 1986.

[8] P. M. Gresho and R. L. Sani. *Incompressible Flow and the Finite Element Method*. Wiley, 1998.

[9] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer-Verlag, 1994.

[10] A. Kristoffersen and K.-A. Mardal. Solving heat distribution in room using mixed finite element for coupling the Navier-Stokes equations and the heat equation. In H. I. Andersson and B. Skallerud, editors, *Second National Conference on Computational Mechanics (MekIT'03)*, 2003.

[11] H. P. Langtangen. *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*. Textbooks in Computational Science and Engineering. Springer-Verlag, 2nd edition, 2003.

[12] H. P. Langtangen and K.-A. Mardal. A software framework for mixed finite element programming. In P. M. A. Sloot, C. J. K. Tan, J. J. Dongarra, and A. G. Hoekstra, editors, *Proceedings of the 2nd International Conference on Computation al Science*, Lecture Notes in Computer Science. Springer, 2002.

[13] H. P. Langtangen and K.-A. Mardal. Using Diffpack from Python scripts. In H. P. Langtangen and A. Tveito, editors, *Advanced Topics in Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*. Springer-Verlag, 2003.

[14] K.-A. Mardal and H. P. Langtangen. An efficient parallel iterative approach to a fully implicit mixed finite element formulation for the Navier-Stokes equations. In *Computational Fluid Dynamics Conference Proceedings (ECCOMAS CFD 2001)*, 2001.

[15] K.-A. Mardal, H. P. Langtangen, and R. Winther. Error estimates for the linear Navier-Stokes equations. In H. I. Andersson and B. Skallerud, editors, *Second National Conference on Computational Mechanics (MekIT'03)*, 2003.

[16] K.-A. Mardal, H. P. Langtangen, and G.W. Zumbusch. Software tools for multigrid methods. In H. P. Langtangen and A. Tveito, editors, *Advanced Topics in Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*. Springer-Verlag, 2003.

[17] K.-A. Mardal and H.P. Langtangen. An effective iterative approach to a fully implicit mixed finite element formulation for the Navier-Stokes equations. In Harald Osnes Jostein Hellesland and Geir Skeie, editors, *Proceedings of the 13th Nordic Seminar on Computational Mechanics*, pages 89–93, 2000.

[18] K.-A. Mardal, R. Winther, and H.P Langtangen. An optimal iterative approach to the time-dependent Stokes problem. In *Tenth Conference Virtual Proceedings, Copper Mountain Conferences on Multigrid Methods, Copper Mountain, CO, USA, 2001.* URL: http://www.mgnet.org/mgnet-cm2001.html.

[19] J. C. Nedelec. Mixed finite elements in R^3. *Numerische Mathematik*, 35:315–341, 1980.

[20] K. M. Okstad and T. Kvamsdal. An object-oriented finite element program for simulation of aluminium extrusion. In H. P. Langtangen and A. Tveito, editors, *Advanced Topics in Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*. Springer-Verlag, 2003.

[21] A. Quarteroni, M. Tuveri, and A. Veneziani. Computational vascular fluid dynamics: Problems, models and methods. *Computing and Visualisation in Science*, 2:163–197, 2000.

[22] T. Rusten and R. Winther. A preconditioned iterative method for saddle-point problems. *SIAM J. Matrix Anal.*, 1992.

[23] M. Schäfer and S. Turek. Benchmark compuations of laminar flow around cylinder. In E.H. Hirschel, editor, *Flow Simulation with High-Performance Computers II.* Vieweg, 1996.

[24] J. Sundnes, G. Lines, K.-A. Mardal, and A. Tveito. Multigrid block preconditioning for a coupled system of partial differential equations modeling the electrical activity in the heart. *Computer Methods in Biomechanics and Biomedical Engineering*, 5:397–411, 2002.

[25] S. Turek. *Efficient Solvers for Incompressible Flow Problems.* Springer-Verlag, 1999.

[26] J. Xu. The auxiliary space method and optimal multigrid preconditioning techniques for unstructured grids. *Computing*, 56:215–235, 1996.

I

Numerical Methods for Incompressible Viscous Flow

H. P. Langtangen, K.-A. Mardal and R. Winther

Selected review paper for the 25th Anniversary Issue of the journal
Advances in Water Resources, vol 25, 1125-1146, 2002.

Numerical Methods for Incompressible Viscous Flow

Hans Petter Langtangen* Kent-Andre Mardal
Dept. of Scientific Computing, Simula Research Laboratory and
Dept. of Informatics, University of Oslo

Ragnar Winther
Dept. of Informatics, University of Oslo and
Dept. of Mathematics, University of Oslo

May 5, 2003

Abstract

We present an overview of the most common numerical solution strategies for the incompressible Navier–Stokes equations, including fully implicit formulations, artificial compressibility methods, penalty formulations, and operator splitting methods (pressure/velocity correction, projection methods). A unified framework that explains popular operator splitting methods as special cases of a fully implicit approach is also presented and can be used for constructing new and improved solution strategies. The exposition is mostly neutral to the spatial discretization technique, but we cover the need for staggered grids or mixed finite elements and outline some alternative stabilization techniques that allow using standard grids. Emphasis is put on showing the close relationship between (seemingly) different and competing solution approaches for incompressible viscous flow.

1 Introduction

Incompressible viscous flow phenomena arise in numerous disciplines in science and engineering. The simplest viscous flow problems involve just one fluid in the laminar regime. The governing equations consist in this case of the incompressible Navier–Stokes equations,

$$\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v}\cdot\nabla\boldsymbol{v} = -\frac{1}{\varrho}\nabla p + \nu\nabla^2\boldsymbol{v} + \boldsymbol{g}, \tag{1}$$

*Corresponding author. Email: hpl@ifi.uio.no.

and the equation of continuity, also called the incompressibility constraint,

$$\nabla \cdot \boldsymbol{v} = 0\,. \tag{2}$$

In these equations, $\boldsymbol{v}$ is the velocity field, p is the pressure, ϱ is the fluid density, $\boldsymbol{g}$ denotes body forces (such as gravity, centrifugal and Coriolis forces), ν is the kinematic viscosity of the fluid, and t denotes time. The initial conditions consist of prescribing $\boldsymbol{v}$, whereas the boundary conditions can be of several types: (i) prescribed velocity components, (ii) vanishing normal derivatives of velocity components, or (iii) prescribed stress vector components. The pressure is only determined up to a constant, but can be uniquely determined by prescribing the value (as a time series) at one spatial point. Many people refer to the system (1)–(2) as the Navier–Stokes equations. The authors will also adapt to this habit in the present paper.

Most flows in nature and technological devices are turbulent. The transition from laminar to turbulent flow is governed by the Reynolds number, $\mathrm{Re} = Ud/\nu$, where U is a characteristic velocity of the flow and d is a characteristic length of the involved geometries. The basic Navier-Stokes equations describe both laminar and turbulent flow, but the spatial resolution required to resolve the small (and important) scales in turbulent flow makes direct solution of the Navier-Stokes equations too computationally demanding on today's computers. As an alternative, one can derive equations for the average flow and parameterize the effects of turbulence. Such common models models for turbulent flow normally consist of two parts: one part modeling the average flow, and these equations are very similar to (1)–(2), and one part modeling the turbulent fluctuations. These two parts can at each time level be solved sequentially or in a fully coupled fashion. In the former case, one needs methods and software for the system (1)–(2) also in turbulent flow applications. Even in the fully coupled case the basic ideas regarding discretization of (1)–(2) are reused. We also mention that simulation of turbulence by solving the basic Navier-Stokes equations on very fine grids, referred to as Direct Numerical Simulation (DNS), achieves increasing importance in turbulence research as these solutions provide reference databases for fitting parameterized models.

In more complex physical flow phenomena, laminar or turbulent viscous flow is coupled with other processes, such as heat transfer, transport of pollution, and deformation of structures. Multi-phase/multi-component fluid flow models often involve equations of the type (1)–(2) for the total flow coupled with advection-diffusion-type equations for the concentrations of each phase or component. Many numerical strategies for complicated flow problems employ a splitting of the compound model, resulting in the need to solve (1)–(2) as one subset of equations in a possibly larger model involving lots of partial differential equations. Hence, it is evident that complex physical flow phenomena also demand software for solving (1)–(2) in a robust fashion.

Viscous flow models have important applications within the area of water resources. The common Darcy-type models for porous media flow are based on averaging viscous flow in a network of pores. However, the averaging introduces the permeability parameter, which must be measured experimentally,

often with significant uncertainty. For multi-phase flow the ad hoc extensions of the permeability concept to relative permeabilities is insufficient for satisfactory modeling of many flow phenomena. Moreover, the extensions of Darcy's law to flow in fractured or highly porous media introduce considerable modeling uncertainty. A more fundamental approach to porous media flow is to simulate the viscous flow at the pore scale, in a series of network realizations, and compute the relation between the flow rate and the pressure differences. This is an important way to gain more insight into deriving better averaged flow models for practical use and to better understand the permeability concept [6, 80]. The approach makes a demand for solving (1)–(2) in highly complex geometries, but the left-hand side of (1) can be neglected because of small Reynolds numbers (small characteristic length).

Water resources research and engineering are also concerned with free surface flow and currents in rivers, lakes, and the ocean. The commonly used models in these areas are based on averaging procedures in the vertical direction and ad hoc incorporation of viscous and turbulent effects. The shortcomings of averaged equations and primitive viscosity models are obvious in very shallow water, and in particular during run-up on beaches and inclined dam walls. Fully three-dimensional viscous flow models based on (1)–(2) with free surfaces are now getting increased interest as these are becoming more accurate and computationally feasible [1, 24, 33, 69, 72, 79].

Efficient and reliable numerical solution of the incompressible Navier–Stokes equations for industrial flow or water resources applications is extremely challenging. Very rapid changes in the velocity field may take place in thin boundary layers close to solid walls. Complex geometries can also lead to rapid local changes in the velocity. Locally refined grids, preferably in combination with error estimation and automatic grid adaption, are hence a key ingredient in robust methods. Most implicit solution methods for the Navier–Stokes equations end up with saddle-point problems, which complicates the construction of efficient iterative methods for solving the linear systems arising from the discretization process. Implicit solution methods also make a demand for solving large systems of nonlinear algebraic equations. Many incompressible viscous flow computations involve large-scale flow applications with several million grid points and thereby a need for the next generation of super-computers before becoming engineering or scientific practice. We have also mentioned that Navier–Stokes solvers are often embedded in much more complex flow models, which couple turbulence, heat transfer, and multi-specie fluids. Before attacking such complicated problems it is paramount that the numerical state-of-the-art of Navier–Stokes solvers is satisfactory. Turek [84] summarizes the results of benchmarks that were used to assess the quality of solution methods and software for unsteady flow around a cylinder in 2D and 3D. The discrepancy in results for the lifting force shows that more research is needed to develop sufficiently robust and reliable methods.

Numerical methods for incompressible viscous flow is a major part of the rapidly growing field *computational fluid dynamics* (CFD). CFD is now emerging as an operative tool in many parts of industry and science. However, CFD

is not a mature field either from a natural scientist's or an application engineer's point of view; robust methods are still very much under development, many different numerical tracks are still competing, and reliable computations of complex multi-fluid flows are still (almost) beyond reach with today's methods and computers. We believe that at least a couple of decades of intensive research are needed to merge the seemingly different solution strategies and make them as robust as numerical models in, e.g., elasticity and heat conduction. Sound application of CFD today therefore requires advanced knowledge and skills both in numerical methods and fluid dynamics. To gain reliability in simulation results, it should be a part of common practice to compare the results from different discretizations, not only varying the grid spacings but also changing the discretization type and solution strategy. This requires a good overview and knowledge of different numerical techniques. Unfortunately, many CFD practitioners have a background from only one "numerical school" practicing a particular type of discretization technique and solution approach. One goal of the present paper is to provide a generic overview of the competing and most dominating methods in the part of CFD dealing with laminar incompressible viscous flow.

Writing a complete review of numerical methods for the Navier–Stokes equations is probably an impossible task. The book by Gresho and Sani [27] is a remarkable attempt to review the field, though with an emphasis on finite elements, but it required over 1000 pages and 48 pages of references. The page limits of a review paper demand the authors to only briefly report a few aspects of the field. Our focus is to present the basic ideas of the most fundamental solution techniques for the Navier–Stokes equations in a form that is accessible to a wide audience. The exposition is hence of the introductory and "engineering" type, keeping the amount of mathematical details to a modest level. We do not limit the scope to a particular spatial discretization technique, and therefore we can easily outline a common framework and reasoning which demonstrate the close connections between seemingly many different solution procedures in the literature. Hence, our hope is that this paper can help newcomers to the numerical viscous flow field see some structure in the jungle of Navier–Stokes solvers and papers, without having to start by digesting thick textbooks.

The literature on numerical solutions of the Navier–Stokes equations is overwhelming, and only a small fraction of the contributions is cited in this paper. Some books and reviews that the authors have found attractive are mentioned next. These references serve as good starting points for readers who want to study the contents of the present paper in more detail. Fletcher [21] contains a nicely written overview of some finite element and finite difference techniques for incompressible fluid flow (among many other topics). Gentle introductions to numerical methods and their applications to fluid flow can be found in the textbooks [3, 20, 28, 58, 59] (finite differences, finite volumes) and [60, 65, 66, 89] (finite elements). More advanced texts include [15, 26, 27, 30, 61, 25, 84, 86]. Readers with a background in functional analysis and special interest in mathematics and finite element methods are encouraged to address Girault and Raviart [25] and the reviews by Glowinski and Dean [16] and Rannacher [63].

Readers interested in the efficiency of solution algorithms for the Navier–Stokes equations should consult Turek [84]. Gresho and Sani's comprehensive book [27] is accessible to a wide audience and contains thorough discussions of many topics that are weakly covered in most other literature, e.g., questions related to boundary conditions. The book's extensive report on practical experience with various methods is indispensable for CFD scientists, software developers, and consultants. An overview of CFD books is available on the Internet [36].

Section 2 describes the natural first approach to solving the Navier–Stokes equations and points out some basic numerical difficulties. Necessary conditions to ensure stable spatial discretizations are treated in Section 3. Thereafter we consider approximate solution strategies where the Navier–Stokes equations are transformed to more common and tractable systems of partial differential equations. These strategies include modern stabilization techniques (Section 4.1), penalty methods (Section 4.2), artificial compressibility (Section 4.3), and operator splitting techniques (Section 5). The latter family of strategies is popular and widespread and are known under many names in the literature, e.g., projection methods and pressure (or velocity) correction methods. We end the overview of operating splitting methods with a framework where such methods can be viewed as special preconditioners in an iterative scheme for a fully implicit formulation of the Navier–Stokes equations. Section 7 mentions some examples of existing software packages for solving incompressible viscous flow problems, and in Section 8 we point out important areas for future research.

2 A Naive Derivation of Schemes

With a background from a basic course in the numerical solution of partial differential equations, one would probably think of (1) as some kind of heat equation and try the simplest possible scheme in time, namely an explicit forward step

$$\frac{\boldsymbol{v}^{\ell+1}-\boldsymbol{v}^{\ell}}{\Delta t}+\boldsymbol{v}^{\ell}\cdot\nabla\boldsymbol{v}^{\ell}=-\frac{1}{\varrho}\nabla p^{\ell}+\nu\nabla^2\boldsymbol{v}^{\ell}+\boldsymbol{g}^{\ell}\,. \tag{3}$$

Here, Δt is the time step and superscript ℓ denotes the time level. The equation can be trivially solved for $\boldsymbol{v}^{\ell+1}$, after having introduced, e.g., finite elements [27], finite differences [3], finite volumes [20], or spectral methods [11] to discretize the spatial operators. However, the fundamental problem with this approach is that the new velocity $\boldsymbol{v}^{\ell+1}$ does not, in general, satisfy the other equation (2), i.e., $\nabla\cdot\boldsymbol{v}^{\ell+1}\neq 0$. Moreover, there is no natural computation of $p^{\ell+1}$.

A possible remedy is to introduce a pressure at $p^{\ell+1}$ in (3), which leaves two unknowns, $\boldsymbol{v}^{\ell+1}$ and $p^{\ell+1}$, and hence requires a simultaneous solution of

$$\boldsymbol{v}^{\ell+1}+\frac{\Delta t}{\varrho}\nabla p^{\ell+1} = \boldsymbol{v}^{\ell}-\Delta t\boldsymbol{v}^{\ell}\cdot\nabla\boldsymbol{v}^{\ell}+\Delta t\nu\nabla^2\boldsymbol{v}^{\ell}+\Delta t\boldsymbol{g}^{\ell}, \tag{4}$$

$$\nabla\cdot\boldsymbol{v}^{\ell+1} = 0\,. \tag{5}$$

We can eliminate $\boldsymbol{v}^{\ell+1}$ by taking the divergence of (4) to obtain a Poisson

equation for the pressure,

$$\nabla^2 p^{\ell+1} = \frac{\varrho}{\Delta t}\nabla\cdot(\boldsymbol{v}^\ell - \Delta t\boldsymbol{v}^\ell\cdot\nabla\boldsymbol{v}^\ell + \Delta t\nu\nabla^2\boldsymbol{v}^\ell + \Delta t\boldsymbol{g}^\ell)\,. \tag{6}$$

However, there are no natural boundary conditions for $p^{\ell+1}$. Hence, solving (6) and then finding $\boldsymbol{v}^{\ell+1}$ trivially from (4) is therefore not in itself a sufficient solution strategy. More sophisticated variants of this method are considered in Section 5, but the lack of explicit boundary data for $p^{\ell+1}$ will remain a problem.

More implicitness of the velocity terms in (1) can easily be introduced. One can, for example, try a semi-implicit approach, based on a Backward Euler scheme, using an "old" velocity (as a linearization technique) in the convective term $\boldsymbol{v}\cdot\nabla\boldsymbol{v}$:

$$(1 + \Delta t\boldsymbol{v}^\ell\cdot\nabla - \Delta t\nu\nabla^2)\boldsymbol{v}^{\ell+1} + \frac{\Delta t}{\varrho}\nabla p^{\ell+1} = \boldsymbol{v}^\ell + \Delta t\boldsymbol{g}^{\ell+1}, \tag{7}$$

$$\nabla\cdot\boldsymbol{v}^{\ell+1} = 0\,. \tag{8}$$

This problem has the proper boundary conditions since (7) and (8) have the same order of the spatial operators as the original system (1)–(2). Using some discretization in space, one arrives in both cases at a linear system, which can be written on block form:

$$\begin{bmatrix} \boldsymbol{N} & \boldsymbol{Q} \\ \boldsymbol{Q}^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix}. \tag{9}$$

The vector $\boldsymbol{u}$ contains in this context all the spatial degrees of freedom (i.e., grid point values) of the vector field $\boldsymbol{v}^{\ell+1}$, whereas $\boldsymbol{p}$ is the vector of pressure degrees of freedom in the grid.

A fully implicit approach, using a backward Euler scheme for (1), where the convective term $\boldsymbol{v}\cdot\nabla\boldsymbol{v}$ is evaluated as $\boldsymbol{v}^{\ell+1}\cdot\nabla\boldsymbol{v}^{\ell+1}$, leads to a nonlinear equation in $\boldsymbol{v}^{\ell+1}$. Standard Newton or Picard iteration methods result in a sequence of matrix systems of the form (9) at each time level.

In contrast to linear systems arising from standard discretization of, e.g., the diffusion equation, the system (9) may be singular. Special spatial discretization or stabilizing techniques are needed to ensure an invertible matrix in (9) and are reviewed in Section 3 and 4. In the simplest case, $\boldsymbol{N}$ is a symmetric and positive definite matrix (this requires the convective term $\boldsymbol{v}\cdot\nabla\boldsymbol{v}$ to be evaluated explicitly at time level ℓ, such that the term appears on the right-hand side of (7)), and $\boldsymbol{Q}$ is a rectangular matrix. The stable spatial discretizations are designed such that the matrix $\boldsymbol{Q}^T\boldsymbol{Q}$ is non-singular. It should be noted that these conditions on $\boldsymbol{N}$ and $\boldsymbol{Q}$ lead to the property that the coefficient matrix in (9) is symmetric and non-singular but indefinite. This indefiniteness causes some difficulties. For example, a standard iterative method like the preconditioned conjugate gradient method can not be directly used. In fact, preconditioners for these saddle point problems are much more delicate to construct even when using more general solvers like, e.g., GMRES and may lead to breakdown if

not constructed properly. Many of the time stepping procedures for the Navier-Stokes system have been partially motivated by the desire to avoid the solution of systems of the form (9). However, as we shall see later, such a strategy will introduce other difficulties.

3 Spatial Discretization Techniques

So far we have only been concerned with the details of the time discretization. Now we shall address spatial discretization techniques for the systems (4)–(5) or (7)–(8).

3.1 Finite Differences and Staggered Grids

Initial attempts to solve the Navier–Stokes equations employed straightforward centered finite differences to the spatial operators on a regular grid, with the pressure and velocity components being unknown at the corners of each cell. Two typical terms in the equations would then be discretized as follows in a uniform 2D grid:

$$\left[\frac{\partial p}{\partial x}\right]_{i,j}^{\ell+1} \approx -\frac{p_{i+1,j}^{\ell+1} - p_{i-1,j}^{\ell+1}}{2\Delta x} \tag{10}$$

and

$$\left[\frac{\partial^2 u}{\partial y^2}\right]_{i,j}^{\ell} \approx \frac{u_{i,j-1}^{\ell} - 2u_{i,j}^{\ell} + u_{i,j+1}^{\ell}}{\Delta y^2},$$

where Δx and Δy are uniform spatial cell sizes, $\phi_{i,j}^{\ell}$ means the numerical value of a function ϕ at the point with spatial index (i,j) at time level ℓ.

Two types of instabilities were soon discovered, associated with this type of spatial discretization. The pressure can be highly oscillatory or even undetermined by the discrete system, although the corresponding velocities may be well approximated. The reason for this phenomenon is that the symmetric difference operator (10) will annihilate checkerboard pressures, i.e., pressures which oscillate between 1 and -1 on each grid line connecting the grid points. In fact, if the vertices are colored in a checkerboard pattern, then the pressure at the black vertices will not be related to the pressure at the white vertices. Hence, the pressure is undetermined by the discrete system and wild oscillations or overflow will occur. This instability is related to whether the system (9) is singular or not. There is also a "softer" version of this phenomenon when (9) is nearly singular. Then the pressure will not necessarily oscillate, but it will not converge to the actual solution either.

The second type of instability is visible as non-physical oscillations in the velocities at high Reynolds numbers. This instability is the same as encountered when solving advection-dominated transport equations, hyperbolic conservation laws, or boundary layer equations and can be cured by well-known techniques, among which upwind differences represent the simplest approach. We shall not

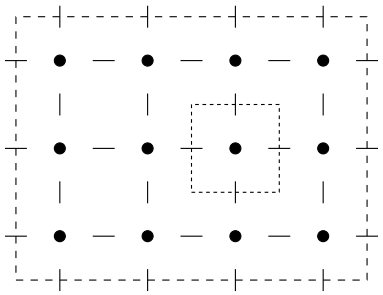

Figure 1: Example on a staggered grid for the Navier–Stokes equations, where p and $\boldsymbol{v} = (u, v)$ are unknown at different spatial locations. The $\bullet$ denotes p points, $-$ denotes u points, whereas $|$ denotes v points.

be concerned with this topic in the present paper, but the interested reader can consult the references [9, 20, 21, 27, 59, 71] for effective numerical techniques.

The remedy for oscillatory or checkerboard pressure solutions is, in a finite difference context, to introduce a staggered grid in space. This means that the primary unknowns, the pressure and the velocity components, are sought at different points in the grid. Figure 1 displays such a grid in 2D, and Figure 2 zooms in on a cell and shows the spatial indices associated with the point values of the pressure and velocity components that enter the scheme.

Discretizing the terms $-\partial p/\partial x$ and $\partial^2 u/\partial y^2$ on the staggered grid at a point with spatial indices $(i, j + \frac{1}{2})$ now results in

$$\left[\frac{\partial p}{\partial x}\right]^{\ell+1}_{i,j+\frac{1}{2}} \approx -\frac{p^{\ell+1}_{i+\frac{1}{2},j+\frac{1}{2}} - p^{\ell+1}_{i-\frac{1}{2},j+\frac{1}{2}}}{\Delta x}$$

and

$$\left[\frac{\partial^2 u}{\partial y^2}\right]^{\ell}_{i,j+\frac{1}{2}} \approx \frac{u^{\ell}_{i,j-\frac{1}{2}} - 2u^{\ell}_{i,j+\frac{1}{2}} + u^{\ell}_{i,j+\frac{3}{2}}}{\Delta y^2}.$$

The staggered grid is convenient for many of the derivatives appearing in the equations, but for the nonlinear terms it is necessary to introduce averaging. See, for instance, [3, 21, 20, 28] for more details regarding discretization on staggered grids.

Finite volume methods are particularly popular in CFD. Although the final discrete equations are similar to those obtained by the finite difference method, the reasoning is different. One works with the integral form of the equations (1)–(2), obtained either by integrating (1)–(2) or by direct derivation from basic physical principles. The domain is then divided into control volumes. These control volumes are different for the integral form of (2) and the various components of the integral form of (1). For example, in the grid in Figure 1 the dotted cell, also appearing in Figure 2, is a typical control volume for the integral form of the equation of continuity, whereas the control volumes for the

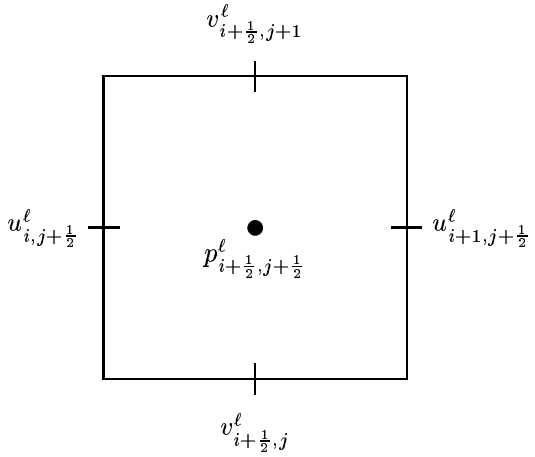

Figure 2: A typical cell in a staggered grid.

components of the equation of motion are shifted half a cell size in the various spatial directions. The governing equations in integral form involve volume/area integrals over the interior of a control volume and surface/line integrals over the sides. In the computation of these integrals, there is freedom to choose the type of interpolation for $\boldsymbol{v}$ and p and the numerical integration rules. Many CFD practitioners prefer finite volume methods because the derivation of the discrete equations is based directly on the underlying physical principles, thus resulting in "physically sound" schemes. From a mathematical point of view, finite volume, difference, and element methods are closely related, and it is difficult to decide that one approach is superior to the others; these spatial discretization methods have different advantages and disadvantages. We refer to textbooks like [3] and [20] for detailed information about the finite volume discretization technology.

Staggered grids are widespread in CFD. However, in recent years other ways of stabilizing the spatial discretization have emerged. Auxiliary terms in the equations or certain splittings of the operators in the Navier–Stokes equations can allow stable pressure solutions also on a standard grid. Avoiding staggered grids is particularly convenient when working with curvelinear grids. Much of the fundamental understanding of stabilizing the spatial discretization has arisen from finite element theory. We therefore postpone the discussion of particular stabilization techniques until we have reviewed the basics of finite element approximations to the spatial operators in the time-discrete Navier–Stokes equations.

3.2 Mixed Finite Elements

The staggered grids use different interpolation for the pressure and the velocity, hence we could call it mixed interpolation. In the finite element world the analog interpolation is referred to as mixed elements. The idea is, basically, to employ different basis functions for the different unknowns. A finite element function is expressed as a linear combination of a set of prescribed basis functions, also called shape functions or trial functions [88]. These basis functions are defined relative to a grid, which is a collection of elements (triangles, qaudrilaterals, tetrahendra, or boxes), so the overall quality of a finite element approximation depends on the shape of the elements and the type of basis functions. Normally, the basis functions are lower-order polynomials over a single element.

One popular choice of basis functions for viscous flow is quadratic piecewise polynomials for the velocity components and linear piecewise polynomials for the pressure. This was in fact the spatial discretization used in the first report, by Taylor and Hood [76], on finite element methods for the Navier–Stokes equations.

The Babuska-Brezzi (BB) condition [8, 25, 27, 30] is central for ensuring that the linear system of the form (9) is non-singular. Much of the mathematical theory and understanding of importance for the numerical solution of the Navier-Stokes equations has been developed for the simplified Stokes problem, where the acceleration terms on the left-hand side of (1) vanish:

$$\mathbf{0} = -\frac{1}{\varrho}\nabla p + \nu\nabla^2 \boldsymbol{v} + \boldsymbol{g}, \tag{11}$$

$$\nabla \cdot \boldsymbol{v} = 0\,. \tag{12}$$

We use the Galerkin method to formulate the discrete problem, seeking approximations

$$\boldsymbol{v} \approx \hat{\boldsymbol{v}} = \sum_{r=1}^{d}\sum_{i=1}^{n} v_i^r \boldsymbol{N}_i^r, \tag{13}$$

$$p \approx \hat{p} = \sum_{i=1}^{m} p_i L_i, \tag{14}$$

where $\boldsymbol{N}_i^r = N_i \boldsymbol{e}_r$, N_i and L_i are some scalar basis functions and $\boldsymbol{e}_r$ is the unity vector in the direction r. Here d is the number of spatial dimensions, i.e., 2 or 3. The number of velocity unknowns is dn, whereas the pressure is represented by m unknowns. Using N_i as weighting function for (11) and L_i as weighting function for (12), and integrating over Ω, one can derive a linear system for the coefficients v_i^r and p_i :

$$\sum_{j=1}^{n} \bar{N}_{ij} v_j^r + \sum_{j=1}^{m} Q_{ij}^r p_j = f_i^r, \quad i = 1, \ldots, dn, r = 1, \ldots, d \tag{15}$$

$$\sum_{r=1}^{d}\sum_{j=1}^{n} Q_{ji}^r v_j^r = 0, \quad i = 1, \ldots, m \tag{16}$$

where

$$\bar{N}_{ij} = \int_\Omega \nu \nabla N_i \cdot \nabla N_j d\Omega, \tag{17}$$

$$Q^r_{ij} = \frac{1}{\varrho}\int_\Omega \frac{\partial L_i}{\partial x_r} N_j d\Omega = -\frac{1}{\varrho}\int_\Omega \frac{\partial N_i}{\partial x_r} L_j d\Omega + \frac{1}{\varrho}\int_{d\Omega} N_i L_j n_r d\Gamma, \tag{18}$$

$$f^r_i = \int_\Omega g^r \boldsymbol{N}^r_i d\Omega\,. \tag{19}$$

We shall write such a system on block matrix form (like (9)):

$$\left[\begin{array}{cc} \boldsymbol{N} & \boldsymbol{Q} \\ \boldsymbol{Q}^T & \boldsymbol{0} \end{array}\right] \left[\begin{array}{c} \boldsymbol{u} \\ \boldsymbol{p} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{f} \\ \boldsymbol{0} \end{array}\right], \quad \boldsymbol{N} = \left[\begin{array}{ccc} \bar{\boldsymbol{N}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \bar{\boldsymbol{N}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \bar{\boldsymbol{N}} \end{array}\right], \quad \boldsymbol{Q} = \left[\begin{array}{c} \boldsymbol{Q}^1 \\ \boldsymbol{Q}^2 \\ \boldsymbol{Q}^3 \end{array}\right] \tag{20}$$

Here $\bar{\boldsymbol{N}}$ is the matrix with elements $\bar{N}_{ij}$, $\boldsymbol{Q}^r$ has elements Q^r_{ij}, and

$$\boldsymbol{u} = (v^1_1, \ldots, v^1_n, v^2_1, \ldots, v^2_n, v^3_1, \ldots, v^3_n)^T, \quad \boldsymbol{p} = (p_1, \ldots, p_n)^T\,. \tag{21}$$

The $\boldsymbol{N}$ matrix is seen to be $dn \times dn$, whereas $\boldsymbol{Q}$ is $m \times dn$. In (20)-(21) we have assumed that $d = 3$. Moreover, we have multiplied the equation of continuity by the factor $-1/\varrho$ to obtain a symmetric linear system.

We shall now go through some algebra related to the block form of the Stokes problem, since this algebra will be needed later in Section 5.6. Let us write the discrete counterpart to (11)–(12) as:

$$\boldsymbol{N}\boldsymbol{u} + \boldsymbol{Q}\boldsymbol{p} = \boldsymbol{f}, \tag{22}$$

$$\boldsymbol{Q}^T\boldsymbol{u} = \boldsymbol{0}\,. \tag{23}$$

The matrices $\boldsymbol{N}$ and $\boldsymbol{Q}$ can in principle arise from any spatial discretization method, e.g., finite differences, finite volumes, or finite elements, although we will specifically refer to the latter in what follows. First, we shall ask the question: What conditions on $\boldsymbol{N}$ and $\boldsymbol{Q}$ are needed to ensure that $\boldsymbol{u}$ and $\boldsymbol{p}$ are uniquely determined? We assume that $\boldsymbol{N}$ is positive definite (this assumption is actually the first part of the BB condition, which is satisfied for all "standard" elements). We can then multiply (22) by $\boldsymbol{N}^{-1}$ to obtain an expression for $\boldsymbol{u}$, which can be inserted in (23). The result is a linear system for $\boldsymbol{p}$:

$$-\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{Q}\boldsymbol{p} = \boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{f}\,. \tag{24}$$

Once the pressure is known, the velocities are found by solving

$$\boldsymbol{N}\boldsymbol{u} = (\boldsymbol{f} - \boldsymbol{Q}\boldsymbol{p})\,.$$

To obtain a uniquely determined $\boldsymbol{u}$ and $\boldsymbol{p}$, $\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{Q}$, which is referred to as the *Schur complement*, must be non-singular. A necessary sufficient condition to ensure this is $\mathrm{Ker}(\boldsymbol{Q}) = \{0\}$, which is equivalent to requiring that

$$\sup_{\hat{\boldsymbol{v}}} \int_\Omega \hat{p}\nabla \cdot \hat{\boldsymbol{v}} > 0, \tag{25}$$

for all discrete pressure $\hat{p} \neq 0$, where the supremum is taken over all discrete velocities on the form (13). This guarantees solvability, but to get convergence of the numerical method, one also needs stability. This is where the famous BB condition comes in:

$$\inf_{p} \sup_{\hat{\boldsymbol{v}}} \frac{\int_\Omega \hat{p} \nabla \cdot \hat{\boldsymbol{v}}}{\|\hat{\boldsymbol{v}}\|_1 \|\hat{p}\|_0} \geq \gamma > 0 \,. \tag{26}$$

Here, γ is independent of the discretization parameters, and the inf is taken over all $\hat{p} \neq 0$ on the form (14). The condition (26) is stated in numerous books and papers. Here we emphasize the usefulness of (26) as an operative tool for determining which elements for p and $\boldsymbol{v}$ that are "legal", in the sense that the elements lead to a solvable linear system and a stable, convergent method. For example, the popular choice of standard bilinear elements for $\boldsymbol{v}$ and piecewise constant elements for p voilates (26), whereas standard quadratic triangular elements for $\boldsymbol{v}$ and standard linear triangules for p fulfill (26).

Provided the BB condition is fulfilled, with γ not depending on the mesh, one can derive an error estimate for the discretization of the Navier-Stokes equations:

$$\|\hat{\boldsymbol{v}} - \boldsymbol{v}\|_1 + \|\hat{p} - p\|_0 \leq C(h^k \|\boldsymbol{v}\|_{k+1} + h^{l+1} \|p\|_{l+1}), \tag{27}$$

This requires the exact solutions $\boldsymbol{v}$ and p to be in $[H^{k+1}(\Omega)]^d$ and $H^{l+1}(\Omega)$, respectively. The constant C is independent of the spatial mesh parameter h. The degree of the piecewise polynomial used for the velocity and the pressure is k and l, respectively, (see e.g. [29] or [25]). Since (27) involves the H^1 norm of $\boldsymbol{v}$, and the convergence rate of $\boldsymbol{v}$ in L^2 norm is one order higher, it follows from the estimate (27) that $k = l + 1$ is the optimal choice, i.e., the velocity is approximated with accuracy of one higher order than the pressure. For example, the Taylor-Hood element [76] with quadratic velocity components and linear pressure gives quadratic and linear L_2-convergence in the mesh parameter for the velocities and pressure, respectively (under reasonable assumptions), see [5].

In simpler words, one could say that the computer resources are not wasted. We get what we *can* and *should* get. Elements that do not satisfy the BB condition may give an approximation that does not converge to the solution, and if it does, it may not converge as fast as one should expect from the order of the elements.

Numerous mixed finite elements satisfying the BB condition have been proposed over the years. However, elements not satisfying the BB condition may also work well. The element with bilinear velocities and constant pressure, which does violates the BB condition, is popular and usable in many occasions. A comprehensive review of mixed finite elements for incompressible viscous flow can be found in [27].

4 Stabilization Techniques

Staggered grids or mixed finite elements can be challenging from an implementational point of view, especially when using unstructured, adaptive and/or

hierarchical grids. Therefore, there has been significant interest in developing stabilization techniques which allow standard grids and equal order interpolation of $\boldsymbol{v}$ and p.

The singularity of the matrix (9) can be circumvented by introducing a stabilization matrix $\epsilon \boldsymbol{D}$ and possibly a perturbation of the right hand side, $\epsilon \boldsymbol{d}$,

$$\begin{bmatrix} \boldsymbol{N} & \boldsymbol{Q} \\ \boldsymbol{Q}^T & -\epsilon \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ -\epsilon \boldsymbol{d} \end{bmatrix}, \tag{28}$$

where ϵ is a parameter that should be chosen either from physical knowledge or by other means. It can also be a spatially local parameter, which can be important for anisotropic meshes and boundary layer problems. There are mainly three methods used to construct $\epsilon \boldsymbol{D}$, all based on perturbed versions of the equation of continuity,

$$\nabla \cdot \boldsymbol{v} = \epsilon \nabla^2 p, \tag{29}$$

$$\nabla \cdot \boldsymbol{v} = -\epsilon p, \tag{30}$$

$$\nabla \cdot \boldsymbol{v} = -\epsilon \frac{\partial p}{\partial t}. \tag{31}$$

The approach (29) was derived with the purpose of stabilizing pressure oscillations and allowing standard grids and elements. Section 4.1 deals with this approach. The equations (30) and (31) were not derived as stabilization methods, but were initiated from alternative physical and mathematical formulations of viscous incompressible flow, as we outline in Sections 4.2 and 4.3.

4.1 Pressure Stabilization Techniques

Finite elements not satisfying the BB condition often lead to non-physical oscillations in the pressure field. It may therefore be tempting to introduce a regularization based on $\nabla^2 p$, which will smooth the pressure solution [8]. One can show that the BB condition can be avoided by, e.g., introducing a stabilization term in the equation of continuity as shown in (29). It is common to write this perturbed equation with a slightly different perturbation parameter;

$$\nabla \cdot \boldsymbol{v} = \epsilon h^2 \nabla^2 p, \tag{32}$$

where ϵ is a constant to be chosen. Now the velocities and the pressure can be represented with equal order, "standard" finite elements. We have introduced an $\mathcal{O}(h^2)$ perturbation of the problem, and there is hence no point in using higher-order elements. Consistent generalizations that also apply to higher-order elements have been proposed, a review can be found in Gresho and Sani [27] and Franca et al. in [30]. The idea behind these methods is that one observes that by taking the divergence of (11) we get an equation that includes a $\nabla^2 p$ term like in (32),

$$\frac{1}{\varrho} \nabla^2 p = \nabla \cdot (\nu \nabla^2 \boldsymbol{v}) + \nabla \cdot \boldsymbol{g}. \tag{33}$$

This divergence of (11) can be represented by the weak form

$$\int_\Omega (-\frac{1}{\varrho}\nabla\hat{p} + \nu\nabla^2\hat{\boldsymbol{v}} + \boldsymbol{g}) \cdot \nabla L_i \, d\Omega = 0,$$

where the pressure basis functions are used as weighting functions. The left-hand side of this equation can then be added to (16) with a local weighting parameter ϵh_K^2 in each element. The result becomes

$$\sum_{r=1}^{d}\sum_{j=1}^{n} \hat{Q}_{ji}^r v_j^r - \epsilon \sum_{j=1}^{m} D_{ij} p_j = -\epsilon d_i, \quad i = 1, \ldots, m, \tag{34}$$

where

$$D_{ij} = \sum_K h_K^2 \int_{\Omega_K} \nabla L_i \cdot \nabla L_j d\Omega, \tag{35}$$

$$\hat{Q}_{ij}^r = Q_{ij}^r + \epsilon \sum_K h_K^2 \int_{\Omega_K} \nu\nabla^2 \boldsymbol{N}_j^r \frac{\partial L_i}{\partial x_r} d\Omega \tag{36}$$

$$d_i^r = \sum_K h_K^2 \int_{\Omega_K} g^r \frac{\partial L_i}{\partial x_r} d\Omega \, . \tag{37}$$

The sum over K is to be taken over all elements; Ω_K is the domain of element K and h_K is the local mesh size. We see that this stabilization is not symmetric since $\hat{Q}_{ij}^r \neq \hat{Q}_{ji}^r$, however it is easy to see that a symmetric stabilization can be made by an adjustment of (15), such that

$$\int_\Omega (-\frac{1}{\varrho}\nabla\hat{p} + \nu\nabla^2\hat{\boldsymbol{v}} + \boldsymbol{g}) \cdot \nabla^2 \boldsymbol{N}_i^r d\Omega$$

is added to (15) with the same local weighting parameter. The use of second-order derivatives excludes linear polynomials for $\boldsymbol{N}_i^r$. Detailed analysis of stabilization methods for both Stokes and Navier–Stokes equations can be found in [82].

One problem with stabilization techniques of the type outlined here is the choice of ϵ, since the value of ϵ influences the accuracy of the solution. If ϵ is too small we will experience pressure oscillations, and if ϵ is too large the accuracy of the solution deteriorates, since the solution is far from divergence free locally, although it is divergence free globally [27]. The determination of ϵ is therefore important. Several more or less complicated techniques exist, among the simplest is the construction of 'optimal bubbles' which is equivalent to the discretization using the MINI element [8, 27]. Problems with this approach have been reported; one often experiences $\mathcal{O}(h)$ pressure oscillations in boundary layers with stretched elements, but a fix (multiply ϵ with a proper factor near the boundary layer) is suggested in [54]. An adaptive stabilization parameter calculated locally from properties of the element matrices and vectors is suggested in [78]. This approach gives a more robust method in the boundary layers.

4.2 Penalty Methods

A well-known result from variational calculus is that minimizing a functional

$$J(v) = \int_\Omega |\nabla v|^2 d\Omega$$

over all functions v in the function space $H^1(\Omega)$, such that $v|_{\partial\Omega} = g$ where g is the prescribed boundary values, is equivalent to solving the Laplace problem

$$\nabla^2 u = 0 \text{ in } \Omega, \quad u = g \text{ on } \partial\Omega\,.$$

The Stokes problem (11)–(12) can be recast into a variational problem as follows: Minimize

$$J(\boldsymbol{w}) = \int_\Omega \varrho\,(\nu \nabla \boldsymbol{w} : \nabla \boldsymbol{w} - \boldsymbol{g} \cdot \boldsymbol{w})\, d\Omega$$

over all $\boldsymbol{w}$ in some suitable function space, subject to the constraint

$$\nabla \cdot \boldsymbol{w} = 0\,.$$

Here, $\nabla \boldsymbol{w} : \nabla \boldsymbol{w} = \sum_r \sum_s w_{r,s} w_{r,s}$ is the "inner product" of two tensors (and $w_{r,s}$ means $\partial w_r / \partial x_s$). As boundary conditions, we assume that $\boldsymbol{w}$ is known or the stress vector vanishes, for the functional $J(\boldsymbol{w})$ to be correct (extension to more general conditions is a simple matter). This constrained minimization problem can be solved by the method of Lagrange multipliers: Find stationary points of

$$\hat{J}(\boldsymbol{w}, p) = J(\boldsymbol{w}) - \int_\Omega p \nabla \cdot \boldsymbol{w}\, d\Omega$$

with respect to $\boldsymbol{w}$ and p, $-p$ being the Lagrange multiplier. The solution $(\boldsymbol{w}, p)$ is a saddle point of $\hat{J}$,

$$\hat{J}(\boldsymbol{w}, q) \le \hat{J}(\boldsymbol{w}, p) \le \hat{J}(\boldsymbol{v}, p)$$

and fulfills the Stokes problem (11)–(12).

The penalty method is a way of solving constrained variational problems approximately. One works with the modified functional

$$\tilde{J}(\boldsymbol{w}) = J(\boldsymbol{w}) + \frac{1}{2}\lambda^2 \int_\Omega (\nabla \cdot \boldsymbol{w})^2 d\Omega,$$

where λ is a prescribed, large parameter. The solution is governed by the equation

$$\frac{1}{\varrho}\lambda \nabla(\nabla \cdot \boldsymbol{v}) + \nu \nabla^2 \boldsymbol{v} = \boldsymbol{g}\,. \tag{38}$$

or the equivalent mixed formulation,

$$-\nu \nabla^2 \boldsymbol{v} + \frac{1}{\varrho}\nabla p = \boldsymbol{g}, \tag{39}$$

$$\nabla \cdot \boldsymbol{v} + \frac{1}{\lambda} p = 0\,. \tag{40}$$

For numerical solution, (38) is a tremendous simplification at first sight; equation (38) is in fact equivalent to the equation of linear elasticity, for which robust numerical methods are well known. The penalty method does not seem to need mixed elements or staggered grids and is hence easy to implement.

The governing equation (38) is only an approximation to (11)–(12), where the latter model is obtained in the limit $\lambda \to \infty$. A too low λ leads to mass loss, whereas a large λ value leads to numerical difficulties (known as the locking problem in elasticity). Because of the large λ parameter, explicit time discretization leads to impractical small time steps, and implicit schemes in time are therefore used, with an associated demand of solving matrix systems. The disadvantage of the penalty method is that efficient *iterative* solution of these matrix systems is hard to construct. The discrete approximations of the system (38) will be positive definite. However, as λ approach infinity the system will tend to a discrete Stokes system, i.e., a discrete version of (39)-(40) with $\frac{1}{\lambda} = 0$. Hence, in the limit the elimination of the pressure is impossible, and this effect results in bad conditioning of the systems derived from (38) when λ is large. The dominating solution techniques have therefore been variants of Gaussian elimination. However, progress has been made with iterative solution techniques, see Reddy and Reddy [68].

The penalty method has a firm theoretical basis for the Stokes problem [64]. Ad hoc extensions to the full Navier–Stokes equations are done by simply replacing equation (2) by

$$p = -\lambda \nabla \cdot \boldsymbol{v}$$

and eliminating the pressure p. This results in the governing flow equation

$$\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{v} = \frac{\lambda}{\varrho} \nabla (\nabla \cdot \boldsymbol{v}) + \nu \nabla^2 \boldsymbol{v} + \boldsymbol{g} \,. \tag{41}$$

In a sense, this is a nonlinear and time-dependent version of the standard linear elasticity equations.

One problem with the penalty method and standard elements is often referred to as *locking*. The locking phenomena can be illustrated by seeking a divergence-free velocity field subject to homogeneous Dirichlet boundary conditions on a regular finite element grid. For the standard linear elements the only solution to this problem is $\boldsymbol{v} = \boldsymbol{0}$. In the case of the penalty method we see that as $\lambda \to \infty$, $\boldsymbol{v} = \boldsymbol{0}$ is the only solution to (41) unless the matrix associated with the λ term is singular. One common way to avoid locking is, in a finite element context, to introduce *selective reduced integration*, which causes the matrix associated with the λ term to be singular. The selective reduced integration consists in applying a Gauss-Legendre rule to the λ term that is of one order lower than the rule applied to other terms (provided that rule is of minimum order for the problem in question). For example, if bilinear elements are employed for $\boldsymbol{v}$, the standard 2×2 Gauss-Legendre rule is used for all integrals, except those containing λ, which are treated by the 1×1 rule. The same technique is known from linear elasticity problems when the material approaches the incompressible limit. We refer to [34, 64] or standard textbooks [65, 66, 89] for more details.

The use of selective reduced integration is justified by the fact that under certain conditions the reduced integration is equivalent to *consistent integration*, which is defined as the integration rule that is obtained if mixed elements were used to discretize (39)- (40) before eliminating the pressure to obtain (38). This equivalence result does, however, need some conditions on the elements. For instance, the difference between consistent and reduced integration was investigated in [19], and they reported much higher accuracy of mixed methods with consistent integration when using curved higher-order elements.

The locking phenomena is related to the finite element space and not to the equations themselves. For standard linear elements the incompressibility constraint will affect all degrees of freedom and therefore the approximation will be poor. Another way of circumventing this problem can therefore be to use elements where the incompressibility constraint will only affect some of the degrees of freedom, e.g., the element used to approximate Darcy-Stokes flow [55] .

The penalty formulation can also be justified by physical considerations (Stokes' viscosity law [23]). We also mention that the method can be viewed as a *velocity* Schur complement method (cf. *pressure* Schur complement methods in Section 5.8). The Augmented Lagrangian method is a regularization technique closely related to the penalty method. For a detailed discussion we refer to the book by Fortin and Glowinski [22].

4.3 Artificial Compressibility Methods

If there had been a term $\partial p/\partial t$ in the equation of continuity (2), the system of partial differential equation for viscous flow would be similar to the shallow water equations (with a viscous term). Simple explicit time stepping methods would then be applicable.

To introduce a pressure derivative in the equation of continuity, we consider the Navier–Stokes equations for compressible flow:

$$\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{v} = -\frac{1}{\varrho}\nabla p + \nu \nabla^2 \boldsymbol{v} + \boldsymbol{g}, \tag{42}$$

$$\frac{\partial \varrho}{\partial t} + \nabla \cdot (\varrho \boldsymbol{v}) = 0\,. \tag{43}$$

In (42) we have neglected the bulk viscosity since we aim at a model with small compressibility to be used as an approximation to incompressible flow. The assumption of small compressibility, under isothermal conditions, suggests the linearized equation of state

$$p = p(\varrho) \approx p_0 + c_0^2(\varrho - \varrho_0), \tag{44}$$

where $c_0^2 = (\partial p/\partial \varrho)_0$ is the velocity of sound at the state (ϱ_0, p_0). We can now eliminate the density in the equation of continuity (43), resulting in

$$\frac{\partial p}{\partial t} + c_0^2 \varrho_0 \nabla \cdot \boldsymbol{v} = 0\,. \tag{45}$$

Equations (42) and (45) can be solved by, e.g., explicit forward differences in time. Here we list a second-order accurate leap-frog scheme, as originally suggested by Chorin [13]:

$$\frac{\boldsymbol{v}^{\ell+1} - \boldsymbol{v}^{\ell-1}}{2\Delta t} + \boldsymbol{v}^{\ell} \cdot \nabla \boldsymbol{v}^{\ell} = -\frac{1}{\varrho_0}\nabla p^{\ell} + \nu \nabla^2 \boldsymbol{v}^{\ell} + \boldsymbol{g}^{\ell} \tag{46}$$

$$\frac{p^{\ell+1} - p^{\ell-1}}{2\Delta t} = -c_0^2 \varrho_0 \nabla \cdot \boldsymbol{v}^{\ell}\,. \tag{47}$$

This time scheme can be combined with centered spatial finite differences on standard grids or on staggered grids; Chorin [13] applied a DuFort-Frankel scheme on a standard grid. When solving the similar shallow water equations, most practitioners apply a staggered grid of the type in Figure 1 as this give a more favorable numerical dispersion relation. Peyret and Taylor [59] recommend staggered grids for slightly compressible viscous flow for the same reason.

Artificial compressibility methods are often used to obtain a stationary solution. In this case, one can introduce $\alpha = \varrho_0 c_0^2$ and use α and Δt for optimizing a pseudo-time evolution of the flow towards a stationary state. A basic problem with the approach is that the time step Δt is limited by the inverse of c_0^2, which results in very small time steps when simulating incompressibility ($c_0 \to \infty$). Implicit time stepping in (42) and (45) can then be much more efficient. In fact, explicit temporal schemes in (46)–(47) are closely related to operator splitting techniques (Sections 5 and 5.6), where the pressure Poisson equation is solved by a Jacobi-like iterative method [59]. Therefore, the scheme (46)–(47) is a very slow numerical method unless the flow exhibits rapid transient behavior of interest. Having said this, we should also add that artificial compressibility methods with explicit time integration have been very popular because of the trivial implementation and parallelization.

5 Operator Splitting Methods

The most popular numerical solution strategies today for the Navier–Stokes equations are based on operator splitting. This means that the system (1)–(2) is split into a series of simpler, familiar equations, such as advection equations, diffusion equations, advection-diffusion equations, Poisson equations, and explicit/implicit updates. Efficient numerical methods are much easier to construct for these standard equations than for the original system (1)–(2) directly. In particular, the evolution of the velocity consists of two main steps. First we neglect the incompressibility condition and compute a predicted velocity. Thereafter, the velocity is corrected by performing "a projection" onto the divergence free vector fields.

5.1 Explicit Schemes

To illustrate the basics of operator splitting ideas, we start with a forward step in (1):

$$\boldsymbol{v}^{\ell+1} = \boldsymbol{v}^{\ell} - \Delta t \boldsymbol{v}^{\ell} \cdot \nabla \boldsymbol{v}^{\ell} - \frac{\Delta t}{\varrho} \nabla p^{\ell} + \Delta t \nu \nabla^2 \boldsymbol{v}^{\ell} + \Delta t \boldsymbol{g}^{\ell} \,. \tag{48}$$

The problem is that $\boldsymbol{v}^{\ell+1}$ does not satisfy the equation of continuity (2), i.e., $\nabla \cdot \boldsymbol{v}^{\ell+1} \neq 0$. Hence, we cannot claim that $\boldsymbol{v}^{\ell+1}$ in (48) is the velocity at the new time level $\ell + 1$. Instead, we view this velocity as a *predicted* (also called tentative or intermediate) velocity, denoted here by $\boldsymbol{v}^*$, and try to use the incompressibility constraint to compute a correction $\boldsymbol{v}^c$ such that $\boldsymbol{v}^{\ell+1} = \boldsymbol{v}^* + \boldsymbol{v}^c$. For more flexible control of the pressure information used in the equation for $\boldsymbol{v}^*$ we multiply the pressure term ∇p^{ℓ} by an adjustable factor β:

$$\boldsymbol{v}^{*} = \boldsymbol{v}^{\ell} - \Delta t \boldsymbol{v}^{\ell} \cdot \nabla \boldsymbol{v}^{\ell} - \Delta t \frac{\beta}{\varrho} \nabla p^{\ell} + \Delta t \nu \nabla^2 \boldsymbol{v}^{\ell} + \Delta t \boldsymbol{g}^{\ell} \,. \tag{49}$$

The $\boldsymbol{v}^{\ell+1}$ velocity to be sought should fulfill (48) with the pressure being evaluated at time level $\ell + 1$ (cf. Section 2):

$$\boldsymbol{v}^{\ell+1} = \boldsymbol{v}^{\ell} - \Delta t \boldsymbol{v}^{\ell} \cdot \nabla \boldsymbol{v}^{\ell} - \frac{\Delta t}{\varrho} \nabla p^{\ell+1} + \Delta t \nu \nabla^2 \boldsymbol{v}^{\ell} + \Delta t \boldsymbol{g}^{\ell} \,.$$

Subtracting this equation and the equation for $\boldsymbol{v}^*$ yields an expression for $\boldsymbol{v}^c$:

$$\boldsymbol{v}^c = \boldsymbol{v}^{\ell+1} - \boldsymbol{v}^* = -\frac{\Delta t}{\varrho} \nabla (p^{\ell+1} - \beta p^{\ell}) \,.$$

That is,

$$\boldsymbol{v}^{\ell+1} = \boldsymbol{v}^* - \frac{\Delta t}{\varrho} \nabla (p^{\ell+1} - \beta p^{\ell})$$

We must require $\nabla \cdot \boldsymbol{v}^{\ell+1} = 0$ and this leads to a Poisson equation for the pressure difference $\phi \equiv p^{\ell+1} - \beta p^{\ell}$:

$$\nabla^2 \phi = \frac{\varrho}{\Delta t} \nabla \cdot \boldsymbol{v}^* \,. \tag{50}$$

After having computed ϕ from this equation, we can update the pressure and the velocity:

$$p^{\ell+1} = \beta p^{\ell} + \phi, \tag{51}$$

$$\boldsymbol{v}^{\ell+1} = \boldsymbol{v}^* - \frac{\Delta t}{\varrho} \nabla \phi \,. \tag{52}$$

An open question is how to assign suitable boundary conditions to ϕ; the function, its normal derivative, or a combination of the two must be known at the complete boundary since ϕ fulfills a Poisson equation. On the other hand, the pressure only needs to be specified (as a function of time) at a single point

in space, when solving the original problem (1)–(2). There are two ways of obtaining the boundary conditions. One possibility is to compute $\partial p/\partial n$ from (1), just multiply by the unit normal vector at the boundary. From these expressions one can set up $\partial\phi/\partial n$. The second way of obtaining the boundary conditions is derived from (52); if $\boldsymbol{v}^{\ell+1}$ is supposed to fulfill the Dirichlet boundary conditions then

$$\nabla\phi|_{\partial\Omega} = \frac{\Delta t}{\varrho}(\boldsymbol{v}^{\ell+1} - \boldsymbol{v}^*)|_{\partial\Omega} = 0, \tag{53}$$

since $\boldsymbol{v}^*$ already has the proper boundary conditions. This relation is valid on all parts of the boundary where the velocity is prescribed. Because ϕ is the solution of (50), $\partial\phi/\partial n$ can be controlled, but these homogeneous boundary conditions are in conflict with the ones derived from (1) and (52) [61]. We see that the boundary conditions can be derived in different ways, and the surprising result is that one arrives at different conditions. Additionally we see that after the update (52) we are no longer in control of the tangential part of the velocity at the boundary. The problem with assigning proper boundary conditions for the pressure may result in a large error for the pressure near the boundary. Often one experiences an $\mathcal{O}(1)$ error in a boundary layer with width $\approx \sqrt{\Delta t\nu}$. This error can often be removed by extrapolating pressure values from the interior domain to the boundary. We refer to Gresho and Sani [27] for a thorough discussion of boundary conditions for the pressure Poisson equation.

The basic operator splitting algorithm can be summarized as follows.

1. Compute the prediction $\boldsymbol{v}^*$ from the explicit equation (49).
2. Compute ϕ from the Poisson equation (50).
3. Compute the new velocity $\boldsymbol{v}^{\ell+1}$ and pressure $p^{\ell+1}$ from the explicit equations (51)–(52).

Note that all steps are trivial numerical operations, except for the need to solve the Poisson equation, but this is a much simpler equation than the original problem (1)–(2).

5.2 Implicit Velocity Step

Operator splittings based on implicit difference schemes in time are more robust and stable than the explicit strategy just outlined. To illustrate how more implicit schemes can be constructed, we can take a backward step in (1) to obtain a predicted velocity $\boldsymbol{v}^*$:

$$\boldsymbol{v}^* + \Delta t\boldsymbol{v}^*\cdot\nabla\boldsymbol{v}^* + \Delta t\frac{\beta}{\varrho}\nabla p^\ell - \Delta t\nu\nabla^2\boldsymbol{v}^* + \Delta t\boldsymbol{g}^{\ell+1} = \boldsymbol{v}^\ell\,. \tag{54}$$

Alternatively, we could use the more flexible θ-rule in time (see below). Equation (54) is nonlinear, and a simple linearization strategy is to use $\boldsymbol{v}^\ell\cdot\nabla\boldsymbol{v}^*$ instead of $\boldsymbol{v}^*\cdot\nabla\boldsymbol{v}^*$,

$$\boldsymbol{v}^* + \Delta t\boldsymbol{v}^\ell\cdot\nabla\boldsymbol{v}^* + \Delta t\frac{\beta}{\varrho}\nabla p^\ell - \Delta t\nu\nabla^2\boldsymbol{v}^* + \Delta t\boldsymbol{g}^{\ell+1} = \boldsymbol{v}^\ell\,. \tag{55}$$

Also in the case we keep the nonlinearity, most linearization methods end up with solving a sequence of convection-diffusion equations like (55). The $\boldsymbol{v}^{\ell+1}$ velocity is supposed to fulfill

$$\boldsymbol{v}^{\ell+1} + \Delta t \boldsymbol{v}^{\ell} \cdot \nabla \boldsymbol{v}^{\ell+1} + \frac{\Delta t}{\varrho} \nabla p^{\ell+1} - \Delta t \nu \nabla^2 \boldsymbol{v}^{\ell+1} + \Delta t \boldsymbol{g}^{\ell+1} = \boldsymbol{v}^{\ell} \,.$$

The correction $\boldsymbol{v}^c$ is now $\boldsymbol{v}^{\ell+1} - \boldsymbol{v}^*$, i.e.,

$$\boldsymbol{v}^c = \boldsymbol{s}(\boldsymbol{v}^c) + \frac{\Delta t}{\varrho} \nabla \phi, \quad \boldsymbol{s}(\boldsymbol{v}^c) = \Delta t(-\boldsymbol{v}^{\ell} \cdot \nabla \boldsymbol{v}^c + \nu \nabla^2 \boldsymbol{v}^c) \,. \tag{56}$$

Note that so far we have not done anything "illegal", and this system can be written as a mixed system,

$$\boldsymbol{v}^c - \boldsymbol{s}(\boldsymbol{v}^c) + \frac{\Delta t}{\varrho} \nabla \phi = 0, \tag{57}$$

$$\nabla \cdot \boldsymbol{v}^c = \nabla \cdot \boldsymbol{v}^* \,. \tag{58}$$

It is then common to neglect or simplify $\boldsymbol{s}$, such that the problem changes into a mixed formulation of the Poisson equation,

$$\boldsymbol{v}^c - \frac{\Delta t}{\varrho} \nabla \phi = 0, \tag{59}$$

$$\nabla \cdot \boldsymbol{v}^c = \nabla \cdot \boldsymbol{v}^* \,. \tag{60}$$

Elimination of $\boldsymbol{v}^c$ yields a Poisson equation like (50),

$$\nabla^2 \phi = \frac{\varrho}{\Delta t} \nabla \cdot \boldsymbol{v}^* \,. \tag{61}$$

The problems at the boundary that were discussed in the previous section apply to this method as well. Different choices of and approximations to $\boldsymbol{s}$ give rise to different methods. We shall come back to this point later when discretizing in space prior to splitting the original equations.

To summarize, the sketched implicit operator splitting method consists of solving an advection-diffusion equation (55), a Poisson equation (61), and then performing two explicit updates (we assume that $\boldsymbol{s}$ is neglected):

$$\boldsymbol{v}^{\ell+1} = \boldsymbol{v}^* - \nabla \frac{\Delta t}{\varrho} \phi, \tag{62}$$

$$p^{\ell+1} = \beta p^{\ell} + \phi \,. \tag{63}$$

The outlined operator splitting approaches reduce the Navier–Stokes equations to a system of standard equations (explicit updates, linear convection-diffusion equations, and Poisson equations). These equations can be discretized by standard finite elements, that is, there is seemingly no need for mixed finite elements, a fact that simplifies the implementation of viscous flow simulators significantly.

5.3 A More Accurate Projection Method

In Brown *et al.* [10] an attempt to remove the boundary layer introduced in the pressure by the projection method discussed above is described, cf. also [17, 52]. Previously, in Sections 5.1 and 5.2, we neglected the term $\boldsymbol{s}(\boldsymbol{v}^c)$, since the scheme was only first order in time. This resulted in a problem with the boundary conditions on ϕ. If the scheme is second-order in time, we can not remove this term. In [10] a second-order scheme in *velocity and pressure* is described. In addition, many previous attempts to construct second-order methods for the incompressible Navier-Stokes equations are reviewed there. In order to describe the approach let us start to form a centered scheme at time level $\ell+1/2$ for the momentum equation:

$$\begin{aligned} \frac{\boldsymbol{v}^{\ell+1}-\boldsymbol{v}^{\ell}}{\Delta t}+\nabla p^{\ell+1/2} &= -[\boldsymbol{v}\cdot\nabla\boldsymbol{v}]^{\ell+1/2}+\frac{\nu}{2}\nabla^2(\boldsymbol{v}^{\ell+1}+\boldsymbol{v}^{\ell})+\boldsymbol{g}^{\ell+1/2}, \quad (64)\\ \nabla\cdot\boldsymbol{v}^{\ell+1} &= 0\,. \quad (65) \end{aligned}$$

Here the approximation $[\boldsymbol{v}\cdot\nabla\boldsymbol{v}]^{\ell+1/2}$ is assumed to be extrapolated from the solution on previous time levels. A predicted velocity is computed by

$$\frac{\boldsymbol{v}^{*,\ell+1}-\boldsymbol{v}^{*,\ell}}{\Delta t}=-[\boldsymbol{v}\cdot\nabla\boldsymbol{v}]^{\ell+1/2}+\frac{\nu}{2}\nabla^2(\boldsymbol{v}^{*,\ell+1}+\boldsymbol{v}^{*,\ell})+\boldsymbol{g}^{\ell+1/2}\,. \quad (66)$$

Note that, since we now use $\boldsymbol{v}^{*,\ell}$ instead of $\boldsymbol{v}^{\ell}$ as the initial solution at level ℓ, $\boldsymbol{v}^*$ follows its own evolution equation. The initial conditons $\boldsymbol{v}^{*,0}=\boldsymbol{v}^0$ should be used. Subtracting (66) from (64) we obtain an equation for the velocity correction, $\boldsymbol{v}^{c,\ell+1}=\boldsymbol{v}^{\ell+1}-\boldsymbol{v}^{*,\ell+1}$,

$$\frac{\boldsymbol{v}^{c,\ell+1}-\boldsymbol{v}^{c,\ell}}{\Delta t}+\nabla p^{\ell+1/2}=\frac{\nu}{2}\nabla^2(\boldsymbol{v}^{c,\ell+1}+\boldsymbol{v}^{c,\ell})\,. \quad (67)$$

Note that this is a diffusion equation for $\boldsymbol{v}^c$ with a gradient, $-\nabla p$, as a forcing term. If we assume that this implies that $\boldsymbol{v}^c$ is itself a gradient we can conclude that

$$\boldsymbol{v}^{\ell+1}-\boldsymbol{v}^{*,\ell+1}=\boldsymbol{v}^{c,\ell+1}=\nabla\phi^{\ell+1}, \quad (68)$$

for a suitable function $\phi^{\ell+1}$. From $\nabla\cdot\boldsymbol{v}^{\ell+1}=0$ we get

$$-\nabla^2\phi^{\ell+1}=-\nabla\cdot\boldsymbol{v}^{*,\ell+1}\,. \quad (69)$$

To solve this equation, it remains to assign proper boundary conditions to $\phi^{\ell+1}$. From the discussion in Section 5.1 we know that the boundary conditions on ϕ can be determined such that $\boldsymbol{v}^{l+1}$ fullfill the normal components (or one tangential component), i.e.

$$\frac{\partial\phi}{\partial\boldsymbol{n}}|_{\partial\Omega}=\boldsymbol{n}\cdot(\boldsymbol{v}^{*,\ell+1}-\boldsymbol{v}^{l+1})|_{\partial\Omega}=0\,. \quad (70)$$

We have now fixed the normal components of the boundary conditions on $\boldsymbol{v}^{*,\ell+1}$, but we have lost control over the tangential part. In [10] they therefore propose

to use an extrapolated value for $\hat{\phi}^{\ell+1}$ to determined the tangential parts of $\boldsymbol{v}^*$ such that

$$\boldsymbol{t} \cdot \boldsymbol{v}^{*,\ell+1}|_{\partial\Omega} = \boldsymbol{t} \cdot (\boldsymbol{v}^{\ell+1} + \nabla\hat{\phi}^{\ell+1})|_{\partial\Omega}, \tag{71}$$

where $\boldsymbol{t}$ is a tangent vector (in 3D both tangent vectors must be used).

A the relation between p and ϕ is computed by inserting (68) into (67) to get the pressure update,

$$p^{\ell+1} = \frac{\phi^{\ell+1} - \phi^{\ell}}{\Delta t} - \frac{\nu}{2}\nabla^2(\phi^{\ell+1} + \phi^{\ell}) \,. \tag{72}$$

To summerize this approach a complete time step consists of

1. Evolve $\boldsymbol{v}^*$ by (66) and the boundary conditions given by (70) and (71).
2. Solve (69) for $\phi^{\ell+1}$ using the boundary condition (70).
3. Compute $\boldsymbol{v}^{\ell+1}$ and $p^{\ell+1}$ using (68) and (72).

We refer to Brown *et al.* [10] for more details. A critical and nonobivious step, seems to be the correctness of the derivation of (68) from (67). This may depend on the given boundary conditions.

5.4 Relation to Stabilization Techniques

The operator splitting techniques in time, as explained in (5.1) and (5.2), seem to work quite well in spite of their simplicity compared to the original coupled system (1)–(2). Some explanation of why the method works can be found in [62, 63, 73, 74]. The point is that one can show that the operator splitting method from Section (5.2) is equivalent to solving a system like (1)–(2) with an old pressure in (1) and a stabilization term $\Delta t\nabla^2 p$ on the right-hand side of (2). This stabilization term makes it possible to use standard elements and grids. Other suggested operator splitting methods [63] can be interpreted as a $\Delta t\partial p/\partial t$ stabilization term in the equation of continuity, i.e., a method closely related to the artificial compressibility scheme from Section 4.3.

5.5 Fractional Step Methods

Fractional step methods constitute another class of popular strategies for splitting the Navier–Stokes equations. A typical fractional step approach [2, 4, 10, 20, 87] may start with a time discretization where the convective term is treated explicitly, whereas the pressure and the viscosity term are treated implicitly:

$$\begin{aligned} \boldsymbol{v}^{\ell+1} - \boldsymbol{v}^{\ell} + \Delta t\boldsymbol{v}^{\ell} \cdot \nabla\boldsymbol{v}^{\ell} &= -\Delta t\frac{\beta}{\varrho}\nabla p^{\ell+1} + \Delta t\nu\nabla^2\boldsymbol{v}^{\ell+1} + \Delta t\boldsymbol{g}^{\ell+1}, & (73) \\ \nabla \cdot \boldsymbol{v}^{\ell+1} &= 0 \,. & (74) \end{aligned}$$

One possible splitting of (73)–(74) is now

$$\boldsymbol{v}^* - \boldsymbol{v}^\ell + \Delta t \boldsymbol{v}^\ell \cdot \nabla \boldsymbol{v}^\ell = 0, \tag{75}$$

$$\boldsymbol{v}^{**} = \boldsymbol{v}^* + \Delta t \nu \nabla^2 \boldsymbol{v}^{**} + \Delta t \boldsymbol{g}^{\ell+1}, \tag{76}$$

$$\boldsymbol{v}^{\ell+1} = \boldsymbol{v}^{**} - \frac{\Delta t}{\varrho} \nabla p^{\ell+1}, \tag{77}$$

$$\nabla \cdot \boldsymbol{v}^{\ell+1} = 0\,. \tag{78}$$

Notice that combining (75)–(77) yields (73). Equation (75) is a pure advection equation and can be solved by appropriate explicit methods for hyperbolic problems. Equation (76) is a standard heat conduction equation, with implicit time differencing. Finally, (77)–(78) is a mixed Poisson problem, which can be solved by special methods for mixed Poisson problems, or one can insert $\boldsymbol{v}^{\ell+1}$ from (77) into (78) to obtain a pressure Poisson equation,

$$\nabla^2 p^{\ell+1} = \frac{\varrho}{\Delta t} \nabla \cdot \boldsymbol{v}^{**}\,. \tag{79}$$

After having solved this equation for $p^{\ell+1}$, (77) is used to find the velocity $\boldsymbol{v}^{\ell+1}$ at the new time level. Using (79) and then (77) instead of solving (77)–(78) simultaneously has the advantage of avoiding staggered grids or mixed finite elements. However, (79) requires extra pressure boundary conditions at the whole boundary as discussed previously.

The fractional step methods offer flexibility in the splitting of the Navier–Stokes equations into equations that are significantly simpler to work with. For example, in the presented scheme, one can apply specialized methods to treat the $\boldsymbol{v}\cdot\nabla\boldsymbol{v}$ term because this term is now isolated in a Burgers equation (75) for which numerous accurate and efficient explicit solution methods exist. The implicit time stepping in the scheme is isolated in a standard heat or diffusion equation (76) whose solution can be obtained very efficiently. The last equation (79) is also a simple equation with a wealth of efficient solution methods. Although each of the equations can be solved with good control of efficiency, stability, and accuracy, it is an open question of how well the overall, compound solution algorithm behaves. *This is the downside of all operator splitting methods, and therefore these methods must be used with care.*

More accurate (second-order in Δt) fractional step schemes than outlined here can be constructed, see Glowinski and Dean [16] for a framework and Brown *et al.* [10] for review.

5.6 Discretizing in Space Prior to Discretizing in Time

The numerical strategies in Sections 5.1–5.4 are based on discretizing (1)–(2) first in time, to get a set of simpler partial differential equations, and then discretizing the time-discrete equations in space. One fundamental difficulty with the this approach is that we derive a second-order Poisson equation for the pressure itself or a pressure increment. Such a Poisson equation implies a demand for more boundary conditions for p than what is required in the original system

(1)–(2), as discussed in the previous section. The cause of these problematic, and unnatural, boundary conditions on the pressure is the simplification of the system (57)-(58) to (59)-(60), where the term $\boldsymbol{s}(\boldsymbol{v}^c)$, containing $\Delta t\nu\nabla^2\boldsymbol{v}^c$, is neglected. If we keep this term the system (57)-(58) is replaced by

$$(1 - \Delta t\nu\nabla^2)\boldsymbol{v}^c - \frac{\Delta t}{\varrho}\nabla\phi = 0, \tag{80}$$

$$\nabla \cdot \boldsymbol{v}^c = \nabla \cdot \boldsymbol{v}^* . \tag{81}$$

This system is a modified stationary Stokes system, which can be solved under the correct boundary conditions on the velocity field $\boldsymbol{v}^c$. However, this system can not easily be reduced to a simple Poisson equation for the pressure increment ϕ. Instead, we have to solve the complete coupled system in $\boldsymbol{v}^c$ and ϕ, and when this system is discretized we obtain algebraic systems of the form (9). Hence, the implementation of the correct boundary conditions seems to be closely tied to the need to solve discrete saddle point systems of the form (9).

Another attempt to avoid constructing extra consistent boundary conditions for the pressure is to first discretize the original system (1)-(2) in space. Hence, we need to discretize both the dynamic equation and the incompressibility conditions, using discrete approximations of the pressure and the velocity. This will lead to a system of ordinary differential equations with respect to time, with a set of algebraic constraints representing the incompressibility conditions, and with the proper boundary conditions built into the spatial discretization. A time stepping approach, closely related to operator splitting, for such constrained systems is to first facilitate an advancement of the velocity just using the dynamic equation. As a second step we then "project" the velocity onto the space of divergence free velocities. The two steps in this procedure are closely related to the approach discussed in Sections 5.1 and 5.2. For example, equation (55) can be seen as a dynamic step, while (59)-(60), or simply (61), can be seen as the projection step. However, the projection induced by the system (59)-(60) is not compatible with the boundary conditions of the original system (and this may lead to large error in the pressure near the boundary). In contrast, the projection introduced by the system (80)-(81) has the correct boundary conditions.

In order to discuss this approach in greater detail let us apply either a finite element, finite volume, finite difference, or spectral method to discretize the spatial operators in the system (1)–(2). This yields a system of ordinary differential equations, which can be expressed in the following form:

$$\boldsymbol{M}\dot{\boldsymbol{u}} + \boldsymbol{K}(\boldsymbol{u})\boldsymbol{u} = -\boldsymbol{Q}\boldsymbol{p} + \boldsymbol{A}\boldsymbol{u} + \boldsymbol{f} \tag{82}$$

$$\boldsymbol{Q}^T\boldsymbol{u} = \boldsymbol{0}. \tag{83}$$

Here, $\boldsymbol{u}$ is a vector of velocities at the (velocity) grid points, $\boldsymbol{p}$ is a vector of pressure values at the (pressure) grid points, $\boldsymbol{K}$ is a matrix arising from discretizing $\boldsymbol{v} \cdot \nabla$, $\boldsymbol{M}$ is a mass matrix (the identity matrix $\boldsymbol{I}$ in finite difference/volume methods), $\boldsymbol{Q}$ is a discrete gradient operator, $\boldsymbol{Q}^T$ is the transpose of $\boldsymbol{Q}$, representing a discrete divergence operator, and $\boldsymbol{A}$ is a discrete Laplace operator.

The right-hand side $\boldsymbol{f}$ contains body forces. Stable discretizations require mixed finite elements or staggered grids for finite volume and difference methods. Alternatively, one can add stabilization terms to the equations. The extra terms to be added to (82)–(83) are commented upon in Section 5.8.

We can easily devise a simple explicit method for (82) by using the same ideas as in Section 5.1. A tentative or predicted discrete velocity field $\boldsymbol{u}^*$ is computed by

$$\boldsymbol{M}\boldsymbol{u}^* = \boldsymbol{M}\boldsymbol{v}^\ell + \Delta t(-\boldsymbol{K}(\boldsymbol{u}^\ell)\boldsymbol{u}^\ell - \beta\boldsymbol{Q}\boldsymbol{p}^\ell + \boldsymbol{A}\boldsymbol{u}^\ell + \boldsymbol{f}^\ell)\,. \tag{84}$$

A correction $\boldsymbol{u}^c$ is sought such that $\boldsymbol{u}^{\ell+1} = \boldsymbol{u}^* + \boldsymbol{u}^c$ fulfills $\boldsymbol{Q}^T\boldsymbol{u}^{\ell+1} = \boldsymbol{0}$. Subtracting $\boldsymbol{u}^*$ from $\boldsymbol{u}^{\ell+1}$ yields

$$\boldsymbol{u}^c = -\Delta t\boldsymbol{M}^{-1}\boldsymbol{Q}\boldsymbol{\phi}, \quad \boldsymbol{\phi} \equiv \boldsymbol{p}^{\ell+1} - \beta\boldsymbol{p}^\ell\,. \tag{85}$$

Now a projection step onto the constraint $\boldsymbol{Q}^T\boldsymbol{u}^{\ell+1} = \boldsymbol{0}$ results in an equation for $\boldsymbol{\phi}$:

$$\boldsymbol{Q}^T\boldsymbol{M}^{-1}\boldsymbol{Q}\boldsymbol{\phi} = \frac{1}{\Delta t}\boldsymbol{Q}^T\boldsymbol{u}^*\,. \tag{86}$$

This is a *discrete Poisson equation* for the pressure. For example, employing finite difference methods in a spatial staggered grid yields $\boldsymbol{M} = \boldsymbol{I}$ and $\boldsymbol{Q}^T\boldsymbol{Q}$ is then the standard 5- or 7-star discrete Laplace operator. The matrix $\boldsymbol{Q}^T\boldsymbol{M}^{-1}\boldsymbol{Q}$ is a counterpart to matrices arising from ∇^2 in the Poisson equations for ϕ in Sections 5.1 and 5.2.

Having computed $\boldsymbol{\phi}$, the new pressure and velocity values are found from

$$\boldsymbol{p}^{\ell+1} = \beta\boldsymbol{p}^\ell + \boldsymbol{\phi}, \tag{87}$$

$$\boldsymbol{u}^{\ell+1} = \boldsymbol{u}^* - \Delta t\boldsymbol{M}^{-1}\boldsymbol{Q}\boldsymbol{\phi}\,. \tag{88}$$

5.7 Classical Schemes

In this subsection we shall present a common setting for many popular classical schemes for solving the Navier-Stokes equations. We start with formulating an implicit scheme for (82) using the θ-rule for flexibility; $\theta = 1$ gives the Backward Euler scheme, $\theta = 1/2$ results in the trapezoidal rule (or a Crank-Nicolson scheme), and $\theta = 0$ recovers the explicit Forward Euler scheme treated above. The time-discrete equations can be written as

$$\boldsymbol{N}\boldsymbol{u}^{\ell+1} + \Delta t\boldsymbol{Q}\boldsymbol{p}^{\ell+1} = \boldsymbol{q}, \tag{89}$$

$$\boldsymbol{Q}^T\boldsymbol{u}^{\ell+1} = \boldsymbol{0} \tag{90}$$

where

$$\boldsymbol{N} = \boldsymbol{M} + \theta\Delta t\boldsymbol{R}(\boldsymbol{u}^\ell), \tag{91}$$

$$\boldsymbol{R}(\boldsymbol{u}^\ell) = \boldsymbol{K}(\boldsymbol{u}^\ell) - \boldsymbol{A}, \tag{92}$$

$$\boldsymbol{q} = (\boldsymbol{M} - (1-\theta)\Delta t\boldsymbol{R}(\boldsymbol{u}^\ell))\boldsymbol{u}^\ell + \Delta t\boldsymbol{f}^{\ell+1}) \tag{93}$$

are introduced to save space in the equations. Observe that we have linearized the convective term by using $\boldsymbol{R}(\boldsymbol{u}^\ell)$ on the left-hand side of (89). One could, of course, resolve the nonlinearity by some kind of iteration instead.

To proceed, we skip the pressure or use old pressure values in (89) to produce a predicted velocity $\boldsymbol{u}^*$:

$$\boldsymbol{N}\boldsymbol{u}^* = \boldsymbol{q} - \beta\Delta t\boldsymbol{Q}\boldsymbol{p}^\ell\,. \tag{94}$$

The correction $\boldsymbol{u}^c = \boldsymbol{u}^{\ell+1} - \boldsymbol{u}^*$ is now governed by

$$\begin{aligned} \boldsymbol{N}\boldsymbol{u}^c + \Delta t\boldsymbol{Q}\boldsymbol{\phi} &= \boldsymbol{0}, & (95)\\ \boldsymbol{Q}^T\boldsymbol{u}^c &= \boldsymbol{Q}^T\boldsymbol{u}^*, & (96) \end{aligned}$$

The system (95)–(96) for $(\boldsymbol{u}^c, \boldsymbol{\phi})$ corresponds to the system (80)–(81). Eliminating $\boldsymbol{u}^c$ gives

$$\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{Q}\boldsymbol{\phi} = -\frac{1}{\Delta t}\boldsymbol{Q}^T\boldsymbol{u}^*\,. \tag{97}$$

We shall call this equation the *Schur complement pressure equation* [84].

Solving (97) requires inverting $\boldsymbol{N}$, which is not an option since $\boldsymbol{N}^{-1}$ is dense and $\boldsymbol{N}$ is sparse. Several rough approximations $\tilde{\boldsymbol{N}}^{-1}$ to $\boldsymbol{N}^{-1}$ have therefore been proposed. In other words, we solve

$$\boldsymbol{Q}^T\tilde{\boldsymbol{N}}^{-1}\boldsymbol{Q}\boldsymbol{\phi} = -\frac{1}{\Delta t}\boldsymbol{Q}^T\boldsymbol{u}^*\,. \tag{98}$$

The simplest approach is to let $\boldsymbol{N}$ be an approximation to $\boldsymbol{M}$ only, i.e., $\tilde{\boldsymbol{N}} = \boldsymbol{I}$ in finite difference methods and $\tilde{\boldsymbol{N}}$ equal to the lumped mass matrix $\boldsymbol{M}$ in finite element methods. The approximation $\tilde{\boldsymbol{N}} = \boldsymbol{I}$ leaves us with a standard 5- or 7-star Poisson equation. With $\theta = 0$ we recover the simple explicit scheme from the end of Section 5.6, whereas $\theta = 1$ gives an implicit backward scheme of the same nature as the one described in Section 5.2.

To summarize the algorithm at a time level, we first make a prediction $\boldsymbol{u}^*$ from (94), then solve (98) for the pressure increment $\boldsymbol{\phi}$, and then update the velocity and pressure by

$$\boldsymbol{u}^{\ell+1} = \boldsymbol{u}^* - \Delta t\boldsymbol{N}^{-1}\boldsymbol{Q}\boldsymbol{\phi}, \quad \boldsymbol{p}^{\ell+1} = \beta\boldsymbol{p}^\ell + \boldsymbol{\phi}\,. \tag{99}$$

Let us now comment upon classical numerical methods for the Navier-Stokes equations and show how they can be considered as special cases of the algorithm in the previous paragraph. The history of operator splitting methods starts in the mid and late 1960s. Harlow and Welch [31] suggested an algorithm which corresponds to $\beta = 0$ in our set up and centered finite differences on a staggered spatial grid. The Poisson equation for $\boldsymbol{\phi}$ hence becomes an equation for $p^{\ell+1}$ directly. Chorin [14] defined a similar method, still with $\beta = 0$, but using a non-staggered grid. Temam [77] developed more or less the same method independently, but with explicit time stepping. Hirt and Cook [32] introduced $\beta = 1$ in our terminology. All of these early contributions started with spatially

discrete equations and performed the splitting afterwards. The widely used SIMPLE method [58] consists of choosing $\theta \neq 0$

$$\tilde{\boldsymbol{N}} = \text{diag}(\boldsymbol{N}) = \text{diag}(\boldsymbol{M} + \theta \Delta t \boldsymbol{R}(\boldsymbol{u}^\ell))$$

when solving (98). A method very closely related to SIMPLE is the segregated finite element approach, see e.g. Ch. 7.3 in [35].

Most later developments follow either the approach for the current subsection or the alternative view from Sections 5.1 and 5.2. Much of the focus in the history of operator splitting methods has been on constructing second- and higher-order splittings in the framework of Sections 5.1, 5.2, 5.3, and 5.5, see e.g. [2, 4] and [10].

We remark that the step for $\boldsymbol{u}^*$ is unnecessary if we solve the system (95)–(96) for $(\boldsymbol{u}^c, \boldsymbol{\phi})$ correctly, basically we then solve the "exact" system (89)–(90). The point is that we solve (95)–(96) approximately because we replace $\boldsymbol{N}$ in (95) by $\tilde{\boldsymbol{N}}$ when we eliminate $\boldsymbol{u}^c$ to form the pressure equation (98). The next section presents a framework where the classical methods from the current section appear as one iteration in an iterative solution procedure for the fully implicit system (89)–(90).

5.8 Fully Implicit Methods

Rannacher [63] and Turek [84] propose a general framework to analyze the efficiency and robustness of operator splitting methods. We first consider the *fully implicit* system (89)–(90). Eliminating $\boldsymbol{u}^{\ell+1}$ yields (cf. the similar elimination for the Stokes problem in Section 3.2)

$$\boldsymbol{Q}^T \boldsymbol{N}^{-1} \boldsymbol{Q} \boldsymbol{p}^{\ell+1} = \frac{1}{\Delta t} \boldsymbol{Q}^T \boldsymbol{N}^{-1} \boldsymbol{q}, \tag{100}$$

which we can call the *Schur complement pressure equation* for the implicit system. Notice that we obtain the same solution for the pressure in both (100) and (89)–(90). The velocity needs to be computed, after $\boldsymbol{p}^{\ell+1}$ is from (89), which requires an efficient solution of linear systems with $\boldsymbol{N}$ as coefficient matrix (Multigrid is an option here).

Turek [84] suggests that many common solution strategies can be viewed as special cases of a preconditioned Richardson iteration applied to the Schur complement pressure equation (100). Given a linear system

$$\boldsymbol{B} \boldsymbol{p}^{\ell+1} = \boldsymbol{b},$$

the preconditioned Richardson iteration reads

$$\boldsymbol{p}^{\ell+1,k+1} = \boldsymbol{p}^{\ell+1,k} - \boldsymbol{C}^{-1}(\boldsymbol{B}\boldsymbol{p}^{\ell+1,k} - \boldsymbol{b}), \tag{101}$$

where $\boldsymbol{C}^{-1}$ is a preconditioner and k an iteration counter. The iteration at a time level is started with the pressure solution at the previous time level:

$$\boldsymbol{p}^{\ell+1,0} = \boldsymbol{p}^{\ell}\,.$$

Applying this approach to the Schur complement pressure equation (100) gives the recursion

$$\boldsymbol{p}^{\ell+1,k+1} = \boldsymbol{p}^{\ell+1,k} - \boldsymbol{C}^{-1}(\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{Q}\boldsymbol{p}^{\ell+1,k} - \frac{1}{\Delta t}\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{q})\,. \tag{102}$$

We now show that the operator splitting methods from Section 5.7, based on solving $\boldsymbol{u}^*$ from (94), solving (98) for $\boldsymbol{\phi}$, and then updating the velocity and pressure with (99), can be rewritten in the form (102). This allows us to interpret the methods from Section 5.7 in a more general framework and to improve the numerics and generate new schemes.

To show this equivalence, we start with the pressure update (99) and insert (98) and (94) subsequently:

$$\begin{aligned}
\boldsymbol{p}^{\ell+1} &= \boldsymbol{p}^{\ell} + \boldsymbol{\phi} && (103)\\
&= \boldsymbol{p}^{\ell} + (\boldsymbol{Q}^T\tilde{\boldsymbol{N}}^{-1}\boldsymbol{Q})^{-1}\frac{1}{\Delta t}\boldsymbol{Q}^T\boldsymbol{u}^* && (104)\\
&= \boldsymbol{p}^{\ell} + (\boldsymbol{Q}^T\tilde{\boldsymbol{N}}^{-1}\boldsymbol{Q})^{-1}\frac{1}{\Delta t}\boldsymbol{Q}^T(\boldsymbol{N}^{-1}\boldsymbol{q} - \Delta t\boldsymbol{N}^{-1}\boldsymbol{Q}\boldsymbol{p}^{\ell}), && (105)\\
&= \boldsymbol{p}^{\ell} - (\boldsymbol{Q}^T\tilde{\boldsymbol{N}}^{-1}\boldsymbol{Q})^{-1}(\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{Q}\boldsymbol{p}^{\ell} - \frac{1}{\Delta t}\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{q})\,. && (106)
\end{aligned}$$

We have assumed that $\beta = 1$, since this is the case in the Richardson iteration. Equation (106) can be generalized to an iteration on $\boldsymbol{p}^{\ell+1}$:

$$\boldsymbol{p}^{\ell+1,k+1} = \boldsymbol{p}^{\ell+1,k} - (\boldsymbol{Q}^T\tilde{\boldsymbol{N}}^{-1}\boldsymbol{Q})^{-1}(\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{Q}\boldsymbol{p}^{\ell+1,k} - \frac{1}{\Delta t}\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{q})\,. \tag{107}$$

We see that (107) is constistent with (106) for the first iteration $k = 1$ if $\boldsymbol{p}^{\ell+1,0} = \boldsymbol{p}^{\ell}$. Moreover, we notice that (107) is identical to (102), *provided we choose the preconditioner* $\boldsymbol{C}$ as $\boldsymbol{Q}^T\tilde{\boldsymbol{N}}\boldsymbol{Q}$. In other words, classical operator splitting methods from Section 5.7 can be viewed as one preconditioned Richardson iteration on the fully implicit system (89)–(90) (though formulated as (100)). If we perform more iterations in (107), we essentially have an Uzawa algorithm for the original fully implicit system (89)–(90). Using more than one iteration corresponds to iterating on the pressure in (94), i.e., we solve (94) and (98) more than once at each time level, using the most recent pressure approximation to $\boldsymbol{p}^{\ell+1}$ in (94).

The various choices of $\tilde{\boldsymbol{N}}$ outlined in Section 5.7 resemble various classical operator splitting methods when used in the preconditioner $\boldsymbol{C} = \boldsymbol{Q}^T\tilde{\boldsymbol{N}}\boldsymbol{Q}$ in the framework (107). This includes explicit and implicit projection methods, pressure correction methods, SIMPLE variants, and also Uzawa methods and the Vanka smoother used in Multigrid methods [83].

With the classical operator splitting methods reformulated as one iteration of an iterative method for the original fully implicit system, one can more easily attack the fully implicit system directly; the building blocks needed are fast solvers for $\boldsymbol{N}^{-1}$ and $(\boldsymbol{Q}^T\tilde{\boldsymbol{N}}\boldsymbol{Q})^{-1}$, but these are already available in software for the classical methods. In this set up, solving the fully implicit mixed system

(89)–(90) is from an implementational point of view not more complicated than using a classical operator splitting method from Section 5.7 method repeatedly. This is worth noticing because people tend to implement and use the classical methods because of their numerical simplicity compared with the fully implicit mixed system.

Noticing that (107) is an Uzawa method, we could introduce inexact Uzawa methods [18], where $\boldsymbol{N}^{-1}$ is replaced by $\tilde{\boldsymbol{N}}^{-1}$ in the right-hand side of (107) (with some additional terms for making (107) consistent with the original linear system). This can represent significant computational savings. One view of such an approach is that one speeds up solving the predictior step (94) when we iterate over the predictor-correction equations in the classical methods.

The system (82)–(83) can easily be augmented with stabilitzation terms, resulting in a modification of the system (89)–(90):

$$\begin{aligned} \boldsymbol{N}\boldsymbol{u}^{\ell+1} + \Delta t \boldsymbol{Q}\boldsymbol{p}^{\ell+1} &= \boldsymbol{q}, & (108)\\ \boldsymbol{Q}^T\boldsymbol{u}^{\ell+1} - \epsilon \boldsymbol{D}\boldsymbol{p} &= -\epsilon \boldsymbol{d}\,. & (109) \end{aligned}$$

Eliminating $\boldsymbol{u}^{\ell+1}$ yields

$$(\Delta t \boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{Q} + \epsilon \boldsymbol{D})\boldsymbol{p}^{\ell+1} = \boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{q} + \epsilon \boldsymbol{d}\,. \tag{110}$$

With this stabilization one can avoid mixed finite elements or staggered finite difference/volume grids.

Let us now discuss how the preconditioner $\boldsymbol{C}$ can be chosen more generally. If we define the error in iteration k as $\boldsymbol{e}^k = p^{\ell+1,k} - p^{\ell+1}$, one can easily show that $\boldsymbol{e}^k = (\boldsymbol{I} - \boldsymbol{C}\boldsymbol{B})\boldsymbol{e}^{k-1}$. One central question is if just one iteration is enough and under what conditions the iteration is convergent. The latter property is fulfilled if the modulus of the eigenvalues of the amplification matrix $\boldsymbol{I} - \boldsymbol{C}\boldsymbol{B}$ are less than unity. Hence, the choice of $\boldsymbol{C}$ is important both with respect to efficiency and robustness of the solution method.

Turek proposes an efficient preconditioner C^{-1} for (101):

$$\boldsymbol{C}^{-1} = \alpha_R \boldsymbol{B}_R^{-1} + \alpha_D \boldsymbol{B}_D^{-1} + \alpha_K \boldsymbol{B}_K^{-1}, \tag{111}$$

where

- $\boldsymbol{B}_R$ is an optimal (reactive) preconditioner for $\boldsymbol{Q}^T\boldsymbol{M}\boldsymbol{Q}$,
- $\boldsymbol{B}_D$ is an optimal (diffusive) preconditioner for $\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q}$,
- $\boldsymbol{B}_K$ is an optimal (convective) preconditioner for $\boldsymbol{Q}^T\boldsymbol{K}\boldsymbol{Q}$,

Both $\boldsymbol{B}_R$ and $\boldsymbol{B}_D$ can be constructed optimally by standard methods; $\boldsymbol{B}_R$ can be constructed by a Multigrid sweep on a Poisson-type equation, and $\boldsymbol{B}_D$ can be made simply by an inversion of a lumped mass matrix. However, no optimal preconditioner is known for $\boldsymbol{B}_K$. It is assumed that $\epsilon\boldsymbol{D}$ does not change the condition number significantly and it is therefore not considered in the preconditioner.

It is also possible to improve the convergence and in particular the robustness by utilizing other iterative methods than the Richardson iteration, which is the simplest iterative method of all. One particular attractive class of methods is the Krylov (or Conjugate Gradient-like) methods. Methods like GMRES are in principle always convergent, but the convergence is highly dependent on the condition number of $\boldsymbol{CB}$. The authors are pursuing these matters for future research. If approximate methods (Multigrid or Krylov solvers) are used for $\boldsymbol{N}^{-1}$ too, we have a nested iteration, and for the outer Krylov method to behave as efficiently as expected, it is necessary to solve the inner iterations accurately. Inexact Uzawa methods would hence be attractive since they only involve $\tilde{\boldsymbol{N}}^{-1}$ in the inner iteration.

There is also a link between operator splitting methods and fully implicit methods without going through the Schur complement pressure equation. In [70] they considered preconditioners for the Stokes problem of the form

$$\begin{bmatrix} \boldsymbol{N} & \boldsymbol{Q} \\ \boldsymbol{Q}^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{0} \end{bmatrix}. \tag{112}$$

and this lead to preconditioners like

$$\begin{bmatrix} \boldsymbol{C}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C}_2 \end{bmatrix}, \tag{113}$$

where $\boldsymbol{C}_1$ should be an approximation of the inverse of $\boldsymbol{N}$ and $\boldsymbol{C}_2$ of $\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{Q}$. This "operator splitting" preconditioner was proved to be optimal provided that $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ were optimal preconditioners for $\boldsymbol{N}$ and $\boldsymbol{Q}^T\boldsymbol{N}^{-1}\boldsymbol{Q}$, respectively. This preconditioner has been extended to the time-dependent fully coupled Stokes problem in [7] and a preconditioner similar to the one proposed by Turek in (111) is considered in [56].

6 Parallel Computing Issues

Laminar flow computations in simple geometries, involving a few thousand unknowns, can now be carried out routinely on PCs or workstations. Nevertheless, more demanding flows met in industrial and scientific applications easily require a grid resolution corresponding to $10^5 - 10^9$ unknowns and hence the use of parallel computers. The suitability of the methods for solving the Navier-Stokes equations on parallel computers is therefore of importance.

The operator splitting methods from Section 5 reduce the Navier-Stokes equations to a sequence of simpler problems. For example, the explicit scheme from Section 5.1 involves explicit updates, in the form of matrix-vector products and vector additions, and the solution of a Poisson equation. Vector additions are trivial to parallelize, and the parallel treatment of products of sparse matrices and vectors is well known. Several methods have been shown to be successful for parallel solution of the Poisson equation, but Multigrid seems to be particularly attractive as the method is the most effecient solution approach for the

Poisson equation on sequential computers and its concurrent versions are well developed and can be realized with close to optimal speed-up. Using Multigrid as a preconditioner for a Conjugate Gradient method does not alter this picture, since the outer Conjugate Gradient method just involves inner products of vectors, vector additions, and matrix-vector products. In conclusion, the classical explicit operator splitting methods are very well suited for parallel computing.

The implicit operator splitting methods require, in addition to explicit updates and the solution of a Poisson equation, also the solution of a convection-diffusion equation. Domain Decomposition methods at the partial differential equation level can split the global convection-diffusion equation into a series of smaller convection-diffusion problems on subdomains, which can be solved in parallel. The efficiency of this method depends on the convergence rate of the iteration, which shows dependence on the nature of the convection. To achieve sufficient efficiency of the iteration, the Domain Decomposition approach must be combined with a coarse grid correction [75]. Alternatively, the convection-diffusion equation can be viewed as a linear system, with non-symmetric coefficient matrix, to be solved in parallel. Using a Conjugate Gradient-like method, such as GMRES og BiCGStab, combined with Multigrid as preconditioner, yields a solution method whose parallel version is well established and can be realized with close to optimal speed-up. Other popular preconditioners may be more challenging to parallelize well; incomplete factorizations fall in this category. The overall performance of the parallel convection-diffusion solver depends on choices of numerical degrees of freedom in the linear solver, but these difficulties are present also in the sequential version of the method. To summarize, the implicit operator splitting methods are well suited for parallel computers if a good Multigrid or Domain Decomposition preconditioner can be found for the corresponding sequential problem.

The classical fully implicit method for the Navier-Stokes equations normally applies variants of Gaussian elimiation as linear solver (after a linearization of the nonlinear system of algebraic equations by, e.g., some Newton-like method). Parallel versions of sparse and banded Gaussian elimination procedures are being developed, but such methods are much more difficult to implement efficiently on parallel computers than the iterative solvers discussed above.

Solving the fully implicit system for the Navier-Stokes equations by iterative strategies (again after a linearization of the nonlinear system), basically means running an iterative method, like Richardson iteration or a Krylov solver, combined with a suitable preconditioner. In the case we choose the preconditioner to be typical steps in operator splitting methods, as described in Section 5.8, fully implicit methods parallelize with the same efficiency as the corresponding implicit operator splitting methods.

The pressure stabilization technique from Section 4.1 is actually a way of formulating the equation of continuity and get rid of the mixed finite element or staggered grid requirement. This approach is typically used in combination with operator splitting or fully implicit methods, and the extra stabilization terms do not change the parallelization of those methods.

Penalty methods lead to a kind of transient, nonlinear equation of elastic-

ity. After discretizing in time and resolving the nonlinearity, we are left with a partial differential equation or linear system of the same nature as the equation of elasticity. The problem, however, is that the Lame constant λ in this equation is large, which makes it hard to construct efficient *iterative* methods. Large-scale computing with penalty methods is relevant only if efficient iterative methods for the sequential problem can be constructed. When these methods are based on Domain Decomposition and/or involve vector operations, matrix-vector products, and Multigrid building blocks, parallelization is feasible, see Reddy et al. [67] for a promising approach.

The classical artificial compressibility method from Section 4.3 is a purely explicit method, just containing explicit updates, and is hence trivial to parallelize well. Also when implicit time discretizations are used, we get matrix systems that can, in principle, be solved in a parallel fashion using the same methods as we described for the implicit operator splitting approach.

7 Software

Development of robust CFD software is a complicated and time consuming task. Most scientists and engineers who need to solve incompressible viscous fluid flow problems must therefore rely on available software packages. An overview of CFD software is available on the Internet [37], and here we shall just mention a few widely distributed packages.

PHOENICS [50], which appeared in 1981, was the first general-purpose tool for CFD and is now probably the most widely used CFD software. The numerical foundation is the finite volume method in the notation of Patankar [58]. The user can add code and solve equations that are not supported by the package. FLUENT [46] is another general-purpose CFD package that addresses laminar and turbulent incompressible flow, also in combination with combustion and multiple phases. The spatial discretization employs a finite volume technique applicable to unstructured meshes. FIDAP [43] is a general-purpose fluid flow package quite similar to FLUENT, but it applies finite elements for the spatial discretization. CFX [39] is another modern, general-purpose package addressing complex flow problems in the process and chemical industries, including turbulence, multi-phase flow, bubble-driven flow, combustion, flames, and so on. Flow-3D [45] offers special techniques for and specializes in incompressible viscous free surface flow, but the package can be used for more standard external and confined flows as well. Another general-purpose CFD package is CFD2000 [38], which uses finite volumes on curvelinear grids and handles incompressible as well as compressible flows, with turbulence, heat transfer, multiple phases, and chemical reactions. CFD++ is a finite volume-based solver with particular emphasis on turbulent flow. It handles many different types of grids and flow regimes (from incompressible to hypersonic). ANSYS/FLOTRAN [44] is a CFD package contained in the ANSYS family of finite element codes. The tight integration with other ANSYS codes for heat transfer, elasticity, and electromagnetism makes it feasible to perform multi-physics simulations, e.g., fluid-

structure interactions and micro-electro-mechanical systems (MEMS). The CFD software mentioned so far covers advanced, commercial, general-purpose tools that offer a complete problem solving environment, with user-friendly grid generation and visualization facilities in addition to the numerical engines.

FEATFLOW [42] is a free package implementing the framework from Section 5.8, with finite elements in space, Multigrid for solving linear and nonlinear systems, and an emphasis on computational efficiency and robustness. FEMLAB is a package built on top of Matlab and offers easy set-up of incompressible flow problems, also coupled with heat transfer, electromagnetism, and elasticity. Fastflo [41] is a code of a similar nature. Mouse [48] is a modern finite volume-based software library, packed with ready-made CFD programs. Some flexible, general-purpose, *programmable* environments and libraries with important applications to CFD are Diffpack [40], FOAM [47], Overture[49], and UG [51].

The quality and robustness of the numerics even in advanced packages, especially when simulating complex multi-fluid flows, may be questionable as we know that our understanding of how to solve the building block (1)–(2) is still limited. More benchmark problem initiatives [84] are needed to classify flow cases and methods whose numerical results are reliable.

The complexity of CFD applications makes a demand for flexibility, where different splitting approaches, different space and time discretizations, different linear and nonlinear system solvers, and different governing equations can be freely combined. Much of the current CFD software, written as large, standalone Fortran 77 programs, lacks this freedom of choice because of an inflexible software design. There is now a growing interest in methodologies for better design of CFD software based on, e.g., object-oriented programming techniques [40, 47, 53, 48, 57, 49, 81, 85].

8 Future Directions of Research

During the last decades there has been tremendous progress in computational fluid dynamics. Even if some of the most basic tools are well understood by now, there are still challenging problems related to numerical methods for incompressible Navier–Stokes equation. In fact, the simplified Stokes problem (11)–(12), especially when (11) is augmented with a time derivative $\partial \boldsymbol{v}/\partial t$, is not fully understood when it comes to stable discretizations [56], efficient solution of the resulting linear systems (in an implicit formulation), as well as *a priori* and *a posteriori* error estimates. It therefore seems fruitful to still address simplified versions of the Navier–Stokes equations to develop and understand numerical methods.

The most popular time stepping methods for the Navier–Stokes equation are not fully implicit. As explained in Section 5.1, an operator splitting strategy can be used to essentially decouple the updating of the velocity and the pressure. However, the choice of boundary conditions for the pressure in these procedures is problematic. The pressure is closely tied to the incompressibility

condition for the velocity, and any decoupling therefore has drawbacks. As in many other fields, engineers prefer to work with robust and stable numerical engines, even at a cost of decreased computational efficiency. This usually implies that fully implicit codes are preferred in industry. Hence, we think that fully implicit schemes, like the ones discussed in Sections 5.2 and 5.8 should be further developed. The study of efficient preconditioners, which are robust with respect to physical and numerical parameters, seems essential. An attractive framework would be to reuse the experience and knowledge built up around (classical) operator splitting schemes as preconditioners for the more robust fully coupled and implicit scheme. For efficiency, Multigrid cycles should be used in such preconditioners, but the optimal combination of Multigrid building blocks (smoothers, cycling strategies etc.) are not yet known for fully implicit formulations. In addition, the use of parallel computational techniques, like Domain Decomposition algorithms, is essential for performing fine scale simulations. As the gap between CPU power and memory access efficiency increases, it might be questionable whether today's numerics are well suited for the coming generation of high-performance computers. This calls for paying closer attention to the interplay between hardware and numerics.

Another topic which will be essential for doing fine scale computations is the development of adaptive algorithms using suitable error estimators. In particular, for simulations in complex geometries such tools will be unavoidable [63, 84]. The ultimate goal is to allow the engineer to specify an error tolerance and let adaptive algorithms and error estimators find the corresponding discretization parameters and solution strategies. This will be an important goal because we cannot expect that sufficient numerical and physical comptenence will be present among all users when CFD becomes a cheap, widely accessible, and apparently simple-to-use tool.

We have already mentioned the particular need for robust Navier–Stokes solvers when such solvers are embedded in complex fluid models, involving multi-species fluids, turbulence, heat transfer, chemical reactions, and coupling to structural deformations. Many of the future scientific water resources applications of incompressible viscous flow will involve such composite models. For example, a better understanding of consolidation of porous media may be based on studying coupled models of Navier–Stokes equations and the equations of elasto-visco-plasticity at the pore level. Geologists are especially interested in cracking mechanisms, where the desire is to simulate deformation, contact, and fracture of sand grains embedded in a viscous fluid. The science of reactive porous media flow has many open questions, and fundamental studies may benefit from simulating pore level multi-phase flow coupled with chemistry models, in particular in the vicinity of rock "walls". Fortunately, the computing technology necessary for realizing such studies is the same that is needed for complex flow problems found in, e.g., the car industry and physiology.

9 Concluding Remarks

We have in this paper presented a basic introduction to some well-known and widely used numerical methods for the incompressible Navier–Stokes equations. The emphasis has been on the fundamental underlying ideas for discretizing the system of partial differential equations. By hiding the details of the spatial discretization (finite elements, finite differences, finite volumes, spectral methods), we hope to better explain the close relationship between different numerical schools, in particular finite differences/volumes and finite elements; most of the schools apply the same fundamental reasoning and arrive in most cases at the same algebraic equations (modulo effects from different treatment of boundary conditions). Naturally, our main emphasis is on advancing the equations in time and especially how to split the equations into more tractable systems. The methods covered herein include artificial compressibility, penalty function formulations, fully implicit schemes, and operator splitting techniques (and their aliases projection methods, fractional step methods, pressure correction strategies).

As an extension of the introduction to operator splitting methods we have proposed a framework, inspired by Rannacher [63] and Turek [84], where many classical and popular methods can be viewed as certain preconditioned iterative methods applied to a robust, fully implicit formulation of the Navier–Stokes equations. This opens up the possibility of a more general view of the classical methods and for constructing an endless series of new solution strategies, all of them trying to solve the fully implicit system iteratively. One such strategy pursued by the authors has been outlined.

Due to page limitations, several important topics had to be left out or only briefly commented upon. Such topics include, among others, detailed information about finite element, finite difference, and finite volume discretization techniques, spectral methods, higher-order temporal schemes, "upwind" techniques for high Reynolds number flow, discretizing boundary conditions, least-squares finite element formulations, vorticity-streamfunction formulations, grid generation techniques, as well as adaptivity methods based on error estimation and control. Relevant textbooks for references and further reading about these subjects are [15, 20, 27, 30, 61, 84, 86, 89].

The lack of a discussion of special methods for high Reynolds number flow, where a careful treatment of the convective term $\boldsymbol{v} \cdot \nabla \boldsymbol{v}$ is important, may give the impression that these flow regimes are beyond scope of the presented solution methods. This is not true; the same basic solution strategies can still be applied, but then in combination special spatial discretization techniques for the convective term (typically upwind differencing or Petrov-Galerkin finite element methods). More sophisticated approaches for highly convective flow apply special space-time integrations, and this may affect the operator splitting strategies. Within a fractional step approach (as in Section 5.5), the convection term is isolated in a Burgers equation, such that the treatment of high Reynolds number flow is then limited to an advanced numerical treatment of a Burgers equation.

No numerical experiments have been presented in this paper. Results of extensive numerical investigations with different solution strategies for the Navier–Stokes equations can be found in textbooks, see [27, 84] in particular. The Virtual Album of Fluid Motion [42, 84] contains an exciting collection of animations for numerous flow configurations.

Finally, we mention that incompressible viscous flow can also be computed by means of lattice gas or lattice Boltzman methods [12]. These methods are statistical in nature and involve collisions of a large number of particles moving in a lattice. The approach is particularly attractive for simulation of complex multi-phase flows where the qualitative flow behavior is in focus. When computational accuracy is important, and only one fluid is flowing in the laminar regime, discretization of the Navier–Stokes equations as outlined in the present paper seems to be the most efficient and robust solution strategy.

Acknowledgements. The authors thank Dr. Torgeir Rusten for numerous fruitful discussions and useful input to the present manuscript.

References

[1] S. Aliabadi and T. Tezduyar. 3D simulation of two-fluid and free-surface flows with the stabilized-finite-element/interface-capturing method. In S. Atluri and P. O'Donoghue, editors, *Modeling and Simulation Based Engineering; Proceedings of International Conference on Computational Engineering Science, Atlanta, Georgia*. 1998.

[2] A. S. Almgren, J. B. Bell, and W. G. Szymczak. A numerical method for the incompressible Navier-Stokes equations based on an approximate projection. *SIAM J. Sci. Comp.*, 17(2):358–369, 1996.

[3] J. D. Anderson. *Computational Fluid Dynamics – The Basics with Applications*. McGraw-Hill, 1995.

[4] J. B. Bell, P. Colella, and H. M. Glaz. A second order projection method for the incompressible Navier-Stokes equations. *J. Comp. Phys.*, 85:257–283, 1989.

[5] M. Bercovier and O. Pironneau. Error estimates for finite element method solution of the Stokes problem in the primitive variables. *Numer. Math.*, pages 211–224, 1979.

[6] M. J. Blunt. Effects of heterogeneity and wetting on relative permeability using pore level modeling. *SPE Journal*, pages 70–87, 1997.

[7] J. H. Bramble and J. E. Pasciak. Iterative techniques for time dependent stokes problems. *Math. Applic*, 1997.

[8] F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Methods*. Springer, 1991.

[9] A. N. Brooks and T. J. R. Hughes. A streamline upwind/Petrov-Galerkin finite element formulation for advection domainated flows with particular emphasis on the incompressible Navier-Stokes equations. *Comput. Meth. Appl. Mech. Engrg.*, pages 199–259, 1982.

[10] D. L. Brown, R. Cortez, and M. L. Minion. Accurate projection methods for the incompressible navier–stokes equations. *J. Comp. Physics*, pages 464–499, 2001.

[11] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral methods in Fluid Dynamics.* Springer Series in Computational Physics. Springer, 1988.

[12] S. Chen and G. D. Doolen. Lattice boltzmann method for fluid flows. *Annual Rev. Fluid Mech.*, pages 329–364, 1998.

[13] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *J. Comp. Phys.*, 2:12–26, 1967.

[14] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22:745–762, 1968.

[15] T. Chung. *Computational Fluid Dynamics.* Cambridge University Press, 2001.

[16] Edward J. Dean and Roland Glowinski. On some finite element methods for the numerical simulation of incompressible viscous flow. In M. D. Gunzburger and R. A. Nicolaides, editors, *Incompressible computational fluid dynamics; Trends and advances.* Cambridge University Press, 1993.

[17] W. E and J.-G. Liu. Finite difference schemes for incompressible flows in the velocity-impulse density formulation. *J. Comput. Phys.*, 1997.

[18] H. C. Elman and G. Golub. Inexact and preconditioned Uzawa algorithms for saddle point problems. *SIAM J. Num. Anal.*, 31:1645–1661, 1994.

[19] M. S. Engelman, R. I. Sani, P. M. Gresho, and M. Bercovier. Consistent vs. reduced integration penalty methods for incompressible media using several old and new elements. *Int. J. Num. Meth. in Fluids*, 2:25–42, 1982.

[20] J. H. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics.* Springer, 1996.

[21] C. A. J. Fletcher. *Computational Techniques for Fluid Dynamics, Vol I and II.* Springer Series in Computational Physics. Springer, 1988.

[22] M. Fortin and R. Glowinski. *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems.* North-Holland, Amsterdam, The Netherlands, Series: Studies in Mathematics and its Applications, Vol 15, 1983.

[23] E. Fukumori and A. Wake. The linkage between penalty function method and second viscosity applied to Navier-Stokes equation. *Computational Mechanics*, 1991.

[24] S. Gignard, S. T. Grilli, and R. Marcer V. Rey. Computation of shoaling and breaking waves in nearshore areas by the coupling of bem and vof methods. In M. Dæhlen and A. Tveito, editors, *Proc. 9th Offshore and Polar Engng. Conf. (ISOPE99), Vol. III*, pages 304–309. 1999.

[25] V. Girault and P. A. Raviart. *Finite Element Methods for Navier-Stokes Equations*. Springer, 1986.

[26] R. Glowinski. *Numerical Methods for Nonlinear Variational Problems*. Springer, 1984.

[27] P. M. Gresho and R. L. Sani. *Incompressible Flow and the Finite Element Method*. Wiley, 1998.

[28] M. Griebel, T. Dornseifer, and T. Neunhoeffer. *Numerical Simulation in Fluid Dynamics: A Practical Introduction*. SIAM, 1997. Also available in German: *Numerische Simulation in der Strömungsmechanik: Eine Praxisorienterte Einfürung*, Vieweg Lehrbuch Scientific Computing, 1995.

[29] M. Gunzburger. *Finite Element Methods for Viscous Incompressible flows*. Academic Press, 1989.

[30] M. D. Gunzburger and R. A. Nicolaides. *Incompressible computational fluid dynamics; Trends and advances*. Cambridge University Press, 1993.

[31] F. H. Harlow and J. E. Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *The physics of fluids*, 8:2182–2189, 1965.

[32] C. W. Hirt and J. L. Cook. Calculating three-dimensional flows around structures and over rough terrain. *J. Comp. Phys.*, 10:324–340, 1972.

[33] L. W. Ho and E. M. Rnquist. Spectral element solution of steady incompressible viscous free-surface flows. *Finite Elements in Analysis and Design*, pages 207–227, 1994.

[34] T. J. R. Hughes, W. K. Liu, and A. Brooks. Finite element analysis of incompressible viscous flows by the penalty function formulation. *J. Comp. Phys.*, 30:1–60, 1979.

[35] Fluent Inc. FIDAP theory manual.
See e.g. *http://fortuitous.ncsa.uiuc.edu/fidapdocumentation/*.

[36] Internet. CFD Online's book page.
http://www.cfd-online.com/Books/.

[37] Internet. CFD online's software overview.
http://www.cfd-online.com/Resources/soft.html.

[38] Internet. CFD2000 software package.
http://www.adaptive-research.com/.

[39] Internet. CFX software package.
http://www.software.aeat.com/cfx/.

[40] Internet. Diffpack software package. *http://www.diffpack.com.*

[41] Internet. Fastflo software package.
http://www.nag.co.uk/simulation/Fastflo/fastflo.html.

[42] Internet. FEATFLOW software package. *http://www.featflow.de.*

[43] Internet. FIDAP software package.
http://www.fluent.com/software/fidap/.

[44] Internet. FLOTRAN software package.
http://www.ansys.com/products/flotran.htm.

[45] Internet. Flow-3D software package. *http://www.flow3d.com.*

[46] Internet. FLUENT software package.
http://www.fluent.com/software/fluent/.

[47] Internet. FOAM software package. *http://www.nabla.co.uk.*

[48] Internet. Mouse software package.
http://fire8.vug.uni-duisburg.de/MOUSE/.

[49] Internet. Overture software package. *http://www.llnl.gov/casc/Overture/.*

[50] Internet. PHOENICS software package.
http://213.210.25.174/phoenics/d_polis/d_info/phover.htm.

[51] Internet. UG software package.
http://cox.iwr.uni-heidelberg.de/ ug/.

[52] J. Kim and P. Moin. Application of a fractional-step method to incompressible navier–stokes equations. *J. Comput. Phys.*, 1985.

[53] H. P. Langtangen. *Computational Partial Differential Equations – Numerical Methods and Diffpack Programming.* Lecture Notes in Computational Science and Engineering. Springer, 1999.

[54] Rainald Löhner. Design of incompressible flow solvers: Practical aspects. In M. D. Gunzburger and R. A. Nicolaides, editors, *Incompressible computational fluid dynamics; Trends and advances.* Cambridge University Press, 1993.

[55] K.-A. Mardal, X.-C. Tai, and R. Winther. Robust finite elements for Darcy-Stokes flow. *(Submitted)*, 2001.

[56] K.-A. Mardal and R. Winther. Uniform preconditioners for the time dependent Stokes problem. *(In preparation)*, 2002.

[57] O. Munthe and H. P. Langtangen. Finite elements and object-oriented implementation techniques in computational fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, 190:865–888, 2000.

[58] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow.* Series in computational methods in mechanics and thermal sciences. McGraw-Hill, 1980.

[59] R. Peyret and T. D. Taylor. *Computational Methods for Fluid Flow.* Springer Series in Computational Physics. Spriver, 1983.

[60] O. Pironneau. *The Finite Element Methods for Fluids.* John Wiley & Sons, 1989.

[61] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations.* Springer Series in Computational Mathematics. Springer, 1994.

[62] R. Rannacher. On chorin's projection method for incompressible Navier-Stokes equations. In *Proc. Oberwolfach Conf. Navier-Stokes Equations: Theory and Numerical Methods*, 199.

[63] R. Rannacher. Finite element methods for the incompressible Navier-Stokes equation. 1999.
http://www.iwr.uni-heidelberg.de/sfb359/Preprints1999.html.

[64] J. N. Reddy. On penalty function methods in the finite element analysis of flow problems. *Int. J. Num. Meth. in Fluids*, 18:853–870, 1982.

[65] J. N. Reddy. *An Introduction to The Finite Element Method.* McGraw-Hill, 1993.

[66] J. N. Reddy and D. K. Gartling. *The Finite Element Method in Heat Transfer and Fluid Dynamics.* CRC Press, 1994.

[67] M. P. Reddy and J. N. Reddy. Multigrid methods to accelerate the convergence of element-by-element solution algorithms for viscous incompressible flows. *Comp. Meth. Appl. Mech. Engng.*, 132:179–193, 1996.

[68] M. P. Reddy, J. N. Reddy, and H. U. Akay. Penalty finite element analysis of incompressible flows using element-by-element solution algorithms. *Comp. Meth. Appl. Mech. Engng.*, 100:169–205, 1992.

[69] M. Rudman. A volume-tracking method for incompressible multi-fluid flows with large density variations. *Int. J. Num. Meth. Fluids*, pages 357–378, 1998.

[70] T. Rusten and R. Winther. A preconditioned iterative method for saddle-point problems. *SIAM J. Matrix Anal.*, 1992.

[71] R. L. Sani, P. M. Gresho, R. L. Lee, and D. F. Griffiths. The cause and cure (?) of the spurious pressures generated by certain FEM solutions of the incompressible Navier-Stokes equations: part 1 (part 2). *Int. J. Num. Meth. in Fluids*, 1:17–43, 1981.

[72] R. Scardovelli and S. Zaleski. Direct numerical simulation of free-surface and interfacial flow. *Annual Rev. Fluid Mech.*, pages 567–603, 1999.

[73] J. Shen. On error estimates of the projection methods for the Navier-Stokes equations: First-order schemes. *SIAM J. Numer. Anal.*, 1996.

[74] J. Shen. On error estimates of the projection methods for the Navier-Stokes equations: Second-order schemes. *Math. Comp*, 1996.

[75] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition – Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.

[76] C. Taylor and P. Hood. A numerical solution of the Navier-Stokes equations using the finite element method. *J. Comp. Phys.*, 1:73–100, 1973.

[77] R. Temam. Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires. *Arc. Ration. Mech. Anal.*, 32:377–385, 1969.

[78] T. E. Tezduyar. Adaptive determination of the finite element stabilization parameters. In *Computational Fluid Dynamics Conference Proceeding*, 2001.

[79] T.E. Tezduyar, S. Aliabadi, and M. Behr. Interface-capturing technique (EDICT) for computation of unsteady flows with interfaces. *Computer Methods in Applied Mechanics and Engineering*, 155:235–248, 1998.

[80] K. E. Thompson and H. S. Fogler. Modeling flow in disordered packed beds from pore-scale fluid mechanics. *AIChE Journal*, 43(6):1377, 1997.

[81] M. Thuné, E. Mossberg, P. Olsson, J. Rantakokko, K. Åhlander, and K. Otto. Object-oriented construction of parallel PDE solvers. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools for Scientific Computing*, pages 203–226. Birkhäuser, 1997.

[82] L. Tobiska and R. Verfürth. Analysis of a streamline diffusion finite element method for the Stokes and Navier-Stokes equations. *SIAM J. Numer. Anal.*, 1996.

[83] S. Turek. A comparative study of time-stepping techniques for the incompressible Navier-Stokes equations: From fully implicit non-linear schemes to semi-implicit projection methods. *Int. J. Num. Meth. in Fluids*, 22:987–1011, 1996.

[84] S. Turek. *Efficient Solvers for Incompressible Flow Problems.* Springer, 1999.

[85] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object orientated techniques. *Computers in Physics*, 12(6):620–631, 1998.

[86] P. Wesseling. *Principles of Computational Fluid Dynamics.* Springer Series in Computational Mathematics. Springer, 2001.

[87] S. Ø. Wille. Nodal operator splitting adaptive finite element algorithms for the navier–stokes equations. *Int. J. Num. Meth. Fluids*, pages 959–975, 1998.

[88] O. C. Zienkiewicz and K. Morgan. *Finite Elements and Approximation.* Wiley, 1983.

[89] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method (3 volumes).* McGraw-Hill, 5th edition, 2000.

II

Mixed Finite Elements

K.-A. Mardal and H. P. Langtangen

In Langtangen and Tveito (eds): *Advanced Topics in Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*, Lecture Notes in Computational Science and Engineering, Springer, 2003.

Mixed Finite Elements

K.-A. Mardal[1,2] and H. P. Langtangen[1,2]

[1] Simula Research Laboratory
[2] Department of Informatics, University of Oslo

Abstract. This chapter explains the need for mixed finite element methods and the algorithmic ingredients of this discretization approach. Various Diffpack tools for easy programming of mixed methods on unstructured grids in 2D and 3D are described. As model problems for exemplifying the formulation and implementation of mixed finite elements we address the Stokes problem for creeping viscous flow and the system formulation of the Poisson equation. Efficient solution of the linear systems arising from mixed finite elements is treated in the chapter on block preconditioning.

4.1 Introduction

In this Chapter we will study two fundamental mathematical models in physics and engineering: the Stokes problem for slow (creeping) incompressible viscous fluid flow and the Poisson equation for, e.g., inviscid fluid flow, heat conduction, porous media flow, and electrostatics. The Stokes problem cannot be discretized by a straightforward Galerkin method as this method turns out to be unstable in the sense of giving non-physical oscillations in the pressure. Mixed finite element methods, however, result in a stable solution. In this chapter we use the term mixed finite elements, when the different physical unknowns utilize different finite element basis functions. In the Stokes problem, one example is quadratic elements for the velocity and linear elements for the pressure. The Poisson equation, on the other hand, does not need mixed finite element methods for a stable solution. However, if the main interest regards the gradient of the solution of the Poisson equation, the mixed formulation seems more natural because the gradient is one of the unknowns.

In the past, two particular difficulties have prevented widespread application of mixed finite element methods: (i) lack of convenient flexible implementations of the methods on general unstructured grids and (ii) the difficulty of constructing efficient solvers for the resulting linear systems. This chapter pays attention to the first issue, while a companion chapter [15] deals with the second. Actually, the methods and software from this chapter have applications far beyond the two model problems on which this exposition is focused. To the authors knowledge, Diffpack is at the time of writing the only software package that supports easy and flexible programming with mixed finite element methods on unstructured grids, coupled with state-of-the-art

iterative schemes, like multigrid, for optimal solution of the linear systems [15].

4.2 Model Problems

4.2.1 The Stokes Problem

The Stokes problem can be formulated as follows:

$$-\mu\nabla^2\boldsymbol{u} + \nabla p = \boldsymbol{f} \text{ in } \Omega, \quad \text{(equation of motion)}, \tag{4.1}$$

$$\nabla \cdot \boldsymbol{u} = 0 \text{ in } \Omega, \quad \text{(mass conservation)}, \tag{4.2}$$

$$\boldsymbol{u} = \boldsymbol{h} \text{ on } \partial\Omega_E, \tag{4.3}$$

$$\frac{\partial \boldsymbol{u}}{\partial n} + p\boldsymbol{n} = 0 \text{ on } \partial\Omega_N\,. \tag{4.4}$$

Here, $\boldsymbol{u}$ is the velocity of the fluid, p is the pressure in the fluid, $\boldsymbol{f}$ represents body forces and n is the unit normal vector pointing out of Ω. The boundary $\partial\Omega = \partial\Omega_E \cup \partial\Omega_N$, where $\partial\Omega_E$ and $\partial\Omega_N$ are the parts with essential and natural boundary conditions, respectively. The Stokes equation is the stationary linearized form of the Navier-Stokes equations and describes the creeping (low Reynolds number) flow of an incompressible Newtonian fluid.

4.2.2 The Mixed Poisson Problem

The Poisson equation

$$-\nabla \cdot (\lambda\nabla p) = g \text{ in } \Omega, \tag{4.5}$$

$$p = h \text{ on } \partial\Omega_E, \tag{4.6}$$

$$\frac{\partial p}{\partial n} = k \text{ on } \partial\Omega_N, \tag{4.7}$$

appears in many physical contexts. The boundary $\partial\Omega = \partial\Omega_E \cup \partial\Omega_N$, where $\partial\Omega_E$ and $\partial\Omega_N$ are the parts with essential and natural boundary conditions, respectively. The λ is here assumed to be a scalar function such that $\lambda_{min} \leq \lambda(\boldsymbol{x}) \leq \lambda_{max}$, where λ_{min} and λ_{max} are positive real numbers and $\boldsymbol{x} \in \Omega$. One interpretation regards porous media flow, where (4.5) arises from a conservation equation combined with Newton's second law. The mass conservation equation reads $\nabla \cdot \boldsymbol{u} = g$, where $\boldsymbol{u}$ is some flux and g denotes the injection or production through wells in groundwater flow or oil recovery. Newton's second law can be expressed by Darcy's law $\boldsymbol{u} = -\lambda\nabla p$, where λ denotes the mobility. This equation can be established as an average of the equation of motion in Stokes problem over a large number of pores. In porous media flow we are primarily interested in $\boldsymbol{u}$, which is usually computed by

solving the Poisson equation and computing $\boldsymbol{u} = -\lambda \nabla p$ numerically. The numerical differentiation implies a loss of accuracy. The mixed formulation of the Poisson equation allows us to approximate the velocity $\boldsymbol{u}$ as one of the unknowns. When applying mixed finite element methods the Poisson equation is reformulated as a system of partial differential equations (PDEs):

$$\frac{1}{\lambda}\boldsymbol{u} + \nabla p = 0 \text{ in } \Omega \quad \text{(Darcy's law)}, \tag{4.8}$$

$$\nabla \cdot \boldsymbol{u} = g \text{ in } \Omega \quad \text{(mass conservation)}, \tag{4.9}$$

$$p = h \text{ on } \partial\Omega_N, \tag{4.10}$$

$$\frac{\partial p}{\partial n} = \boldsymbol{u} \cdot \boldsymbol{n} = k \text{ on } \partial\Omega_E\,. \tag{4.11}$$

Notice that the essential boundary conditions in (4.6) appear as natural conditions in this formulation, while the natural conditions in (4.7) are essential.

Although the Stokes problem and the Poisson equation have seemingly similar mathematical structure, they require different types of mixed finite element methods. Knowing how to solve these two classes of problems should provide sufficient information to solve a wide range of systems of PDEs by mixed finite element methods.

The present chapter is organized as follows. In Section 4.3, we present the basic theory of mixed systems in an abstract setting. This abstract theory will introduce the Babuska-Brezzi conditions, i.e., the conditions that the mixed elements should meet. In Section 4.4, we present finite element spaces appropriate for solving our model problems and the corresponding software tools. We present the implementation of the simulators in Sections 4.5 and 4.6. Efficient iterative schemes, with multigrid, are described in [15]. The mixed finite element method is analyzed in [3,9].

4.3 Mixed Formulation

In this section we shall derive the finite element equations for our two model problems. First, we apply the weighted residual method [9] to the systems of PDEs and derive the resulting discrete equations. Thereafter, we present continuous mixed variational formulations, which can be discretized by introducing appropriate finite-dimensional function spaces [9]. The discretization of our problems needs to fulfill the discrete version of the Babuska-Brezzi conditions, which motivates the special finite elements presented in Section 4.4.

4.3.1 Weighted Residual Methods

The Stokes Problem. The starting point of a weighted residual formulation is the representation of the unknown scalar fields in terms of sums of finite

element basis functions. In the Stokes problem we need to use different basis functions for the velocity components and the pressure. Hence, we may write

$$\boldsymbol{u} \approx \hat{\boldsymbol{v}} = \sum_{r=1}^{d} \sum_{j=1}^{n_v} v_j^r \boldsymbol{e}^r N_j, \tag{4.12}$$

$$p \approx \hat{p} = \sum_{j=1}^{n_p} p_j L_j \,. \tag{4.13}$$

We use $\boldsymbol{u}$ for the continuous vector field, while $\boldsymbol{v}$ is the vector consisting of the velocity values at the nodal points. The N_j and L_j denote the j-th basis functions for the velocity and the pressure, respectively. The $\{\boldsymbol{e}^r\}$ are the unit vectors in Cartesian coordinates and d is the number of space dimensions. The number of nodes for the velocity and the pressure are n_v and n_p, respectively. Notice that the nodal points of the velocity and pressure fields do not necessarily coincide. The unknowns $\{v_j^r\}$ and $\{p_j\}$ are represented as vectors,

$$\boldsymbol{v} = [v_1^1, v_2^1, \ldots, v_{n_v}^1, v_1^2, \ldots, v_{n_v}^2, \ldots, v_1^d, \ldots v_{n_v}^d], \tag{4.14}$$

$$\boldsymbol{p} = [p_1, p_2, \ldots, p_{n_p}] \,. \tag{4.15}$$

The numbering used in (4.14) and (4.15) is only one of many possible numberings, which are described in Section 4.4.4.

Inserting the approximations $\hat{\boldsymbol{v}}$ and $\hat{p}$ in the equation of motion and the equation of continuity results in a residual since neither $\hat{\boldsymbol{v}}$ nor $\hat{p}$ are in general exact solutions of the PDE system. The idea of the weighted residual method is to force the residuals to be zero in a weighted mean. Galerkin's method is a version of the weighted residual method where the weighting functions are the same as the basis functions N_i and L_i. Application of Galerkin's method in the present example consists of using N_i as weighting function for the equation of motion and L_i as weighting function for the equation of continuity. In this way we generate $dn_v + n_p$ equations for the $dn_v + n_p$ unknowns in d space dimensions. The second-order derivative term in the equation of motion is integrated by parts. This is also done with the pressure gradient term $\nabla\hat{p}$. The resulting weighted residual or discrete weak form is as follows:

$$\int_\Omega \left(\mu\nabla\hat{v}^r \cdot \nabla N_i - \frac{\partial N_i}{\partial x^r}\hat{p}\right) d\Omega = \int_\Omega f^r N_i \, d\Omega, \quad r = 1, \ldots, d, \tag{4.16}$$

$$\int_\Omega L_i \nabla \cdot \hat{\boldsymbol{v}} \, d\Omega = 0 \,. \tag{4.17}$$

Notice that we have assumed open boundary conditions, $\frac{\partial \boldsymbol{v}}{\partial n} + p\boldsymbol{n} = 0$. The notation $\hat{v}^r$ means the r-th component of $\hat{\boldsymbol{v}}$,

$$\hat{v}^r = \hat{\boldsymbol{v}} \cdot \boldsymbol{e}^r = \sum_i v_i^r N_i, \tag{4.18}$$

with a similar interpretation for the other quantities with superscript r.

Inserting the finite element expressions for $\hat{\boldsymbol{v}}$ and $\hat{p}$ in (4.16)–(4.17), gives the following linear system:

$$\sum_{j=1}^{n_v} A_{ij} v_j^r + \sum_{j=1}^{n_p} B_{ij}^r p_j = c_i^r, \quad i = 1, \ldots, n_v, \ r = 1, \ldots, d, \tag{4.19}$$

$$\sum_{r=1}^{d} \sum_{j=1}^{n_v} B_{ji}^r v_j^r = 0 \quad i = 1, \ldots, n_p, \tag{4.20}$$

where

$$\hat{v}^r = \sum_{j=1}^{n_v} v_j^r N_j, \tag{4.21}$$

$$\hat{p} = \sum_{j=1}^{n_p} p_j L_j, \tag{4.22}$$

$$A_{ij} = \int_{\Omega} \mu \left(\sum_{k=1}^{d} \frac{\partial N_i}{\partial x_k} \frac{\partial N_j}{\partial x_k} \right) d\Omega, \tag{4.23}$$

$$B_{ij}^r = -\int_{\Omega} \frac{\partial N_i}{\partial x^r} L_j d\Omega, \tag{4.24}$$

$$c_i^r = \int_{\Omega} f^r N_i d\Omega \,. \tag{4.25}$$

The Poisson Problem. The expansions (4.12)–(4.13) are natural candidates when formulating a weighted residual method for the Poisson equation expressed as a system of PDEs (4.8)–(4.9). A Galerkin approach consists of using N_i as weighting functions for (4.8) and L_i as weighting functions for (4.9). The pressure gradient term in (4.8) is integrated by parts, resulting in the following system of discrete equations:

$$\sum_{j=1}^{n_v} A_{ij} v_j^r + \sum_{j=1}^{n_p} B_{ij}^r p_j = 0 \quad i = 1, \ldots, n_v, \ r = 1, \ldots, d, \tag{4.26}$$

$$\sum_{r=1}^{d} \sum_{j=1}^{n_v} B_{ji}^r v_j^r = g_i \quad i = 1, \ldots, n_p, \tag{4.27}$$

where

$$\hat{v}^r = \sum_{j=1}^{n_v} v_j^r N_j, \tag{4.28}$$

$$\hat{p} = \sum_{j=1}^{n_p} p_j L_j, \tag{4.29}$$

$$A_{ij} = \int_\Omega N_i N_j d\Omega, \tag{4.30}$$

$$B_{ij}^r = -\int_\Omega \frac{\partial N_i}{\partial x_r} L_j d\Omega, \tag{4.31}$$

$$g_i = \int_\Omega g L_i d\Omega\,. \tag{4.32}$$

4.3.2 Mixed Elements, Do We Really Need Them?

It is natural to ask whether mixed finite elements are actually needed. What will happen if one just discretizes the problem with a standard technique? One will then often experience non-physical pressure oscillations. The reason is that the pressure is not uniquely determined. This ambiguity comes in addition to the fact that the pressure is only determined up to a constant.

In this Section, the problem is presented from a purely algebraic point of view, to motivate that a naive discretization may lead to trouble (see also [18]). A typical discretization of the above described problems results in a matrix equation of the form:

$$\mathcal{A}\begin{bmatrix}\boldsymbol{v}\\ \boldsymbol{p}\end{bmatrix} = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}^T\\ \boldsymbol{B} & \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{v}\\ \boldsymbol{p}\end{bmatrix} = \begin{bmatrix}\boldsymbol{f}\\ \boldsymbol{g}\end{bmatrix}\,. \tag{4.33}$$

When is $\mathcal{A}$ invertible? The matrix is of dimension $(n+m)\times(n+m)$, where n is the length of the vector $\boldsymbol{v}$ and m is the length of $\boldsymbol{p}$. $\boldsymbol{A}$ is an $n\times n$ matrix, which is invertible for the problems considered here, and $\boldsymbol{B}$ is $m\times n$. We may therefore multiply the first equation by $\boldsymbol{A}^{-1}$ and obtain

$$\boldsymbol{v} = \boldsymbol{A}^{-1}\boldsymbol{f} - \boldsymbol{A}^{-1}\boldsymbol{B}^T\boldsymbol{p}\,. \tag{4.34}$$

We insert (4.34) in the second equation of (4.33) and get,

$$\boldsymbol{B}\boldsymbol{v} = \boldsymbol{B}(\boldsymbol{A}^{-1}\boldsymbol{f} - \boldsymbol{A}^{-1}\boldsymbol{B}^T\boldsymbol{p}) = \boldsymbol{g},$$

or in other words, we must solve

$$-\boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{B}^T\boldsymbol{p} = \boldsymbol{g} - \boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{f}\,.$$

Hence, the system (4.33) is invertible if $\boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{B}^T$ is. The $\boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{B}^T$ is usually called the pressure Schur complement. Since $\boldsymbol{B}$ is a $m \times n$ matrix it is not obvious that $\boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{B}^T$ is invertible. In fact, if the matrices come from a discretization where the velocity and the pressure are approximated with the same type of elements, then the pressure Schur complement is singular. This may result in pressure oscillations. The following algebraic analogy of the Babuska-Brezzi conditions ensure an invertible matrix $\mathcal{A}$,

$$\boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v} > 0, \quad \forall \boldsymbol{v} \in Kernel(\boldsymbol{B}), \quad (\boldsymbol{A} \text{ is sufficiently invertible}), \tag{4.35}$$

and

$$Kernel(\boldsymbol{B}^T) = \{0\}. \quad (\boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{B}^T \text{ is invertible}). \tag{4.36}$$

Although, it is not easy to deduce pressure oscillations from this algebraic discussion, it should be clear that a naive discretization does not necessarily lead to a well posed (at least non-singular) numerical problem.

We also supply the discussion with the following remark on standard linear elements approximation of fields with zero divergence. This is often refereed to as the locking phenomenon in elasticity theory (see, e.g., [2]).

Remark: The Locking Phenomenon. In the Stokes problem the approximate solution of the velocity should be *divergence free*, and it is not obvious that standard linear elements will give a good approximation under this constraint. In fact, an example of extremely poor approximation properties can be constructed with linear continuous elements and homogeneous Dirichlet boundary conditions. The only divergence free function satisfying these conditions is $\hat{\boldsymbol{v}} = 0$, regardless of the mesh parameter h. (This result depends on the geometry of the triangulation, but it will often hold.) Hence, regardless of the equation we want to solve, the constraint, $\operatorname{div} \hat{\boldsymbol{v}} = 0$, enforces $\hat{\boldsymbol{v}} = 0$.

These phenomena clearly shows that a careful discretization of our model problems is needed. We will now address the abstract properties that should be met by the mixed elements. The following conditions will ensure a well-posed discrete problem (i.e., there exist a unique solution depending continuously on the data).

4.3.3 Function Space Formulation

Both our problems can be viewed within the same abstract setting. Let $\boldsymbol{H}$ and P be suitable Hilbert spaces for $\boldsymbol{u}$ and p, respectively. Then both these model problems can be formulated as follows:
Find $(\boldsymbol{u}, p) \in \boldsymbol{H} \times P$ such that

$$\begin{array}{ll} a(\boldsymbol{u},\boldsymbol{w}) + b(p,\boldsymbol{w}) = (\boldsymbol{f},\boldsymbol{w}), & \forall \boldsymbol{w} \in \boldsymbol{H}, \\ b(q,\boldsymbol{u}) \qquad\qquad\quad = (g,q), & \forall q \in P\,. \end{array} \tag{4.37}$$

The analysis of the mixed finite elements is related to this specific structure of the problem. We will need the following spaces:

- $\boldsymbol{L}^2(\Omega) = \{\ \boldsymbol{w} \mid \int_\Omega \boldsymbol{w}^2 d\Omega < \infty\}$
- $\boldsymbol{H}^k(\Omega) = \{\ \boldsymbol{w} \mid \sum_{|\alpha|\leq k} \int_\Omega (D^\alpha \boldsymbol{w})^2 d\Omega < \infty\}$, where $k = |\alpha|$ and

$$D^\alpha w_i = \frac{\partial^{|\alpha|} w_i}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}}$$

- $\boldsymbol{H}(\mathrm{div}; \Omega) = \{\ \boldsymbol{w}\ \in \boldsymbol{L}^2(\Omega) \mid \nabla \cdot \boldsymbol{w} \in L^2(\Omega)\}$

These spaces are Hilbert spaces of vector functions, the corresponding spaces for scalar functions, $L^2(\Omega)$ and $H^1(\Omega)$, are also used. Note that the derivatives are in distribution sense.

The Stokes Problem. The bilinear forms associated with the Stokes problem are:

$$a(\boldsymbol{u}, \boldsymbol{w}) = \int_\Omega \mu \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{w} d\Omega, \tag{4.38}$$

$$b(p, \boldsymbol{w}) = -\int_\Omega p \nabla \cdot \boldsymbol{w} d\Omega\,. \tag{4.39}$$

$$\tag{4.40}$$

The product $\nabla \boldsymbol{u} \cdot \nabla \boldsymbol{w}$ is the "scalar tensor product"

$$\nabla \boldsymbol{u} \cdot \nabla \boldsymbol{w} = \sum_{i=1}^d \nabla u_i \cdot \nabla w_i = \sum_{i=1}^d \sum_{j=1}^d \frac{\partial u_i}{\partial x_j} \frac{\partial w_i}{\partial x_j}\,.$$

Suitable function spaces for this model problem are $\boldsymbol{H} = \boldsymbol{H}^1(\Omega)$ and $P = L^2(\Omega)$. The linear forms (the right-hand sides of (4.37)) are:

$$(\boldsymbol{f}, \boldsymbol{w}) = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{w}\, d\Omega + \int_{\partial\Omega_N} (\mu \boldsymbol{w} \cdot \frac{\partial \boldsymbol{u}}{\partial n} - p\boldsymbol{n} \cdot \boldsymbol{w})\, ds, \tag{4.41}$$

$$(g, q) = 0, \tag{4.42}$$

where $(\frac{\partial \boldsymbol{u}}{\partial n} - p\boldsymbol{n})$ is known at the (parts of the) boundary with natural boundary condition.

The Poisson Problem. The bilinear forms in the mixed formulation of the Poisson equation read:

$$a(\boldsymbol{u}, \boldsymbol{w}) = \int_\Omega \lambda^{-1} \boldsymbol{u} \cdot \boldsymbol{w}\, d\Omega, \tag{4.43}$$

$$b(q, \boldsymbol{u}) = \int_\Omega \nabla q \cdot \boldsymbol{u}\, d\Omega = -\int_\Omega q\, \nabla \cdot \boldsymbol{u}\, d\Omega + \int_{\Omega_N} p\boldsymbol{u} \cdot \boldsymbol{n}\,. \tag{4.44}$$

The natural conditions here are the essential boundary conditions (4.6) As we see from (4.44), we have two possible choices for $b(\cdot,\cdot)$. The first alternative is $b(p,\boldsymbol{w}) = (\nabla p, \boldsymbol{w})$. We must then require that $p \in H^1(\Omega)$ and $\boldsymbol{u} \in \boldsymbol{L}^2(\Omega)$. Another formulation is $b(p,\boldsymbol{w}) = -(p, \nabla\cdot\boldsymbol{w}) + \int_{\Omega_N} p\boldsymbol{u}\cdot\boldsymbol{n}$, where we have used integration by parts. We must then require that $\boldsymbol{u} \in \boldsymbol{H}(\mathrm{div};\Omega)$ and that $p \in L^2(\Omega)$. The difference between $\boldsymbol{H}(\mathrm{div};\Omega)$ and $\boldsymbol{H}^1(\Omega)$ will be important in the construction of the finite elements and the preconditioner in [15]. The linear forms for the Poisson problem (the right-hand sides of (4.37)) are,

$$(\boldsymbol{f},\boldsymbol{w}) = \int_{\partial\Omega_N} p\,\boldsymbol{n}\cdot\boldsymbol{w} ds, \tag{4.45}$$

$$(g,q) = \int_{\Omega} g\cdot q\, d\Omega\,. \tag{4.46}$$

4.3.4 The Babuska-Brezzi Conditions

In the previous section we saw that the fields $\boldsymbol{u}$ and p had different regularity requirements. The discrete analogy should reflect these differences and we therefore use elements designed for the problem at hand. Any combination of elements will *not* work. Let us assume that we have chosen two sets of basis functions $\{\boldsymbol{N}_i\}$ and $\{L_i\}$ that span $\boldsymbol{H}_h$ and P_h, respectively. We will in the following assume that $\boldsymbol{H}_h \subset \boldsymbol{H}$ and $P_h \subset P$. The norms in $\boldsymbol{H}_h$ and P_h can then be derived from $\boldsymbol{H}$ and P, respectively. The Galerkin formulation of the problem can be written on the form:
Find $(\hat{\boldsymbol{v}},\hat{p}) \in \boldsymbol{H}_h \times P_h$ such that

$$\begin{array}{ll} a(\hat{\boldsymbol{v}},\boldsymbol{m}) + b(\hat{p},\boldsymbol{m}) = (\boldsymbol{f},\boldsymbol{m}), & \forall \boldsymbol{m} \in \boldsymbol{H}_h, \\ b(n,\hat{\boldsymbol{v}}) \qquad\qquad\quad = (g,n), & \forall n \in P_h\,. \end{array} \tag{4.47}$$

This is just an abstract formulation of (4.19)-(4.20) and (4.26)-(4.27). The matrix equation of this problem is on the form (4.33). From the above definitions of the $\boldsymbol{H}$ and P we have that $a(\cdot,\cdot)$ and $b(\cdot,\cdot)$ are bounded for both model problems, i.e.,

$$a(\boldsymbol{u},\boldsymbol{v}) \le C\|\boldsymbol{u}\|_{\boldsymbol{H}}\|\boldsymbol{v}\|_{\boldsymbol{H}}, \quad \forall \boldsymbol{u},\boldsymbol{v} \in \boldsymbol{H}, \tag{4.48}$$

$$b(q,\boldsymbol{v}) \le C\|q\|_P\|\boldsymbol{v}\|_{\boldsymbol{H}}, \quad \forall \boldsymbol{v} \in \boldsymbol{H}, q \in P\,. \tag{4.49}$$

However, the two following conditions will in general not be fulfilled. The elements must be "designed" to meet them.

Condition 1. There exists a constant α (independent of h) such that $a(\cdot,\cdot)$ is $\boldsymbol{H}$-elliptic on $\boldsymbol{H}_h$:

$$a(\hat{\boldsymbol{w}},\hat{\boldsymbol{w}}) \ge \alpha\|\hat{\boldsymbol{w}}\|^2_{\boldsymbol{H}}, \quad \forall \hat{\boldsymbol{w}} \in \boldsymbol{H}_h\,. \tag{4.50}$$

This property is trivially satisfied for the Stokes problem. Elements for the mixed Poisson problem are designed to meet this condition.

Condition 2. There exists a constant β (independent of h) such that

$$0 < \beta := \inf_{\hat{q} \in P_h} \sup_{\boldsymbol{w}_h \in \boldsymbol{H}_h} \frac{b(\hat{\boldsymbol{w}}, \hat{q})}{\|\hat{\boldsymbol{w}}\|_{\boldsymbol{H}} \|\hat{q}\|_P}, \tag{4.51}$$

The condition 2 is often called the Babuska-Brezzi condition or the inf-sup condition. These two conditions (in addition to (4.48) and (4.49)) are necessary for a well-posed discrete problem [3]. The algebraic analogies of the conditions 1 and 2 are (4.35) and (4.36), respectively. However, there is an important difference. The conditions are designed to ensure *optimal accuracy*. Hence, for $(\hat{\boldsymbol{v}}, \hat{p})$ found by (4.47) we have,

$$\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{H}} + \|p - \hat{p}\|_P \leq c\big(\inf_{\boldsymbol{v} \in \boldsymbol{H}_h} \|\boldsymbol{v} - \boldsymbol{u}\|_{\boldsymbol{H}} + \inf_{q \in P_h} \|p - q\|_P\big)\,. \tag{4.52}$$

To get optimal accuracy, we must have $\beta > 0$ independently of h (see, e.g., [3]). This is not needed to ensure that the algebraic system in (4.33) is invertible. The β may then decrease towards zero as the grid is refined.

Regularization of Stokes problem. We can avoid the Babuska-Brezzi conditions by perturbing the problem. We restate the problem as:
find $(\hat{\boldsymbol{v}}^\epsilon, \hat{p}^\epsilon)$ *such that*

$$\begin{aligned} a(\hat{\boldsymbol{v}}^\epsilon, \hat{\boldsymbol{w}}) + b(p_h^\epsilon, \hat{\boldsymbol{w}}) &= (\boldsymbol{f}, \hat{\boldsymbol{w}}), \quad \forall \hat{\boldsymbol{w}} \in \boldsymbol{H}_h, \\ b(\hat{q}, \hat{\boldsymbol{v}}^\epsilon) - \epsilon c(\hat{q}, \hat{p}^\epsilon) &= (g, \hat{q}), \quad \forall \hat{q} \in P_h, \end{aligned} \tag{4.53}$$

where ϵ should be small. With this perturbation of the original problem, we get the non-singular matrix equation,

$$\mathcal{A} \begin{bmatrix} \boldsymbol{v}^\epsilon \\ \boldsymbol{p}^\epsilon \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}^T \\ \boldsymbol{B} & -\epsilon \boldsymbol{M} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}^\epsilon \\ \boldsymbol{p}^\epsilon \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{g} + \epsilon \boldsymbol{d} \end{bmatrix}. \tag{4.54}$$

There are mainly three methods used to construct $\epsilon \boldsymbol{M}$, all based on perturbed versions of the equation of continuity,

$$\nabla \cdot \boldsymbol{v} = \epsilon \nabla^2 p, \tag{4.55}$$

$$\nabla \cdot \boldsymbol{v} = -\epsilon p, \tag{4.56}$$

$$\nabla \cdot \boldsymbol{v} = -\epsilon \frac{\partial p}{\partial t}\,. \tag{4.57}$$

The approach (4.55) was derived with the purpose of stabilizing pressure oscillations and allowing standard grids and elements (see, e.g., [8]). The ϵ is usually αh^2, where α can be tuned. This approach is usually preferred and does in fact satisfy a slightly more complicated version of the Babuska-Brezzi conditions, regardless of the choice of elements. The equations (4.56) and

(4.57) were not derived as stabilization methods, but were initiated from alternative physical and mathematical formulations of viscous incompressible flow. They are usually referred to as the penalty method and the artificial compressibility method, respectively [7]. The penalty method allows for elimination of the pressure and is usually used together with reduced integration. A Navier-Stokes solver using the penalty method is implemented in Diffpack and is described in [9]. The regularization techniques have relations to common operator splitting techniques [17].

4.4 Some Basic Concepts of a Finite Element

This Section presents the basics of a general finite element. We will focus on a definition that should be easy to use in a general implementation with mixed elements on a general unstructured mesh. This will be needed later when we implement the finite elements appropriate to our model problems. For further reading about the finite element method, see [2,3,10,9]. Detailed information on finite element programming in Diffpack can be found in [9,13].

4.4.1 A General Finite Element

Consider a spatially varying scalar field $f(x)$, where $x \in \Omega \subset \mathbb{R}^d$. The field $f(x)$ has the following approximate expansion $\hat{f}(x)$ in a finite element method:

$$f(x) \approx \hat{f}(x) = \sum_{j=1}^{n} \alpha_j N_j(x)\,. \tag{4.58}$$

If f also depends on time, we write

$$f(x,t) \approx \hat{f}(x,t) = \sum_{j=1}^{n} \alpha_j(t) N_j(x)\,. \tag{4.59}$$

We refer to $N_j(x)$ as the basis functions in global coordinates and α_j are the *coefficients* in the expansion, or the global degrees of freedom of the discrete scalar field. The choice of basis functions $N_j(x)$ is problem dependent, and we "should" choose them based on the regularity requirements of the differential equation. The basis functions are piecewise polynomials and we can therefore deduce the following (see page 67 in [10]):

$$V_h \subset H^1(\Omega) \Leftrightarrow V_h \subset C^0(\Omega), \tag{4.60}$$

$$V_h \subset L^2(\Omega) \Leftrightarrow V_h \subset C^{-1}(\Omega), \tag{4.61}$$

where $C^0(\Omega)$ is the space of continuous functions and $C^{-1}(\Omega)$ is the space of discontinuous functions (with finite discontinuities). A vector element can often be made as a superposition of scalar elements.

Central to the finite element method is the partition of Ω into non-overlapping *subdomains* Ω_e, $e = 1, \ldots, E$, $\Omega = \Omega_1 \cup \cdots \cup \Omega_E$. In each subdomain Ω_e we define a set of basis functions with support only in Ω_e and the neighboring subdomains. The basis functions are associated with a number of local degrees of freedom. A subdomain, along with its basis functions and degrees of freedom, defines a *finite element*, and the finite elements throughout the domain define a finite element space. The finite element space has a global number of degrees of freedom. These degrees of freedom can be the values of the unknown functions at a set of points (nodes). From an implementational point of view, a great advantage of the finite element method is that it can be evaluated locally and thereafter be assembled to a global linear system. We therefore focus on a local consideration of the finite element method. Similar definitions of finite elements and more information can be found in [2,10,9].

For each Ω_e there is a parametric mapping M_e^{-1} from the physical subdomain to a reference subdomain:

$$\xi = M_e^{-1}(x), \quad x = M_e(\xi), \quad \xi \in \Omega_r \subset \mathbb{R}^d, \quad \boldsymbol{x} \in \Omega_e \subset \mathbb{R}^d\,. \tag{4.62}$$

The Ω_r domain is often called the *reference domain*. The mapping of this reference domain to the physical coordinate system is defined by a set of geometry functions $G_i(\xi)$, i.e., M_e is defined in terms of some G_i functions. Normally, we associate the term *reference element* with the reference domain together with its degrees of freedom and basis functions.

Definition 3. An *isoparametric* element is a finite element for which the basis functions and the geometry functions coincide.

However, most mixed elements are not isoparametric. Diffpack therefore has a very general definition of a finite element. This definition is implemented in `ElmDef` which is the base class for all the elements.

Definition 4. A reference element is characterized by a reference subdomain Ω_r with a set of n_g points $p_1, \ldots p_{n_g}$, referred to as geometry nodes, such that the mapping M_e from the reference domain Ω_r onto the corresponding physical domain Ω_e has the form

$$x(\xi) = \sum_{k=1}^{n_g} G_k(\xi) x_k\,. \tag{4.63}$$

Here x_k are the coordinates in physical space corresponding to the geometry node p_j. Moreover, the geometry function $G_j(\xi)$ has the property that $G_j(p_i) = \delta_{ij}$, where δ_{ij} is the Kronecker delta. The (transposed) Jacobi matrix element J_{ij} of the mapping M_e is then given by

$$J_{ij} = \sum_{k=1}^{n_g} \frac{\partial G_k(\xi)}{\partial \xi_i} x_k^j, \tag{4.64}$$

where x_k^j is the j-th component of x_k.

If the element is non-isoparametric we must also specify the basis functions. We define a number n_{bf} of basis functions $N_1^{ref}, \ldots N_{n_{bf}}^{ref}$ with n_b associated nodes at points $q_1, \ldots q_{n_b}$ in the reference domain. These may in principle be chosen arbitrarily, designed for different purposes. A global basis function is then defined in terms of a mapping of the corresponding reference basis function. We can define a global element by using the geometry mapping M_e in (4.62);

$$N_i^{glob}(x) = N_i^{ref}(M_e^{-1}(x)) = N_i^{ref}(\xi).$$

Definition 5. The restriction of a finite element scalar field, as in (4.58), to a finite element in the reference domain is given by

$$f(x)|_{\Omega_r} = \sum_{j=1}^{n_{bf}} \beta_j N_j(\xi), \quad \xi = (\xi_1, \ldots, \xi_d) \in \Omega_r \,.$$

Here β_j are the local expansion coefficients, corresponding to the local degrees of freedom. The local basis functions are $N_j(\xi)$ and n_{bf} is the number of local basis functions.

Finite element applications usually require the derivatives with respect to the physical (global) coordinates, $\partial N_i/\partial x_j$. The relation between derivatives with respect to local and global coordinates is given by

$$\frac{\partial N_i}{\partial \xi_j} = J_{ij} \frac{\partial N_i}{\partial x_j},$$

where J_{ij} is given in (4.64).

The basis functions in the isoparametric elements are exactly the geometry functions, and the degrees of freedom are represented in the geometry nodes. This means that we get the physical elements (shape of the element and the expressions for the basis function in physical coordinates) by applying the parametric mapping $M_e(\xi)$ on the reference elements. Non-isoparametric elements might also be defined in terms of a mapped reference element. However, such elements might also have other non-equivalent definitions, as we will see for the elements used in $\boldsymbol{H}(\mathrm{div}; \Omega)$.

4.4.2 Examples of Finite Element Spaces

This section describes some typical mixed elements and their implementation. We also list the error estimates for each element. These estimates are used later in the Sections 4.6.3 and 4.6.5 to verify the simulators. We assume that the triangulation is shape-regular and quasi-uniform. To clarify the notation introduced in the previous section we apply it to some well-known finite elements.

The Linear Triangle. The popular linear 2D triangular element is defined as follows. Let the reference domain be,

$$\Omega_r = \{(\xi_1, \xi_2) \,|\, 0 \leq \xi_1 \leq 1,\ \xi_2 \leq \xi_1\}\,. \tag{4.65}$$

Moreover, $n_g = n_{bf} = n_b = 3$, $G_i = N_i$ and

$$N_1(\xi_1, \xi_2) = 1 - \xi_1 - \xi_2, \tag{4.66}$$
$$N_2(\xi_1, \xi_2) = \xi_1, \tag{4.67}$$
$$N_3(\xi_1, \xi_2) = \xi_2\,. \tag{4.68}$$

The geometry and basis function nodes are the corner points of the triangle; $p_1 = (0,0)$, $p_2 = (1,0)$ and $p_3 = (0,1)$. This element is named `ElmT3n2D` in Diffpack. The Figure 4.1 shows the element. The basis function nodes are indicated by the black circles. More details on the local side and node

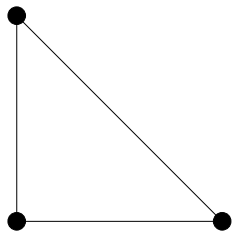

Fig. 4.1. Sketch of the 2D linear triangle element.

numbering can be found in [11].

The 2D Piecewise Constant Element. The 2D piecewise constant element over a triangle is defined by Ω_r in (4.65), $n_b = n_{bf} = 1$, $n_g = 3$, $N_1(\xi_1, \xi_2) = 1$ for all $\xi_1, \xi_2 \in \Omega_r$, while G_i and p_i are identical to the expressions for the standard linear triangle element, described above. The location of the basis function node is arbitrary, but an obvious choice is $q_1 = (1/3, 1/3)$. The name in Diffpack is `ElmT3gn1bn2D`, which refers to its basic properties. It is a triangle element, `ElmT`, with three geometry nodes, `3gn`, and one basis function node, `1bn`, in 2D, `2D`. The Figure 4.2 shows the element.

As we stated in Section 4.3, our model problems require delicate combinations of finite element spaces, i.e, they must satisfy the Babuska-Brezzi conditions (2). Some appropriate mixed elements are presented here and we also discuss the implementation of these.

Mixed Elements for the Stokes Problem. We will now describe the mixed Stokes elements implemented in Diffpack. The Mini and Taylor-Hood

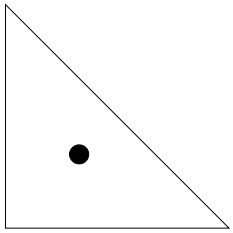

Fig. 4.2. Sketch of the 2D piecewise const element.

elements satisfy the Babuska-Brezzi conditions (4.50)-(4.51) with $\boldsymbol{H} = \boldsymbol{H}^1(\Omega)$ and $P = L^2(\Omega)$. The Crouzeix-Raviart element is non-conforming and the $\|\cdot\|_{\boldsymbol{H}^1(\Omega)}$ norm has to be replaced with the element-wise norm $\|\cdot\|_{\boldsymbol{H}_h{}^1(\Omega)}$ [5].

The Taylor–Hood element. When solving Stokes problem or Navier-Stokes-type of equations for the pressure and velocity fields, mixed finite element methods are traditionally used. A possible choice is to let the geometry be described by six nodes in a triangle reference element. The velocity components may then use basis functions that coincide with the geometry functions ($n_g = n_b = n_{bf}$, $q_i = p_i$ and $N_i = G_i$). The pressure field, on the other hand, is based on an element where the geometry is the same as for the velocity field, but where the basis functions are linear. For such an element, we see that the basis function nodes are identical to a subset of the geometry function nodes, the corner nodes. The corresponding basis functions are the same as in the linear triangle element (4.66)-(4.68). Both the velocity and the pressure elements are continuous (more regularity than strictly needed for the pressure). For sufficiently regular $(\boldsymbol{u}, p)$ we have the following approximation result for $k = 1, 2$ (e.g., [9]).

$$\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{H}^1(\Omega)} + \|p - \hat{p}\|_{L^2(\Omega)} \leq Ch^k(\|\boldsymbol{u}\|_{\boldsymbol{H}^{k+1}(\Omega)} + \|p\|_{H^k(\Omega)}), \quad (4.69)$$

$$\|p - \hat{p}\|_{H^1(\Omega)} \leq Ch^{k-1}(\|\boldsymbol{u}\|_{\boldsymbol{H}^{k+1}(\Omega)} + \|p\|_{H^k(\Omega)}) . \quad (4.70)$$

If Ω is convex we have,

$$\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{L}^2(\Omega)} \leq Ch^{k+1}(\|\boldsymbol{u}\|_{\boldsymbol{H}^{k+1}(\Omega)} + \|p\|_{H^k(\Omega)}) . \quad (4.71)$$

The constant C is here and in the following used as a generic constant independent of the mesh size h.

These elements are implemented in Diffpack as `ElmT6n2D` (continuous quadratic polynomials) and `ElmT6gn3bn2D` (continuous linear polynomials). The same elements exist on quadrilaterals and is implemented as `ElmB9n2D` and `ElmB9gn4bn2D`. The Figure 4.3 shows the degrees of freedom in the element. In this and the following examples we have basis function nodes for both the pressure and the velocity. The black circles are velocity nodes, while the black squares denote nodes for both the pressure and the velocity.

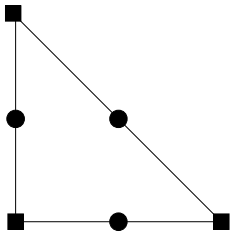

Fig. 4.3. Sketch of the 2D Taylor–Hood element; Quadratic velocity and linear pressure elements.

The Mini element. Another approximation can be obtained by using the so-called Mini element [1]. The starting point is continuous linear elements for both the pressure and the velocity. This combination does not satisfy the Babuska-Brezzi conditions, but if we enrich the velocity space with the so-called "bubble" function, it does. Let the triangle be numbered such that e_i are the edges at the opposite side of the vertex v_i. Let λ_i be a linear function such that $\lambda_i(v_i) = 1$, while $\lambda_i(x_j) = 0\ \forall x_j$ on e_i. The function $\lambda_1\lambda_2\lambda_3$ is the so called bubble function. The bubble function is zero on the edge of the element and has therefore support only in one element. The degrees of freedom are the nodes at the vertices and the center of the triangle. Hence, this element is simply a linear triangle element with one additional (bubble) basis function and the associated basis function node. Hence, $n_g = 3$, $n_{bf} = n_b = 4$. The basis functions in the reference element are

$$N_i(\xi_1, \xi_2) = G_i(\xi_1, \xi_2), \quad \text{for } i \leq 3, \tag{4.72}$$

$$N_4(\xi_1, \xi_2) = 27\,(1 - \xi_1 - \xi_2)\xi_1\xi_2, \tag{4.73}$$

where the geometry functions, G_i, are defined by the linear triangle element (page 166). The corresponding basis function nodes read,

$$q_i = p_i, \quad \text{for } i \leq 3, \tag{4.74}$$

$$q_4 = (1/3, 1/3)\,. \tag{4.75}$$

We have the following error estimate (e.g., [9]) for the Mini element:

$$\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{H}^1(\Omega)} + \|p - \hat{p}\|_{L^2(\Omega)} \leq Ch(\|\boldsymbol{u}\|_{\boldsymbol{H}^2(\Omega)} + \|p\|_{H^1(\Omega)})\,. \tag{4.76}$$

If Ω is convex we have,

$$\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{L}^2(\Omega)} \leq Ch^2(\|\boldsymbol{u}\|_{\boldsymbol{H}^2(\Omega)} + \|p\|_{H^1(\Omega)})\,. \tag{4.77}$$

We note that the approximation results rely on the linear polynomials, whereas the bubble functions give stability.

These elements are available in Diffpack as `ElmT3gn4bn2D` and should be used together with linear pressure elements, `ElmT3n2D`. The degrees of freedom is shown in figure 4.4.

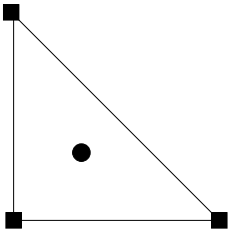

Fig. 4.4. Sketch of the 2D Mini velocity element and the linear pressure element.

Definition 6. A finite element space V_h is a *conforming* approximation of V if it is a subspace of V. A non-conforming finite element space approximation of V is an outer approximation, that is, the finite element space is not a subspace of V.

The Crouzeix-Raviart elements. The Crouzeix-Raviart element is linear and continuous at the midpoints of the triangle edges [5]. The midpoints are the only points where it is continuous. Hence, this element is not in $H^1(\Omega)$. It is not conforming and $\boldsymbol{H}^1(\Omega)$ is therefore replaced with $\boldsymbol{H}^1_h(\Omega)$. The element has three basis functions and three associated basis function nodes, $n_g = n_{bf} = n_b = 3$. The basis functions in the reference element are,

$$\begin{aligned} N_1(\xi_1, \xi_2) &= 1 - 2\xi_2, \\ N_2(\xi_1, \xi_2) &= 2(\xi_1 + \xi_2) - 1, \\ N_3(\xi_1, \xi_2) &= 1 - 2\xi_1, \end{aligned}$$

and the corresponding basis function nodes are

$$\begin{aligned} q_1 &= (1/2, 0), \\ q_2 &= (1/2, 1/2), \\ q_3 &= (0, 1/2). \end{aligned}$$

The error estimates for this element are [5],

$$\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{H}^1(\Omega)_h} \leq Ch(\|\boldsymbol{u}\|_{\boldsymbol{H}^2(\Omega)} + \|p\|_{H^1(\Omega)}), \tag{4.78}$$

$$\|p - \hat{p}\| \leq Ch(\|\boldsymbol{u}\|_{\boldsymbol{H}^2(\Omega)} + \|p\|_{H^1(\Omega)}). \tag{4.79}$$

If Ω is convex we have,

$$\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{L}^2(\Omega)} \leq Ch^2(\|\boldsymbol{u}\|_{\boldsymbol{H}^2(\Omega)} + \|p\|_{H^1(\Omega)}). \tag{4.80}$$

It is implemented as `ElmT3gn3bn2D` and is used with the discontinuous constant pressure element `ElmT3gn1bn2D`. The degrees of freedom is shown in figure 4.5. The white circle shows the pressure node. Note that this element only work with essential boundary conditions.

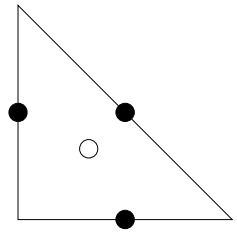

Fig. 4.5. Sketch of the 2D Crouzeix–Raviart element.

Mixed Elements for the Poisson Problem. This section describes mixed elements suitable for the mixed Poisson problem. They satisfy the Babuska–Brezzi conditions (4.50)-(4.51) with $\boldsymbol{H} = \boldsymbol{H}(\mathrm{div};\Omega)$ and $P = L^2(\Omega)$. In $\boldsymbol{H}(\mathrm{div};\Omega)$ continuity is only needed in the normal direction on the sides of the elements [3]. The following elements are designed to meet this requirement.

The Raviart–Thomas elements. A popular choice of elements is the Raviart-Thomas element [19] of order 0 for the velocity approximation $\hat{\boldsymbol{v}}$ and discontinuous piecewise constants for $\hat{p}$. The Raviart-Thomas elements are designed to approximate $\boldsymbol{H}(\mathrm{div};\Omega)$. They are continuous only in the normal direction across the elements boundaries. The Raviart-Thomas elements of order 0 are on the form:

$$\boldsymbol{N}_i = \begin{pmatrix} a+dx \\ b+dy \\ c+dz \end{pmatrix} \text{ in 3D and } \boldsymbol{N}_i = \begin{pmatrix} a+dx \\ b+dy \end{pmatrix} \text{ in 2D}\,. \tag{4.81}$$

The constants a, b, c, d are determined such that

$$\boldsymbol{N}_i \cdot \boldsymbol{n}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}\,. \tag{4.82}$$

where $\boldsymbol{N}_i$ are the basis functions and $\boldsymbol{n}_j$ are the normal vectors associated with the side i and j, respectively. This yields a global definition of the elements.

Using (4.81)-(4.82) to the reference element geometry we get the following basis functions in the reference element. The element has three basis functions and three associated basis function nodes, $n_g = n_{bf} = n_b = 3$. The ξ_1 components are,

$$\begin{aligned} N_1(\xi_1,\xi_2) &= -\xi_1, \\ N_2(\xi_1,\xi_2) &= -\sqrt{2}\,\xi_1, \\ N_3(\xi_1,\xi_2) &= 1-\xi_1, \end{aligned}$$

and the ξ_2 components read,

$$N_1(\xi_1, \xi_2) = 1 - \xi_2,$$
$$N_2(\xi_1, \xi_2) = -\sqrt{2}\,\xi_2,$$
$$N_3(\xi_1, \xi_2) = -\xi_2\,.$$

The basis function nodes are,

$$q_1 = (1/2, 0),$$
$$q_2 = (1/2, 1/2),$$
$$q_3 = (0, 1/2)\,.$$

While the previously mentioned elements is used such that the vector field is composed by scalar elements,

$$\hat{\boldsymbol{v}} = \sum_i \mathbf{a}_i N_i,$$

the Raviart-Thomas element is a *vector* element in the sense that

$$\hat{\boldsymbol{v}} = \sum_i a_i \boldsymbol{N}_i\,.$$

Moreover, these vector elements need to satisfy (4.82) globally, and this is not necessarily the case. We wish to define these elements in terms of a mapped reference element, because this is a very flexible strategy.

Following the basic concept of scalar elements we wish to use the definition for the reference element to map the reference element to the global element. However, the usual geometry mapping M_e does *not* preserve the continuity in the normal direction and the mapped elements will therefore not be in $\boldsymbol{H}(\mathrm{div}; \Omega)$. A slightly modified mapping does preserve the continuity in the normal direction, but unfortunately the mapping can not be defined componentwise by scalar elements. We remember the geometry mapping:

$$\boldsymbol{x} = \boldsymbol{M}_e(\xi)\,.$$

In order to make elements that are continuous in the normal direction of the mapped reference element, we must assume that the geometry mapping is affine. Affine mappings can be expressed by

$$\boldsymbol{x} = \boldsymbol{M}_e(\xi) = \boldsymbol{x}_e + \boldsymbol{B}_e\xi\,.$$

for some matrix $\boldsymbol{B}_e$ and a fixed point $\boldsymbol{x}_e$ in physical space. The mapped reference element can then be defined as:

$$\boldsymbol{N}(\boldsymbol{x}) = \frac{1}{\det \boldsymbol{B}_e}\boldsymbol{B}_e \cdot \boldsymbol{N}^{ref}(\boldsymbol{M}_e^{-1}(\boldsymbol{x}))\,. \tag{4.83}$$

This mapping is implemented in `MxMapping`. The element is a vector element and this is reflected in the code:

```
mfe.N(d,i);
mfe.dN(d,i,k);
```

Here, `MxFiniteElement` uses `MxMapping` to compute the basis function and its derivatives. Notice that this is in contrast to the standard use of vector elements composed by scalar elements,

```
mfe(d).N(i);
mfe(d).dN(i,k);
```

An example of use can be found in `PoissonMx.cpp`. To check whether the element requires the special mapping one can check the boolean variable `special_mapping` in `ElmDef` or `MxFiniteElement`. Vector fields based on these elements require a special initialization. This is described in section 4.6.4

The 2D Raviart-Thomas reference element is shown in Figure 4.6 and we refer to [3,19] for more information. We have the following approximation properties [6]:

$$\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{L}^2(\Omega)} \leq ch\|\boldsymbol{u}\|_{\boldsymbol{H}^1(\Omega))} \tag{4.84}$$

$$\|p - \hat{p}\|_{L^2(\Omega)} \leq c(h\|p\|_{H^1(\Omega)} + h^2\|p\|_{H^2(\Omega)}) \,. \tag{4.85}$$

Figure 4.6 shows the degrees of freedom in the element.

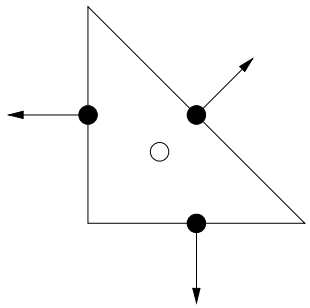

Fig. 4.6. Sketch of the 2D Raviart-Thomas element for the mixed formulation of the Poisson equation.

A Mixed Poisson Element Which is Robust for Stokes Problem All the previous elements discussed have been designed either for the Stokes problem or the mixed Poisson problem. We will now introduce a new element described in [16] which has good approximation properties in both cases. We will refer to this as the "robust" element, in lack of a better name. The polynomial space is defined by:

$$\boldsymbol{V}(T) = \{\boldsymbol{v} \in \boldsymbol{P}_3^2 : \nabla \cdot \boldsymbol{v} \in \boldsymbol{P}_0, \quad (\boldsymbol{v} \cdot \boldsymbol{n})|_e \in \boldsymbol{P}_1, \quad \forall E(T)\},$$

where $E(T)$ is the set of edges in triangulation. This element is designed such that the normal components on the faces are continuous, as is needed in $\boldsymbol{H}(\mathrm{div};\Omega)$. The tangential components are not continuous, but their mean value are. Hence, the element is conforming in $\boldsymbol{H}(\mathrm{div};\Omega)$, but non-conforming in $\boldsymbol{H}^1(\Omega)$. The element has nine basis functions and nine associated basis function nodes, $n_g = n_{bf} = n_b = 9$. The ξ_1 components of the basis functions in the reference element read,

$$
\begin{aligned}
N_1(\xi_1,\xi_2) &= 2\,\xi_1 + 9\,\xi_1\xi_2 - 7.5\,{\xi_1}^2 + 6\,{\xi_1}^3 - 18\,{\xi_2}^2\xi_1,\\
N_2(\xi_1,\xi_2) &= 6\,\xi_1 - 36\,\xi_1\xi_2 - 6\,{\xi_1}^2 + 36\,{\xi_2}^2\xi_1 + 24\,{\xi_1}^2\xi_2,\\
N_3(\xi_1,\xi_2) &= 6\,\xi_1 - 51\,\xi_1\xi_2 - 13.5\,{\xi_1}^2 + 6\,{\xi_1}^3 + 54\,{\xi_2}^2\xi_1 + 48\,{\xi_1}^2\xi_2,\\
N_4(\xi_1,\xi_2) &= \sqrt{2}\,(-7.5\,\xi_1 + 48\,\xi_1\xi_2 + 18\,{\xi_1}^2 - 9\,{\xi_1}^3 - 45\,{\xi_2}^2\xi_1 - 48\,{\xi_1}^2\xi_2),\\
N_5(\xi_1,\xi_2) &= \sqrt{2}\,(-3\,\xi_1 + 18\,\xi_1\xi_2 + 9\,{\xi_1}^2 - 6\,{\xi_1}^3 - 18\,{\xi_2}^2\xi_1 - 24\,{\xi_1}^2\xi_2),\\
N_6(\xi_1,\xi_2) &= \sqrt{2}\,(8.5\,\xi_1 - 36\,\xi_1\xi_2 - 24\,{\xi_1}^2 + 15\,{\xi_1}^3 + 27\,{\xi_2}^2\xi_1 + 48\,{\xi_1}^2\xi_2),\\
N_7(\xi_1,\xi_2) &= -0.5 - 7\,\xi_1 + 2\,\xi_2 + 27\,\xi_1\xi_2 + 25.5\,{\xi_1}^2 - 18\,{\xi_1}^3 - 18\,{\xi_2}^2\xi_1 - 48\,{\xi_1}^2\xi_2,\\
N_8(\xi_1,\xi_2) &= 6\,\xi_1 - 12\,\xi_1\xi_2 - 18\,{\xi_1}^2 + 12\,{\xi_1}^3 + 24\,{\xi_1}^2\xi_2,\\
N_9(\xi_1,\xi_2) &= 1.5 - 3\,\xi_1 - 2\,\xi_2 + 15\,\xi_1\xi_2 - 4.5\,{\xi_1}^2 + 6\,{\xi_1}^3 - 18\,{\xi_2}^2\xi_1,
\end{aligned}
$$

and the ξ_2 components are,

$$
\begin{aligned}
N_1(\xi_1,\xi_2) &= 1.5 - 2\,\xi_1 - 3\,\xi_2 + 15\,\xi_1\xi_2 - 4.5\,{\xi_2}^2 + 6\,{\xi_2}^3 - 18\,{\xi_1}^2\xi_2,\\
N_2(\xi_1,\xi_2) &= -6\,\xi_2 + 12\,\xi_1\xi_2 + 18\,{\xi_2}^2 - 12\,{\xi_2}^3 - 24\,{\xi_2}^2\xi_1,\\
N_3(\xi_1,\xi_2) &= 0.5 + 2\,\xi_1 - 7\,\xi_2 + 27.0\,\xi_1\xi_2 + 25.5\,{\xi_2}^2 - 18\,{\xi_2}^3 - 48\,{\xi_2}^2\xi_1 - 18\,{\xi_1}^2\xi_2,\\
N_4(\xi_1,\xi_2) &= \sqrt{2}\,(8.5\,\xi_2 - 36\,\xi_1\xi_2 - 24\,{\xi_2}^2 + 15\,{\xi_2}^3 + 48\,{\xi_2}^2\xi_1 + 27\,{\xi_1}^2\xi_2),\\
N_5(\xi_1,\xi_2) &= \sqrt{2}\,(-3\,\xi_2 + 18\,\xi_1\xi_2 + 9\,{\xi_2}^2 - 6\,{\xi_2}^3 - 24\,{\xi_2}^2\xi_1 - 18\,{\xi_1}^2\xi_2),\\
N_6(\xi_1,\xi_2) &= \sqrt{2}\,(-7.5\,\xi_2 + 48\,\xi_1\xi_2 + 18\,{\xi_2}^2 - 9\,{\xi_2}^3 - 48\,{\xi_2}^2\xi_1 - 45\,{\xi_1}^2\xi_2),\\
N_7(\xi_1,\xi_2) &= 6\,\xi_2 - 51\,\xi_1\xi_2 - 13.5\,{\xi_2}^2 + 6\,{\xi_2}^3 + 48\,{\xi_2}^2\xi_1 + 54\,{\xi_1}^2\xi_2,\\
N_8(\xi_1,\xi_2) &= -6\,\xi_2 + 36\,\xi_1\xi_2 + 6\,{\xi_2}^2 - 24\,{\xi_2}^2\xi_1 - 36\,{\xi_1}^2\xi_2,\\
N_9(\xi_1,\xi_2) &= 2\,\xi_2 + 9\,\xi_1\xi_2 - 7.5\,{\xi_2}^2 + 6\,{\xi_2}^3 - 18\,{\xi_1}^2\xi_2\,.
\end{aligned}
$$

The corresponding basis function nodes are,

$$
\begin{aligned}
q_1 &= (1/4, 0),\\
q_2 &= (2/4, 0),\\
q_3 &= (3/4, 0),\\
q_4 &= (3/4, 1/4),\\
q_5 &= (2/4, 2/4),\\
q_6 &= (1/4, 3/4),\\
q_7 &= (0, 3/4),\\
q_8 &= (0, 2/4),\\
q_9 &= (0, 1/4),
\end{aligned}
$$

We have the error estimates for the mixed Poisson equation,

$$\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{L}^2(\Omega)} \leq Ch^2 \|\boldsymbol{u}\|_{\boldsymbol{H}^2(\Omega)}, \tag{4.86}$$

$$\|\mathrm{div}\,(\boldsymbol{u} - \hat{\boldsymbol{v}})\|_{L^2(\Omega)} \leq Ch \|\mathrm{div}\,\boldsymbol{u}\|_{H^1(\Omega)}, \tag{4.87}$$

$$\|p - \hat{p}\|_{L^2(\Omega)} \leq Ch(\|p\|_{H^1(\Omega)} + h\|\boldsymbol{u}\|_{\boldsymbol{H}^2(\Omega)})\,. \tag{4.88}$$

The corresponding estimates for Stokes problem reads,

$$
\begin{aligned}
\|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{H}^1(\Omega)} &\leq Ch\|\boldsymbol{u}\|_{\boldsymbol{H}^2(\Omega)},\\
\|p - \hat{p}\|_{L^2(\Omega)} &\leq Ch(\|p\|_{H^1(\Omega)} + \|\boldsymbol{u}\|_{\boldsymbol{H}^2(\Omega)})\,.
\end{aligned}
$$

The element is implemented in Diffpack as `ElmT3gn9bn2Du` and `ElmT3gn9bn2Dv` and should be combined with the piecewise constant pressure elements, `ElmT3gn1bn2D`. Figure 4.7 shows the basis function nodes.

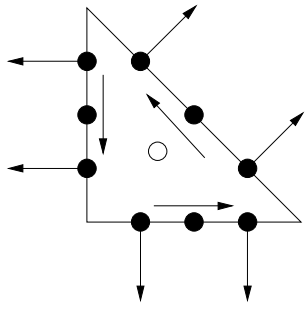

Fig. 4.7. The degrees of freedom of the robust element.

Remark: Stokes elements for the Poisson problem The mixed elements for the Poisson problem considered above were such that $\boldsymbol{v} \in \boldsymbol{H}(\text{div}; \Omega)$ and $p \in L^2(\Omega)$. One alternative approach is to seek an approximation where $\boldsymbol{v} \in \boldsymbol{L}^2(\Omega)$ and $p \in H^1(\Omega)$. Several of the above mentioned Stokes elements have continuous pressure elements and one might therefore suspect that these elements could be used. This is indeed true, at least for the Mini element and the Taylor–Hood element. Some numerical experiments in Section 4.6.5 show the behavior of these elements (c.f. also [16]). However, usually the highest level of accuracy is wanted in the velocity (or else one would not use the mixed formulation) and therefore this formulation is not often used.

4.4.3 Diffpack Implementation

The definition of the reference element is provided by a subclass of `ElmDef`. This definition is based on parameters (variables) in the base class `ElmDef` as well as virtual functions defined in `ElmDef`. For example, n_b, n_{bf} and n_g are integers in the class `ElmDef` having the names `nne_basis`, `nbf` and `nne_geomt`, respectively. The virtual function `geomtFunc` defines the geometry functions $G_i(\xi)$, while the virtual function `basisFunc` defines the basis functions $N_i(\xi)$. The derivatives $\partial N_i/\partial \xi_j$ and $\partial G_i/\partial \xi_j$ are provided by the virtual functions `dLocBasisFunc` and `dLocGeomtFunc`, respectively. Notice that the derivatives refer to the reference coordinates ξ. It is the derivatives with respect to the physical coordinates that are of interest. Class `BasisFuncAtPt` is designed for evaluating and storing the geometry and basis functions, their global derivatives and the (transposed) Jacobi matrix determinant, at a particular ξ point. This class relies on information from `ElmDef` and the global coordinates of the geometry nodes of the element.

Class `FiniteElement` is designed to offer the programmer easy access to the global element: the basis functions, their global derivatives, and the Jacobi determinant, evaluated at a point ξ in the reference domain Ω_r (corresponding to a physical point $\boldsymbol{x}$ in Ω_e). Class `FiniteElement` is naturally based on a layered design where it gains its information from `BasisFuncGrid`, `ElmDef`, `BasisFuncAtPt` and `GridFE` objects. In the case of mixed finite elements, one needs a `FiniteElement` object for each type of basis functions. This is provided by the class `MxFiniteElement`, which contains an array of pointers (handles) to `FiniteElement` objects and much of the same interface as class `FiniteElement`. Scalar finite element fields are represented by `FieldFE` and rely on information from `FiniteElement`, `BasisFuncGrid` and `GridFE` objects. `FieldFE` objects are designed to represent the solution field and can be evaluated and differentiated at arbitrary points.

The initial version of Diffpack was written for isoparametric elements. Hence, most of the virtual functions in the `ElmDef` have a default implementation in class `ElmDef` for isoparametric elements (if the function will depend on the element shape, the default version assumes a box shape).

For instance, the Mini element is implemented in a subclasses `ElmT3gn4bn2D`, which is a subclass of `ElmT3n2D`, which is a subclass of the base class for all elements, `ElmDef`. In this case `ElmT3n2D` supply the information related to the geometry of the element. All that was needed to implement `ElmT3gn4bn2D` was the basis functions and the location of the basis function nodes.

The information on the elements, in particular the geometry nodes, their connectivity and boundary indicators [9,4], is represented by class `GridFE` in Diffpack. Class `BasisFuncGrid` contains the basis function nodes, their connectivity, the boundary indicators at basis function nodes etc. In addition, class `BasisFuncGrid` has a `GridFE` object (or rather a pointer or handle to such an object) such that a `BasisFuncGrid` object has complete information of all the geometry and basis function nodes and their relevant associated data.

When working with isoparametric elements, class `BasisFuncGrid` simply uses the `GridFE` object for looking up information on basis function nodes and degrees of freedom in scalar fields. For a user it is then only necessary to create a `GridFE` object. When other classes are initialized with a `GridFE` object, and they need information on the basis functions (class `FieldFE` is an example), it is assumed that the elements are isoparametric and a `BasisFuncGrid` object is easily constructed internally in these classes. None of the terms related to the distinction between basis and geometry functions need to be familiar to the user in the isoparametric case. In particular, class `BasisFuncGrid` is not apparent at all, see for example [9].

If non-isoparametric elements are applied, the user must allocate and initialize a `BasisFuncGrid` for each scalar field. In this way, the extra complexity associated with the details of non-isoparametric elements is only visible when it is really needed. One should notice that this design goal is readily achieved due to our usage of abstract data types and object-oriented programming.

The `GridFE` and `BasisFuncGrid` classes organize the *global* topology of the geometry and basis function information. The (local) definition of the basis functions are provided by an `ElmDef` subclass.

4.4.4 Numbering Strategies

Having performed the mixed finite element discretization, we end up, as usual, with a system of linear algebraic equations. The book-keeping of element degrees of freedom and linear system degrees of freedom is non-trivial in mixed methods and is described in the following.

The special numbering. It is assumed that isoparametric elements are used and that there are m unknowns per node and a total of n nodes. In other words, one needs to use the same element type for all the unknown fields that enter a system of partial differential equations. Problems for which this is a suitable approach, involve the Navier equations of elasticity, the incompressible Navier-Stokes equations treated by a penalty function approach,

and simultaneous (implicit) solution of pressures and concentrations (saturations) in multi-phase porous media flow. The geometry and basis function nodes coincide, and it is natural to number the degrees of freedom of the linear system in this sequence:

$$u_1^1, u_1^2, \ldots, u_1^m, u_2^1, \ldots, u_2^m, \ldots, u_n^1, \ldots, u_n^m, \tag{4.89}$$

where u_i^j is the degree of freedom of scalar field no. j at node i. Given a global node i and a scalar field number j, the global degree of freedom number is $m(i-1)+j$. The `DegFreeFE` class in Diffpack takes care of computing the global degree of freedom number according to this formula.

At the element level, the structure of the special numbering is the same as at the global level. That is, if i is a local node and j is the field number, the associated *local* degree of freedom of the merged fields is $m(i-1)+j$. This information is very important since the local numbering is fundamental when setting up the elemental matrix and vector contributions in the `integrands` functions in Diffpack. The book [9] contains some relevant examples. As an example on the special numbering, consider the 2×2 grid of bilinear elements

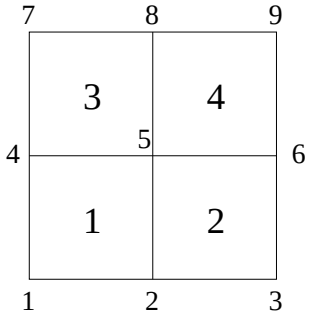

Fig. 4.8. Sketch of a 2×2 grid, with bilinear elements, and the corresponding numbering of elements and nodes.

in Figure 4.8. If there is only one scalar field, the degrees of freedom numbering in the linear system naturally follows the nodal numbering. A measure of the associated matrix bandwidth could be the largest difference between two degree freedom numbers in the same element. Here this is 4 (e.g. $5-1=4$ in the first element). With two unknown scalar fields, $u^{(1)}$ and $u^{(2)}$, there are two ways of structuring the degrees of freedom in the linear system. The suggestion above results in

$$u_1^1, u_1^2, u_2^1, u_2^2, \ldots, u_n^1, u_n^2 \tag{4.90}$$

and depicted in Figure 4.9. Instead of numbering the local degrees of freedom

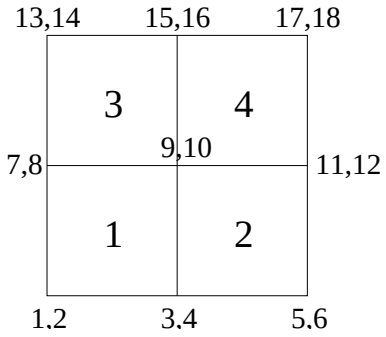

Fig. 4.9. The special numbering of degrees of freedom in the linear system arising from two unknown scalar fields over the grid in Figure 4.8.

at each node consecutively, we could first number all the degrees of freedom of scalar field one and then number the degrees of freedom of scalar field two:

$$u_1^1, u_2^1, \ldots, u_n^1, u_1^2, \ldots, u_n^2 \tag{4.91}$$

The numbering on a 2×2 grid is displayed in Figure 4.10. The impact of the

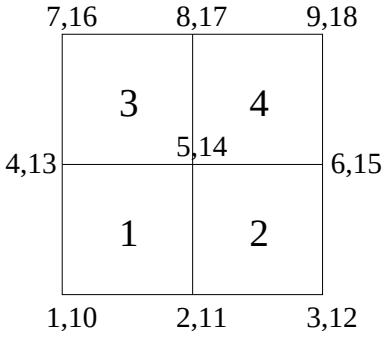

Fig. 4.10. A possible numbering of degrees of freedom in the linear system arising from two unknown scalar fields over the grid in Figure 4.8.

two numberings on the bandwidth of the coefficient matrix should be clear: 9 in the first case and 13 in the second case. On a grid with $q \times q$ elements the corresponding figures read $2q + 5$ and $q^2 + 3q + 3$!

The general numbering. The general numbering is based on the following strategy. At the element level, a field-by-field numbering like (4.91) is used for the element degrees of freedom,

$$u_1^1, u_2^1, \ldots, u_n^1, u_1^2, \ldots, u_n^2, \ldots, u_1^m, \ldots, u_n^m$$

where now n is the number of nodes in the element and m is the number of unknown scalar fields. More generally, if we have n_j degrees of freedom for field no. j, we order the unknowns like this:

$$u_1^1, u_2^1, \ldots, u_{n_1}^1, u_1^2, \ldots, u_{n_2}^2, \ldots, u_1^m, \ldots, u_{n_m}^m \, .$$

The corresponding *global* numbering is constructed from a simple (and general) algorithm that yields a reasonable small bandwidth: For each element we run through each local degree of freedom and increase the corresponding global number by one, provided the local degree of freedom has not been given a global number in a previously visited element. Such numbering of the degrees of freedom applied to a single scalar field is exemplified in Figure 4.11. In element 1 we go through the local nodes 1-4 and assign corresponding

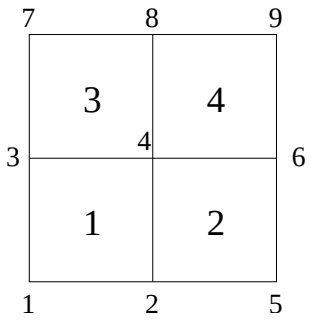

Fig. 4.11. The general numbering applied to a single scalar field over the grid in Figure 4.8.

global degree of freedom numbers 1-4. In element 2, the first local degree of freedom was already treated in element 1. The next local degree of freedom (local node no. 2) has not been treated before and can therefore be given the global degree of freedom number 5. The reader is encouraged to continue the algorithm and understand how the rest of the degree of freedom numbers arise.

Let us extend the general number example on a 2×2 grid to the case where we have two scalar fields as unknowns. In the first element we then run through the local degrees of freedom 1-4 of the first scalar field and generate corresponding global numbers 1-4. Then we run through the four degrees of freedom of the second scalar field and assign them to the global numbers 5-8. Proceeding with element two, the two nodes on the left have already been treated in element 2 so the second degree of freedom of scalar field no. 1 is assigned the global number 9, while the fourth degree of freedom of scalar field no. 1 corresponds to the global number 10. The second scalar field contributes with two new degrees of freedom, 11 and 12. Also in this case the

reader should understand the rest of the global degree of freedom numbers. Figure 4.12 shows the numbering.

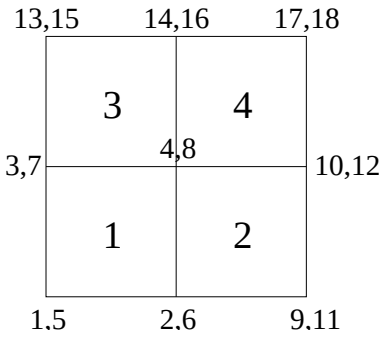

Fig. 4.12. The general numbering applied to two scalar fields over the grid in Figure 4.8.

Finally, we consider an example involving different number of degrees of freedom in different fields. Again we focus on the grid in Figure 4.8, but now we have two scalar fields with bilinear elements and one scalar field with piecewise constant elements. The location of the degree of freedom for the constant value in an element could be the centroid. (This example could correspond to bilinear elements for a vector field and piecewise constant elements for a scalar field.) Figure 4.13 shows the complete numbering.

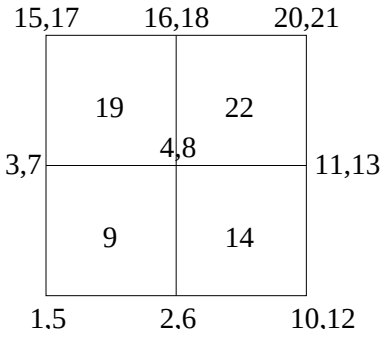

Fig. 4.13. The general numbering applied to two scalar fields with 9 nodes and one scalar field with 4 nodes over the grid in Figure 4.8.

As one can see, the general numbering requires quite some book-keeping. This is performed by the `DegFreeFE` object. However, the programmer must

explicitly deal with the *local* degrees of freedom numbering when setting up the element matrix and vector, but this is quite simple, as one applies either the special numbering or the field-by-field numbering on the element level. The complicated details arise when going from the element to the global degrees of freedom numbering, but class `DegFreeFE` hides the book-keeping from the programmer. Some program examples appear later and illustrates the usage of various Diffpack tools.

4.5 Some Code Examples

In this Section we will build simulators appropriate for mixed systems, step by step. We begin with a simple demonstration program for the numbering schemes in Section 4.4.4 for scalar and vector fields. Then the usage of `BasisFuncGrid` and special vs. general numbering in a `Poisson1`-like (cf. [9]) solver is shown. The next step concerns a simple version of a Stokes-problem solver before we end up with a sophisticated block/multigrid solver for the Stokes problem.

4.5.1 A Demo Program for Special Versus General Numbering

Section 4.4.4 presents a series of examples, Figures 4.8–4.13, on various degree of freedom numbering strategies. Here we shall make a simple demo program that produces these numberings. The program is found in

```
$NOR/doc/mixed/src/numbering/numbering.cpp
```

Assume we have a finite element grid stored in a `GridFE` object `grid`. Making a `DegFreeFE` object directly from the grid, with one unknown per node,

```
DegFreeFE dof (grid, 1);
```

results in a standard numbering of the unknowns, which coincides with the nodal numbering (cf. Figure 4.8). If we have two fields over the grid, i.e., two unknowns per node, we just alter the second parameter to the `DegFreeFE` constructor:

```
DegFreeFE dof (grid, 2);
```

The resulting degree of freedom numbering corresponds to the special numbering (cf. Figure 4.9).

The general numbering requires a slightly different initialization procedure of the `DegFreeFE` object. Now we must explicitly define a `BasisFuncGrid` for the basis functions over the grid and tie a finite element field to this `BasisFuncGrid` object:

```
BasisFuncGrid f_grid (grid);
FieldFE f (f_grid, "f");
```

Notice that the field will here be isoparametric. Having the field `f` we can initialize the `DegFreeFE` object:

```
DegFreeFE dof (f);
```

The corresponding numbering of the degrees of freedom is shown in Figure 4.11. Notice that sending a field, instead of a grid, as argument to a `DegFreeFE` constructor implies the general numbering of the degrees of freedom.

With two scalar fields, having the same number of degrees of freedom, the fields must be attached to a `FieldsFE` collector prior to initializing the degree of freedom handler:

```
BasisFuncGrid u_grid (grid);
FieldFE u (u_grid, "u");
FieldFE v (u_grid, "v");
FieldsFE collection (2, "coll");
collection.attach (u, 1);
collection.attach (v, 2);

DegFreeFE dof (collection);
```

The degree of freedom numbering is depicted in Figure 4.12.

Our final example concerns two fields with isoparametric elements, like `u` and `v` above, plus one field having (possibly) non-isoparametric elements. In other words, we address the general case with several fields and different elements in the different fields. The set-up is the same:

```
BasisFuncGrid u_grid (grid);
FieldFE u (u_grid, "u");
FieldFE v (u_grid, "v");

BasisFuncGrid p_grid (grid);
p_grid.setElmType (p_elm);   // can change to non-isoparametric elm
FieldFE p (p_grid, "p");

FieldsFE collection (3, "coll");
collection.attach (u, 1);
collection.attach (v, 2);
collection.attach (p, 3);

DegFreeFE dof (collection);
```

Figure 4.13 displays the associated degree of freedom numbering.

4.6 Programming with Mixed Finite Elements in a Simulator

Programming with mixed finite elements is closely related to programming with isoparametric elements. The structure of the class that represents the simulator is the same in the two cases. The basic differences are

- Class `FEM` is replaced by its subclass `MxFEM`.
- Class `FiniteElement` is replaced by `MxFiniteElement`, which contains a set of `FiniteElement` objects, each corresponding to the element type of a field involved in the partial differential equations being solved.
- `integrands` is replaced by

```
void integrandsMx (ElmMatVec& elmat, MxFiniteElement& mfe)
```

- Similarly, `integrands4side` is replaced by `integrands4sideMx`, `makeSystem` is replaced by `makeSystemMx` (with a slightly different argument list).
- The programmer must explicitly declare a `BasisFuncGrid` or

```
VecSimplest(BasisFuncGrid)
```

  for each field type that corresponds to a primary unknown.
- All the `BasisFuncGrid` objects must be explicitly created and initialized by the programmer (but the initialization is trivial, just call a function with the element type for each field). The creation of `DegFreeFE`, `FieldFE` and `FieldsFE` objects must be based on a `BasisFuncGrid` object as input rather than on a straight `GridFE` object.

4.6.1 A Standard Solver with Some New Utilities

Consider the `Poisson1` class from [9, ch. 3.1]. Now we change the test problem a bit such that we have a numerical solution that coincides with the exact solution regardless of the number of elements. The test problem for this purpose reads $\nabla^2 u = 0$ in $[0,1]^d$, with $\partial u/\partial n = 0$ on the boundary, except at $x_1 = 0$ where $u = 0$ and at $x_1 = 1$ where $u = 1$. The exact solution is then given as $u(x_1, \ldots, x_d) = x_1$. This should be reproduced by any grid.

The test problem is implemented in class `Laplace1` whose directory is found in `$NOR/doc/mixed/src`. This class is a slight edit of `Poisson1` where we basically have thrown away the flux computations, fixed the domain, and changed the analytical solution and the `fillEssBC` function. Of course, the `define` and `scan` functions are significantly changed.

Our first task is to extend class `Laplace1` to allow for a general node numbering, just for the purpose of explaining how one introduces `BasisFuncGrid` objects in a solver and how the `DegFreeFE` object must be initialized. The following modifications to class `Laplace1` must be performed:

- inclusion of a `BasisFuncGrid` object in the class:

```
Handle(BasisFuncGrid) bgrid;
```

- definition of a menu item for the type of numbering

– creation of the unknown field from the `BasisFuncGrid` object[1]:

```
bgrid.rebind (new BasisFuncGrid (*grid));
u.rebind (new FieldFE (*bgrid, "u"));
```

– creation of the `DegFreeFE` object in two ways, depending on whether the user has chosen the special or general numbering:

```
if (numbering == "special")
  dof.rebind (new DegFreeFE (*grid, 1));
else
  dof.rebind (new DegFreeFE (*u));
```

– of reasons to be explained later, the `fillEssBC` function needs to be rewritten

These extra statements are conveniently placed in a subclass of `Laplace1`:

```
class Laplace1GN : public Laplace1
{
public:
  Handle(BasisFuncGrid) bgrid;
  Laplace1GN () : Laplace1() {}
 ~Laplace1GN () {}
  virtual void define (MenuSystem& menu, int level = MAIN);
  virtual void scan ();
  virtual void fillEssBC ();
};
```

The source code is placed in a subdirectory `Laplace1GN` of `Laplace1`[2].

The redefinition of `fillEssBC` is perhaps not obvious. If we use the familiar constructions for setting boundary conditions, e.g.,

```
if (grid->boNode (i, 1))     // i=geometry node, bo.ind.=1
  dof->fillEssBC (i, 1.0);
```

a wrong solution is computed in the case of a general numbering. The reason is that using the node counter `i` in `dof->fillEssBC(i,1.0)` *implies that the numbering of the unknowns in the linear system coincides with the numbering of the geometry nodes*. Instead of the node number `i` we should use the *global degree of freedom number* corresponding to this node[3]. This number is in general computed by

```
idof = dof->fields2dof (i, 1);
```

[1] The field has a `BasisFuncGrid` which contains a `GridFE` object, making the access to all grid information complete from a field object.

[2] We have run `AddMakeSrc ..` to tell the makefile for `Laplace1GN` that the source depends on files in the parent directory.

[3] The numbering of the geometry nodes and the linear system degrees of freedom with a general numbering differ even when there is only one unknown scalar field.

Moreover, we should for the case of generality in the non-isoparametric case ask `BasisFuncGrid` if a basis function node is subject to the boundary condition:

```
if (bgrid->essBoNode (i, 1))
  dof->fillEssBC (idof, 1.0);
```

With isoparametric elements this can be simplified to

```
if (grid->boNode (i, 1))
  dof->fillEssBC (idof, 1.0);
```

i.e. asking the geometry nodes for essential boundary conditions. (If there is only one unknown per node, we have `idof=i`, of course.)

The next natural extension of class `Laplace1GN` is `Laplace1Mx` that allow for the mixed versions of `FEM` and `FiniteElement`. This class is essentially a merge of classes `Laplace1` and `Laplace1GN`, where `FEM` and `FiniteElement` are replaced by `MxFEM` and `MxFiniteElement`. In addition, we need a `FieldsFE` collection of all fields that enter as unknowns in the linear system. Notice that changing the elements even in the standard formulation of the Poisson problem has an effect. The Crouzeix-Raviart element is often used in to compute the pressure in porous media flow, because of better results than standard elements.

The new parts of the code contain the construction of a `Handle(FieldsFE) collection` object, which is used in many of the mixed finite element utilities:

```
collection.rebind (new FieldsFE (1 /*no of fields*/, "collection"));
collection->attach (*u, 1);
dof.rebind (new DegFreeFE (*collection));
```

The `collection` object is needed when making the linear system:

```
makeSystemMx (*dof, *lineq, *collection, 1);
```

The final argument `1` represents the field number in `collection` whose grid should be used for the numerical integration. (Usually, this should be the field that has the `BasisFuncGrid` with the highest order elements).

The `integrands` routine is replaced by `integrandsMx` and works with an array of `FiniteElement` objects, one entry for each unknown field in the `collector` collection of fields. Here things are as simple as possible, i.e., only one unknown field.

```
void Laplace1Mx:: integrandsMx (ElmMatVec& elmat,
    const MxFiniteElement& mfe)
{
  int i,j,q;
  if (mfe.size() != 1)
    errorFP("Laplace1Mx:: integrandsMx",
            "Wrong size of MxFiniteElement");

  const FiniteElement& fe = mfe(1);
  const real detJxW = mfe.detJxW();
  ...
}
```

The rest of the routine is identical to `Laplace1::integrands`.

4.6.2 A Simulator for the Stokes Problem

The linear system arising from the Stokes problem can be written on the form (4.19)–(4.20). In our first implementation of the Stokes problem we shall assemble a single merged linear system, suitable for direct methods. The linear system is then not of the form (4.19)–(4.20), because the degrees of freedom and equations are merged according to the general numbering. However, at the element level, the discrete equations are on the form (4.19)–(4.20) with n_v being the number of velocity nodes and n_p the number of pressure nodes in an element.

The simulator is realized as a class `Stokes`. Its basic contents are, as usual, handles to a `GridFE`, `LinEqAdmFE`, `DegFreeFE`, `FieldFE`, and `SaveSimRes` objects. In addition we need some new data structures for mixed methods: `BasicFuncGrid` objects for specifying basis function nodes in mixed elements and `FieldsFE` objects for collecting unknown fields for book-keeping of the global set of degrees of freedom in the `DegFreeFE` object.

An outline of the class `Stokes` is listed next.

```
class Stokes : public MxFEM
{
protected:
  Handle(GridFE)        grid;    // underlying finite element grid
  Handle(BasisFuncGrid) v_grid, p_grid;
  Handle(FieldFE)       p;              // Pressure
  Handle(FieldsFE)      v;              // Velocity
  Handle(FieldsFE)      coll;           // (v_1, ..., v_d, p)
  Handle(DegFreeFE)     dof;
  Vec(real)             linsol;
  Handle(LinEqAdmFE)    lineq;
  Handle(SaveSimRes)    database;

  Ptv(real) v_inlet;        // velocity at inlet
  Test_case test_case;

  Handle(FieldFE)       error;

  // used to partition scan into manageable parts:
  void scanGrid();
  virtual void scanData();
  virtual void initFieldsEtc();
  virtual void numbering();

  // standard functions:
  virtual void fillEssBC ();
  virtual void fillEssBC4V (DegFreeFE& dof);
  virtual void fillEssBC4P (DegFreeFE& dof, int P);

  virtual void calcElmMatVecMx
    (int e, ElmMatVec& elmat, MxFiniteElement& mfe);

  virtual void integrandsMx
    (ElmMatVec& elmat, const MxFiniteElement& mfe);

void makeSystemMx
```

```
  (
   DegFreeFE&  dof,
   LinEqAdmFE& lineq,
   FieldsFE&   all_unknowns,
   int         itgrule_manager,     // one of the all_unknowns components
   bool        compute_A = true,
   bool        compute_RHS = true
  );


public:

  Handle(FieldsFunc)    usource;
  Stokes ();
 ~Stokes ();

  virtual void adm (MenuSystem& menu);
  virtual void define (MenuSystem& menu, int level = MAIN);
  virtual void defineData (MenuSystem& menu, int level = MAIN);
  virtual void scan ();

  virtual void solveProblem ();
  virtual void saveResults ();
  virtual void resultReport ();
};
```

The `coll` field contains all the unknown scalar fields and is required in some mixed finite element routines. The `v` field is simply the velocity field as a *vector field* suitable for dumping to a simres database and used later for visualization purposes.

The first non-trivial part of the code in class `Stokes` concerns initialization of the data structures. First we compute the grid, then the basis function grids, followed by the fields, then the degree of freedom handler, and finally the linear system. The element used in each basis function grid can be set on the menu. A typical initialization of a basis function grid object is then like

```
v_grid.rebind (new BasisFuncGrid (*grid));
String v_elm_tp = menu.get ("v element");
v_grid->setElmType (v_grid->getElmType(), v_elm_tp);
```

with similar code for `p_grid`. The string `v element` is read from the input file, where the Crouzeix-Raviart velocity elements and the constant pressure element is specified as,

```
set u element = ElmT3gn3bn2D
set p element = ElmT3gn1bn2D
```

The field objects for the velocity and the pressure are created from the basis function grids:

```
v.rebind (new FieldsFE (*v_grid, nsd, "v"));
p.rebind (new FieldFE (*p_grid, "p"));
```

The next step is to create a collector of all unknown vector and scalar fields.

```
coll.rebind (new FieldsFE (nsd+1,"collector"));
for (int k=1; k<=nsd; k++)
  coll->attach (v()(k), k);
coll->attach (*p, nsd+1);
dof.rebind (new DegFreeFE (*coll));
```

In the last line the `dof` is made from the `FieldsFE` object `coll`. This causes the general numbering to be used.

A central function is `fillEssBC`. The following code segment assigns the essential boundary conditions (by calling up functors for the exact $\boldsymbol{v}$ at the boundary):

```
const int v_nno = v_grid->getNoNodes();
int   idof, i;
Ptv(real) x (nsd), values(nsd);
dof.initEssBC ();

// boundary indicators: Assume the default ones from PreproBox

// we assume that the v_x and v_y basis func nodes coincides such
// that we can use the same loop for all velocity components

for ( i=1; i<= v_nno; i++) {
  if ( v_grid->essBoNode(i)) {
    v_grid->getCoor(x,i);
    v_anal->valuePt(values,x);
    for (int d=1; d<= nsd; d++) {
      idof = dof.fields2dof(i,d);
      dof.fillEssBC(idof, values(d));
    }
  }
}
```

In this example we have splitted the `fillEssBC` function in two functions, `fillEssBC4V` and `fillEssBC4P`. The reason is that we want to reuse these functions in the block solvers [15] that are derived from this class. The statements should be obvious from the previous material on the `Laplace1GN` and `Laplace1Mx` solvers.

The heart of any Diffpack finite element simulator is the `integrands` function. In the present case this routine is more complicated than in scalar PDE problems. The discrete equations (4.19)–(4.20) interpreted at the element level have the suitable form for direct implementation in an `integrands` routine, as we have already stated in Section 4.4.4 that the numbering of equations and degrees of freedom at the element level is always of the field-by-field form. We exemplify this in 2D, while the following code also works in 3D. The element equations consist of n_v equations from the x components of the equation of motion, then n_v equations from the y component (these two correspond to $r = 1$ and $r = 2$ in (4.19)). The last n_p equations stem from the continuity equation (4.20). The unknowns are the n_v nodal values

of $\hat{v}^1$, the n_v nodal values of $\hat{v}^2$, and the n_p nodal values of $\hat{p}$. The element equations can be partitioned like this in 2D:

$$\begin{pmatrix} \boldsymbol{A} & 0 & \boldsymbol{B}^1 \\ 0 & \boldsymbol{A} & \boldsymbol{B}^2 \\ (\boldsymbol{B}^1)^T & (\boldsymbol{B}^2)^T & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_x \\ \boldsymbol{v}_y \\ \boldsymbol{p} \end{pmatrix} = \begin{pmatrix} \boldsymbol{c}^{(1)} \\ \boldsymbol{c}^{(2)} \\ \boldsymbol{0} \end{pmatrix},$$

with $\boldsymbol{A} = \{A_{ij}\}$, $\boldsymbol{B}^r = \{B^r_{ij}\}$, $(\boldsymbol{B}^r)^T = \{B^r_{ji}\}$ and $\boldsymbol{c}^i = \{c^i_i\}$. In `integrands` it is convenient to generate four block matrices: the upper left matrix, corresponding to the Laplace term $\nabla^2\boldsymbol{v}$,

$$\begin{pmatrix} \boldsymbol{A} & 0 \\ 0 & \boldsymbol{A} \end{pmatrix}.$$

The upper right matrix, corresponding to the pressure term $-\nabla p$,

$$\begin{pmatrix} \boldsymbol{B}^1 \\ \boldsymbol{B}^2 \end{pmatrix}.$$

The lower left matrix, corresponding to the divergence term $\nabla \cdot \boldsymbol{v}$,

$$\begin{pmatrix} (\boldsymbol{B}^1)^T & (\boldsymbol{B}^2)^T \end{pmatrix}.$$

Finally, the lower right matrix, which is $\boldsymbol{0}$. With the above formulas we are ready to present the gory details of the `integrands` function:

```
void Stokes:: integrandsMx (ElmMatVec& elmat, const MxFiniteElement& mfe)
{
  const int nsd   = mfe.getNoSpaceDim();
  const int nvxbf = mfe(1).getNoBasisFunc();
  const int nvbf  = nvxbf*nsd;
  const int npbf  = mfe(nsd+1).getNoBasisFunc();
  const real detJxW = mfe.detJxW();
  real  nabla;
  Ptv(NUMT) values(nsd);
  Ptv(NUMT) x(nsd);
  mfe.getGlobalEvalPt(x);
  if (usource.ok()) usource->valuePt(values,x,DUMMY);
  else values.fill(0.0);
  int   i,j,k,d,s;
  int ig,jg;

  // upper left block matrix, the term -Laplace(u)*mfe(U).N(i)

  for (d = 1; d <= nsd; d++){
    for (i = 1; i <= nvxbf; i++) {
      ig = (d-1)*nvxbf+i;
      for (j = 1; j <= nvxbf; j++) {
        jg = (d-1)*nvxbf+j;
        nabla = 0;
        for (k = 1; k <= nsd; k++)
          nabla += mfe(d).dN(i,k)*mfe(d).dN(j,k);
        elmat.A(ig,jg) += nabla*detJxW;
```

```
      }
      elmat.b(ig) += values(d)*mfe(d).N(i)*detJxW;
    }
  }

  // upper right block matrix, the term (dp/dx)*mfe(U).N(i)
  for (s = 1; s <= nsd; s++)
    for (i = 1; i <= nvxbf; i++) {
      ig = (s-1)*nvxbf+i;
      for (j = 1; j <= npbf; j++) {
        jg = nvbf+j;
        elmat.A(ig,jg) -= mfe(s).dN(i,s)*mfe(nsd+1).N(j)*detJxW;
      }
    }

  // lower left block matrix, the term du/dx*mfe(P).N(i),
  // eq.no. nubf+i
  for ( d=1 ; d<= nsd; d++)
    for (i = 1; i <=  npbf; i++) {
      ig = nvbf+i;
      for (j = 1; j <= nvxbf; j++) {
        jg = (d-1)*nvxbf+j;
        elmat.A(ig,jg) -= mfe(nsd+1).N(i)*mfe(d).dN(j,d)*detJxW;
      }
    }
}
```

Notice that both `mfe(d)` hold the N_i functions and their derivatives (because `d <= nsd`), whereas `mfe(nsd+1)` holds the L_i functions. The program administration takes place in `solveProblem`:

```
void Stokes:: solveProblem ()
{
  fillEssBC ();
  makeSystemMx (*dof, *lineq, *coll, 1);
  linsol.fill(0.0);
  lineq->solve();
  dof->vec2field (linsol, *coll);

  database->dump(*v);
  database->dump(*p);
}
```

The contents of `solveProblem` follows the set-up from the examples in [9], except that we call `makeSystemMx` and use the `coll` collection of fields, containing both `v` and `p`, both in `makeSystemMx` and when storing the solution of the linear system (`linsol`) back in the fields (`dof->vec2field`).

At last, we compute the L_2, L_1 and L_∞ norms of the error in both velocity and pressure. This is accomplished with some functions defined in `ErrorNorms`,

```
ErrorNorms::Lnorm (*v_anal,     // supplied function (see above)
                   *v,          // numerical solution
                   DUMMY,         // point of time
                   L1_error, L2_error, Linf_error, // error norms
                   GAUSS_POINTS); // point type for numerical integr.
```

```
s_o <<oform("L2 error of v=%12.5e \n", L2_error);
ErrorNorms::Lnorm (*p_anal,       // supplied function (see above)
                   *p,            //  numerical solution
                   DUMMY,         // point of time
                   L1_error, L2_error, Linf_error, // error norms
                   GAUSS_POINTS); // point type for numerical integr.

s_o <<oform("L2 error of p=%12.5e \n", L2_error );
```

The complete source code is found in

```
$NOR/doc/mixed/src/Stokes
```

4.6.3 Numerical Experiments for the Stokes Simulator

In this section we investigate the behaviour of the error in terms of the mesh size h. We manufacture a non-polynomial smooth solution, simply by chosing some solutions $(\boldsymbol{u}, p)$ and then compute $\boldsymbol{f}$. Low-order polynomial solutions are avoided because these often lead to super convergence. Notice that $\boldsymbol{u}$ must be divergence free and that $\int_\Omega p\, d\Omega = 0$. The manufactured solutions and the corresponding data read,

$$\begin{aligned}
\boldsymbol{u} &= \begin{pmatrix} -sin(xy)x \\ sin(xy)y \end{pmatrix}, \\
p &= cos(xy) - a, \\
a &= \int_\Omega cos(xy)\, d\Omega, \\
\mu &= 1, \\
\boldsymbol{f} &= -\Delta \boldsymbol{u} + \nabla p\,.
\end{aligned}$$

We use this solution instead of a physical relevant problem, to verify the implementation. It is recommended to always do such study of error behavior to eliminate bugs before advancing to more challenging physical problems. We measure the errors,

$$\varepsilon^{\boldsymbol{v}}_{L^2(\Omega)} = \|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{L^2(\Omega)}, \tag{4.92}$$

$$\varepsilon^{p}_{L^2(\Omega)} = \|p - \hat{p}\|_{L^2(\Omega)}\,. \tag{4.93}$$

$$\tag{4.94}$$

The estimated errors are listed in Table 4.1. The Crouzeix-Raviart element and the Mini element show second order accuracy for the velocity and first order accuracy for the pressure, as expected from (4.76)-(4.77) and (4.79)-(4.80). The Taylor-Hood elements are better for smooth solutions, we get third order for the velocity and second order for the pressure, again in complete agreement with (4.69)-(4.71). The results listed below are made with

$\hat{\boldsymbol{v}}$-element	error$\backslash h$	2^{-2}	2^{-3}	2^{-4}	2^{-5}
Crouzeix-Raviart	$\varepsilon^{\boldsymbol{v}}_{L^2(\Omega)}$	2.68e-3	7.00e-4	1.78e-4	4.47e-5
Crouzeix-Raviart	$\varepsilon^{p}_{L^2(\Omega)}$	2.18e-2	9.75e-3	4.53e-3	2.19e-3
Taylor-Hood	$\varepsilon^{\boldsymbol{v}}_{L^2(\Omega)}$	3.52e-4	4.39e-5	5.49e-6	6.85e-7
Taylor-Hood	$\varepsilon^{p}_{L^2(\Omega)}$	2.06e-3	4.66e-4	1.14e-4	2.85e-5
Mini	$\varepsilon^{\boldsymbol{v}}_{L^2(\Omega)}$	1.63e-3	3.98e-4	9.84e-5	2.45e-5
Mini	$\varepsilon^{p}_{L^2(\Omega)}$	4.59e-2	1.55e-2	5.51e-3	1.96e-3

Table 4.1. Error obtained with different elements.

the solver described in [15]. That is, we have used the `SymMinRes` solver with a multigrid block preconditioner. It is worth noting that using a Krylov method, like `SymMinRes`, without a preconditioner leads to *very* slow convergence. Moreover, a standard preconditioner, e.g., RILU, actually often results in breakdown, because the system is indefinite. Instead block preconditioners, together with multigrid techniques from [14], should be used. In fact, optimal preconditioners are constructed in [15].

4.6.4 A Simulator for the Mixed Poisson Problem

We will now describe a mixed Poisson simulator, which is very similar to the Stokes simulator, but where we need to use some other elements. We have mentioned earlier that the Raviart-Thomas element and the robust element are implemented in Diffpack. Typical Stokes elements can also be used, but then lower accuracy of the velocity is obtained. The $\boldsymbol{H}(\mathrm{div};\Omega)$ elements are *vector elements*, in the sense that the degrees of freedom correspond to vectors. These elements require special initialization. Therefore, we use a boolean variable `special_mapping` to distinguish vector elements from the other elements. The initialization of the velocity fields reads,

```
if (special_mapping) {
  u_x.rebind (new FieldFE (*u_bx, "u_x"));
  u_y.rebind (new FieldFE (*u_by, u_x->values (), "u_y"));
} else {
  u_x.rebind (new FieldFE (*u_bx, "u_x"));
  u_y.rebind (new FieldFE (*u_by, "u_y"));
}
```

Hence, in the case of vector elements, the same vector of unknowns, `u_x->values()`, is used both in the `u_x` field and the `u_y` field. However, these fields can be used as usual, e.g., a collection of all the fields is made as:

```
fields_all.rebind (new FieldsFE (3, "u-collector"));
fields_all->attach (*u_x, 1);
fields_all->attach (*u_y, 2);
fields_all->attach (*p, 3);
```

The `fields_all` object is suitable for post-processing. In the simulator for the Stokes problem we based the `DegFreeFE` on `coll` which is similar to the `fields_all` object made here. However, because `u_x` and `u_y` share the same unknowns we would then end up with too many degrees of freedom. Instead only one of the velocity fields is used to initialize the `DegFreeFE`.

```
fields_numbering.rebind (new FieldsFE (2, "u-collector"));
fields_numbering->attach (*u_x, 1);
fields_numbering->attach (*p, 2);
dof_numbering.rebind (new DegFreeFE (*fields_numbering));
```

The degrees of freedom in the Raviart-Thomas and the robust element are vectors (normal and tangential) rather than scalar values, and therefore the `fillEssBC` routine is slightly different. We need to know the normal direction on a side, as well as the value in a given point. To achieve this we make a loop that goes through all local nodes in all elements,

```
void PoissonMx:: fillEssBC4V (DegFreeFE& dof) {
... // initialization
  if (special_mapping) {

    for (e=1; e<= no_elms; e++) {
      nl = v_grid.getNoNodesInElm(e);
      for (l=1; l<= nl; l++) {
        idof = dof.loc2glob(e,l);
        dof.dof2fields(idof, i, f );
        if (v_grid.essBoNode (i)) {
           v_grid.getCoor (x, i);
           dof.fillEssBC (idof, uanalcomp->valuePt (x,e,l));
        }
      }
    }
```

The element number and the local node can be used to compute the normal vector by `MxMapping`. This is done in the `uAnalVecComp` functor holding a pointer to the analytical solution `uAnal` functor `anal_funcs`. The `uAnal` contains the analytical solution in vector form, while the `uAnalVecComp` computes the value in either the normal or tangential direction. The code reads,

```
real uAnalVecComp:: valuePt (const Ptv (real) & p,
                int elm_no, int loc_node, real t ) {
  Ptv (real) values;
  anal_funcs->valuePt (values, p);
  MxFiniteElement mfe (data->fields_all ());
  mfe.setItgRuleManager (1);
  mfe.refill (elm_no);
  MxMapping & mxmapping = mfe.map ();
  Ptv(real) dummy(2); dummy = 0;
  Ptv(real)& vector = dummy;

  if (data->special_mapping) {
    if (data->u_bx->getElmType () == "ElmT3gn3bn2Du") {
      int side = loc_node;
      vector = mxmapping.getNormalVector (side);
       return values.inner(vector);
```

The `integrandsMx` function also needs to be modified slightly. When using vector elements, the basis functions `mfe.N(1,i)` and `mfe.N(2,i)` correspond to the same entry in the element matrix. We manage this by using an additional integer `D`, which is set to one in case of vector elements, while it is set to `d` (as usual) for standard elements. The $(\text{div}\,\hat{\boldsymbol{v}}, L_i)$ term then reads,

```
// lower left block matrix, the term du/dx*mfe(P).N(i), eq.no. nubf+i
 for ( d=1 ; d<= nsd; d++) {
   if ( special_mapping ) D = 1;
   else D = d;
   for (i = 1; i <=  npbf; i++)
     for (j = 1; j <= nuxbf; j++)
       elmat.A(nUbf+i,(D-1)*nuxbf+j) += mfe.N(3,i)*mfe.dN(d,j,d)*detJxW;
 }
```

A suitable `makeSystemMx` is,

```
makeSystemMx (*dof_numbering, *lineq, *fields_all, 1);
```

This variant of `makeSystemMx` makes a matrix based on the numbering of `dof_numbering`, but base the mixed elements on `fields_all`.

We can estimate the error with a varity the functions in `ErrorNorms`. Functions suitable for our purpose are,

```
ErrorNorms::HdivNorm(*uanal, *fields_u, Hdiv_error_norm,
            L2_part_error, div_part_error, inf_error, DUMMY, false);
ErrorNorms::Lnorm(*panal, *p, DUMMY, L1_error_of_p,
            L2_error_of_p, Linf_error_of_p);
```

The complete source code is found in

```
$NOR/doc/mixed/src/PoissonMx
```

4.6.5 Numerical Experiments for the Mixed Poisson Simulator

To verify the program we construct a simple problem with a known analytical solution,

$$\lambda = 1, \tag{4.95}$$

$$p = cos(x * y), \tag{4.96}$$

$$\boldsymbol{v} = \nabla p, \tag{4.97}$$

$$f = \nabla \cdot \boldsymbol{v}\,. \tag{4.98}$$

We have changed the sign of p to obtain symmetry. Krylov solvers that utilize the symmetry, like `SymMinRes` and `Symmlq`, are much more efficient than the methods that do not, e.g, `BiCGStab` and `Orthomin`. In addition to the norms of the errors in (4.92)-(4.93), we also measure the $\boldsymbol{H}(\text{div};\Omega)$ norm,

$$\varepsilon^{\boldsymbol{v}}_{\boldsymbol{H}(div;\Omega)} = \|\boldsymbol{u} - \hat{\boldsymbol{v}}\|_{\boldsymbol{H}(div;\Omega)}, \tag{4.99}$$

The Table 4.2 shows the errors associated with the Raviart-Thomas element, the robust element and the Mini element. Both the Raviart-Thomas element and the robust element are designed for this problem and we get linear convergence of the velocity error in $\boldsymbol{H}(\mathrm{div};\Omega)$ and $\boldsymbol{L}^2(\Omega)$, whereas the pressure converges linearly in $L^2(\Omega)$, as expected from (4.84)-(4.85) and (4.86)-(4.88). On the other hand, the Mini element approximation does not converge in $\boldsymbol{H}(\mathrm{div};\Omega)$. However, the velocity shows linear convergence in $L^2(\Omega)$ and we have quadratic convergence of the pressure in $L^2(\Omega)$ (c.f [16]).

$\hat{\boldsymbol{v}}$-element	error$\backslash h$	2^{-2}	2^{-3}	2^{-4}	2^{-5}
Raviart-Thomas	$\varepsilon^{\boldsymbol{v}}_{\boldsymbol{H}(div;\Omega)}$	1.00e-1	5.05e-2	2.53e-2	1.26e-2
Raviart-Thomas	$\varepsilon^{\boldsymbol{v}}_{L^2(\Omega)}$	6.72e-2	3.36e-2	1.68e-2	8.39e-3
Raviart-Thomas	$\varepsilon^{p}_{L^2(\Omega)}$	1.60e-2	7.69e-3	3.74e-3	1.85e-3
Robust	$\varepsilon^{\boldsymbol{v}}_{\boldsymbol{H}(div;\Omega)}$	6.41e-2	3.22e-2	1.62e-2	8.08e-3
Robust	$\varepsilon^{\boldsymbol{v}}_{L^2(\Omega)}$	5.81e-3	1.49e-3	3.79e-4	9.65e-5
Robust	$\varepsilon^{p}_{L^2(\Omega)}$	1.48e-2	7.34e-3	3.66e-3	1.83e-3
Mini	$\varepsilon^{\boldsymbol{v}}_{\boldsymbol{H}(div;\Omega)}$	1.26e+0	1.32e+0	1.35e+0	1.36e+0
Mini	$\varepsilon^{\boldsymbol{v}}_{L^2(\Omega)}$	4.82e-2	2.42e-2	1.22e-2	6.09e-3
Mini	$\varepsilon^{p}_{L^2(\Omega)}$	2.89e-2	8.75e-3	2.58e-3	7.42e-4

Table 4.2. Error obtained with different elements.

Acknowledgements

The authors are grateful to Professor R. Winther for many useful discussions.

References

1. D.N. Arnold, F. Brezzi, and M. Fortin. A stable finite element method for the stokes equations. *Calcolo*, 1984.
2. S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer-Verlag, 1994.
3. F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Methods*. Springer-Verlag, 1991.
4. A. M. Bruaset and H. P. Langtangen. A comprehensive set of tools for solving partial differential equations; Diffpack. In M. Dæhlen and A. Tveito, editors, *Mathematical Models and Software Tools in Industrial Mathematics*, pages 61–90. Birkhäuser, 1997.
5. M. Crouzeix and P.A. Raviart. Conforming and non–conforming finite element methods for solving the stationary stokes equations. *RAIRO Anal. Numér*, 1973.
6. R. S. Falk and J. E. Osborn. Error estimates for mixed methods. *R.A.I.R.O. Numerical Analysis*, 1980.
7. C. A. J. Fletcher. *Computational Techniques for Fluid Dynamics, Vol I and II*. Springer Series in Computational Physics. Springer-Verlag, 1988.
8. L. P. Franca, T. J. R. Hughes, and R. Stenberg. Stabilized finite element methods. In M. D. Gunzburger and R. A. Nicolaides, editors, *Incompressible Computational Fluid Dynamics; Trends and Advances*. Cambridge University Press, 1993.
9. V. Girault and P. A. Raviart. *Finite Element Methods for Navier-Stokes Equations*. Springer-Verlag, 1986.
10. C. Johnson. *Numerical solution of partial differential equations by the finite element method*. Studentlitteratur, 1987.
11. H. P. Langtangen. Tips and frequently asked questions about Diffpack. World Wide Web document: Diffpack v1.4 Report Series, SINTEF & University of Oslo, 1996.
    URL: *http://www.nobjects.com/diffpack/reports*.
12. H. P. Langtangen. *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*. Textbooks in Computational Science and Engineering. Springer, 2nd edition, 2003.
13. H. P. Langtangen. Details of finite element programming in Diffpack. The Numerical Objects Report Series #1997:9, Numerical Objects AS, Oslo, Norway, October 6, 1997. See *ftp://ftp.nobjects.com/pub/doc/NO97-09.ps.gz*.
14. K.-A. Mardal, H. P. Langtangen, and G.W. Zumbusch. Multigrid methods in diffpack. In *Computational Partial Differential Equations using Diffpack*. Springer, 2003.
15. K.-A. Mardal, J. Sundnes, and H.P Langtangen. *Systems of PDEs and block preconditionering*. Springer, 2003.
16. K.-A. Mardal, X.-C. Tai, and R. Winther. Robust finite elements for Darcy–Stokes flow.
17. R. Rannacher. Finite element methods for the incompressible Navier-Stokes equation. 1999.
    http://www.iwr.uni-heidelberg.de/sfb359/Preprints1999.html.
18. P. A. Raviart. Mixed finite element methods. In D. F. Griffiths, editor, *The Mathematical Basis of Finite Element Methods*. Clarendon Press, Oxford, 1984.

19. P. A. Raviart and J. M. Thomas. A mixed finite element method for 2-order elliptic problems. *Matematical Aspects of Finite Element Methods*, 1977.

III

A Robust Finite Element Method for Darcy-Stokes Flow

K.-A. Mardal, X.-C. Tai and R. Winther

In *SIAM Journal on Numerical Analysis*, vol 40, 1605-1631, 2002.

A ROBUST FINITE ELEMENT METHOD FOR DARCY–STOKES FLOW

KENT ANDRE MARDAL, XUE–CHENG TAI, AND RAGNAR WINTHER

ABSTRACT. Finite element methods for a family of systems of singular perturbation problems of a saddle point structure are discussed. The system is approximately a linear Stokes problem when the perturbation parameter is large, while it degenerates to a mixed formulation of Poisson's equation as the perturbation parameter tends to zero. It is established, basically by numerical experiments, that most of the proposed finite element methods for Stokes problem or the mixed Poisson's system are not well behaved uniformly in the perturbation parameter. This is used as the motivation for introducing a new "robust" finite element which exhibits this property.

1. INTRODUCTION

Let $\Omega \subset \mathbb{R}^2$ be a bounded and connected polygonal domain with boundary $\partial\Omega$. In this paper we shall consider finite element methods for the following singular perturbation problem:

$$\begin{aligned} (\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta})\boldsymbol{u} - \mathbf{grad}\, p &= \boldsymbol{f} \quad \text{in } \Omega, \\ \operatorname{div} \boldsymbol{u} &= g \quad \text{in } \Omega, \\ \boldsymbol{u} &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{1.1}$$

Here $\varepsilon \in (0, 1]$ is a parameter, while $\boldsymbol{\Delta} = \mathbf{diag}(\Delta, \Delta)$ is the Laplace operator on vector fields. The vector field $\boldsymbol{f}$ and scalar field g represent the data. The problem (1.1) only admits a solution if the function g has mean value zero on Ω and "the pressure" p is only determined up to addition of a constant.

We note that when ε is not too small, and $g = 0$, this problem is simply a standard Stokes problem, but with an additional non–harmful lower order term. However, if $\boldsymbol{f} = 0$ and ε approaches zero then the model problem formally tends to a mixed formulation of the Poisson equation with homogeneous Neumann boundary conditions.

When $\varepsilon = 0$ the first equation in (1.1) has the form of Darcy's law for flow in a homogeneous porous medium, where $\boldsymbol{u}$ is a volume averaged velocity. In fact, the system (1.1) can be regarded as a macroscopic

1991 *Mathematics Subject Classification.* Primary 65N12, 65N15, 65N30.

Key words and phrases. singular perturbation problems, Darcy–Stokes flow, nonconforming finite elements, uniform error estimates.

This work was partially supported by the Research Council of Norway (NFR), under grants 128224/431, 133755/441, and 135420/431.

model for flow in an "almost porous media," where $\boldsymbol{u}$ and p represents volume averaged velocity and pressure, respectively. The zero order velocity term in the first equation of (1.1) then typically represents a *Stokes drag*. An attempt to derive Darcy's law from volume averaged Stokes flow is for example discussed in [16]. Generalizations of the system (1.1) have also been proposed in the modeling of macrosegregation formation in binary alloy solidification, cf. [13]. Systems of the form (1.1) may also arise from time discretizations of the Navier–Stokes equation, where the parameter ε corresponds to the square root of the time step, cf. [3]. However, the study of such time discretizations is not the motivation for the present paper.

The purpose of the present paper is to discuss a finite element method for the model problem (1.1) with convergence properties that are uniform with respect to the perturbation parameter ε. In §2 we will introduce some notations and discuss various properties of the model (1.1). Discretizations of the model problem by the finite element method is described in §3. In particular, we will state stability conditions which are uniform with respect to the parameter ε, and show, by numerical experiments, that the standard discretizations, proposed either for $\varepsilon = 1$ or $\varepsilon = 0$, do not satisfy these stability conditions. A new nonconforming finite element discretization is then proposed in §4. We show that this new discretization is uniformly stable, and, as a consequence we establish, in §5, error estimates which are uniform in ε under the assumption that proper regularity estimates hold for the solution. In §6 we then study the asymptotic smoothness of the solution of (1.1) as ε tends to zero. Based on these regularity results we show that, for fixed data $\boldsymbol{f}$ and g, a uniform $O(h^{1/2})$ error estimate in a suitable energy norm can be derived.

In the final section of this paper we study an elliptic system which formally is a generalization of (1.1). This system is given by

$$
\begin{aligned}
(\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta})\boldsymbol{u} - \delta^{-2}\,\mathbf{grad}(\operatorname{div} \boldsymbol{u} - g) &= \boldsymbol{f} \quad \text{in } \Omega, \\
\boldsymbol{u} &= 0 \quad \text{on } \partial\Omega,
\end{aligned}
\tag{1.2}
$$

where $\varepsilon, \delta \in (0, 1]$. By introducing $p = \delta^{-2}(\operatorname{div} \boldsymbol{u} - g)$ this system can be alternatively written on the mixed form

$$
\begin{aligned}
(\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta})\boldsymbol{u} - \mathbf{grad}\, p &= \boldsymbol{f} \quad \text{in } \Omega, \\
\operatorname{div} \boldsymbol{u} - \delta^2 p &= g \quad \text{in } \Omega, \\
\boldsymbol{u} &= 0 \quad \text{on } \partial\Omega.
\end{aligned}
\tag{1.3}
$$

Note that this system also has meaning when $\delta = 0$, and in this case the system reduces to (1.1).

The symmetric and positive definite system (1.2) is discretized by a straightforward finite element approach utilizing the new nonconforming velocity space constructed earlier in this paper, i.e. the mixed

system (1.3) is not introduced in the discretization. We show, by numerical experiments and theory, that under the assumption of sufficiently regular solutions, we obtain error estimates which are uniform both in ε and δ.

2. Preliminaries

We will use $H^m = H^m(\Omega)$ to denote the Sobolev space of scalar functions on Ω with m derivatives in $L^2 = L^2(\Omega)$, with norm $\|\cdot\|_m$. Furthermore, the notation $\|\cdot\|_{m,K}$ is used to indicate that the norm is defined with respect to a domain K different from Ω. The seminorm derived from the partial derivatives of order equal m is denoted $|\cdot|_m$, i.e. $|\cdot|_m^2 = \|\cdot\|_m^2 - \|\cdot\|_{m-1}^2$. The space $H_0^m = H_0^m(\Omega)$ will denote the closure in H^m of $C_0^\infty(\Omega)$. The dual space of H_0^m with respect to the L^2 inner product will be denoted by H^{-m}. Furthermore, L_0^2 will denote the space of L^2 functions with mean value zero. A space written in boldface denotes a 2–vector valued analog of the corresponding scalar space. The notation $(\cdot,\cdot)$ is used to denote the L^2 inner product on scalar, vector, and matrix valued functions.

Below we shall encounter the intersection and sum of Hilbert spaces. We therefore recall the basic definitions of these concepts. If X and Y are Hilbert spaces, both continuously contained in some larger Hilbert spaces, then the intersection $X \cap Y$ and the sum $X + Y$ are themselves Hilbert spaces with the norms

$$\|z\|_{X\cap Y} = (\|z\|_X^2 + \|z\|_Y^2)^{1/2}$$

and

$$\|z\|_{X+Y} = \inf_{\substack{z=x+y \\ x\in X,\, y\in Y}} (\|x\|_X^2 + \|y\|_Y^2)^{1/2}.$$

Furthermore, if $X \cap Y$ is dense in both X and Y then $(X \cap Y)^* = X^* + Y^*$. We refer to [4, Chapter 2] for these results.

If q is a scalar field then $\mathbf{grad}\, q$ will denote the gradient of q, while $\operatorname{div} \boldsymbol{v}$ denotes the divergence of a vector field $\boldsymbol{v}$. We shall also use the diffential operators

$$\mathbf{curl}\, q = \begin{pmatrix} -\partial q/\partial x_2 \\ \partial q/\partial x_1 \end{pmatrix} \quad \text{and} \quad \operatorname{rot} \boldsymbol{v} = \partial \boldsymbol{v}_1/\partial x_2 - \partial \boldsymbol{v}_2/\partial x_1.$$

Note that, due to Green's theorem, these definitions lead to the following "integration by parts formula"

$$\int_\Omega \mathbf{curl}\, q \cdot \boldsymbol{v}\, dx = \int_\Omega q \operatorname{rot} \boldsymbol{v}\, dx + \int_{\partial\Omega} q(\boldsymbol{v}\cdot\boldsymbol{t})\, d\tau, \tag{2.1}$$

where $\boldsymbol{t}$ is the unit tangent vector in the counter clockwise direction on $\partial\Omega$, and τ is the arclength.

The gradient of a vector field $\boldsymbol{v}$ is denoted $\boldsymbol{Dv}$, i.e. $\boldsymbol{Dv}$ is the 2×2 matrix with elements

$$(\boldsymbol{Dv})_{i,j} = \partial \boldsymbol{v}_i / \partial x_j \quad 1 \leq i, j \leq 2.$$

Hence, for any $\boldsymbol{u} \in \boldsymbol{H}^2$ and $\boldsymbol{v} \in \boldsymbol{H}_0^1$ we have

$$-(\boldsymbol{\Delta u}, \boldsymbol{v}) = (\boldsymbol{Du}, \boldsymbol{Dv}) \equiv \int_\Omega \boldsymbol{Du} : \boldsymbol{Dv}\, dx,$$

where the colon denotes the scalar product of matrix fields. Recall also the identity

$$\boldsymbol{\Delta} = \mathbf{grad}\,\mathrm{div} - \mathbf{curl}\,\mathrm{rot}, \tag{2.2}$$

which can be verified by a direct computation. As a consequence, we obtain the identity

$$(\boldsymbol{Du}, \boldsymbol{Dv}) = (\mathrm{div}\,\boldsymbol{u}, \mathrm{div}\,\boldsymbol{v}) + (\mathrm{rot}\,\boldsymbol{u}, \mathrm{rot}\,\boldsymbol{v}) \quad \forall \boldsymbol{u} \in \boldsymbol{H}^1,\ \boldsymbol{v} \in \boldsymbol{H}_0^1. \tag{2.3}$$

In addition to the function spaces introduced above we will also use the space $\boldsymbol{H}(\mathrm{div}) = \boldsymbol{H}(\mathrm{div}; \Omega)$ consisting of all vector fields in $\boldsymbol{L}^2$ with divergence in L^2, i.e.

$$\boldsymbol{H}(\mathrm{div}) = \{\boldsymbol{v} \in \boldsymbol{L}^2 : \mathrm{div}\,\boldsymbol{v} \in L^2\}.$$

Similarly,

$$\boldsymbol{H}(\mathrm{rot}) = \{\boldsymbol{v} \in \boldsymbol{L}^2 : \mathrm{rot}\,\boldsymbol{v} \in L^2\},$$

and the norms of these spaces are denoted by $\|\cdot\|_{\mathrm{div}}$ and $\|\cdot\|_{\mathrm{rot}}$, respectively. Furthermore, $\boldsymbol{H}_0(\mathrm{div})$ is the closed subspace of $\boldsymbol{H}(\mathrm{div})$ consisting of functions with vanishing normal component on the boundary, i.e.

$$\boldsymbol{H}_0(\mathrm{div}) = \{\boldsymbol{v} \in \boldsymbol{H}(\mathrm{div}) : \boldsymbol{v} \cdot \boldsymbol{n} = 0 \quad \text{on } \partial\Omega\},$$

where $\boldsymbol{n}$ is the unit outward normal vector.

Throughout this paper $a_\varepsilon(\cdot, \cdot) : \boldsymbol{H}^1 \times \boldsymbol{H}^1 \mapsto \mathbb{R}$ will denote the bilinear form

$$a_\varepsilon(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}, \boldsymbol{v}) + \varepsilon^2 (\boldsymbol{Du}, \boldsymbol{Dv}).$$

A weak formulation of problem (1.1) is given by:

Find $(\boldsymbol{u}, p) \in \boldsymbol{H}_0^1 \times L_0^2$ such that

$$\begin{aligned} a_\varepsilon(\boldsymbol{u}, \boldsymbol{v}) + (p, \mathrm{div}\,\boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{H}_0^1, \\ (\mathrm{div}\,\boldsymbol{u}, q) &= (g, q) \qquad \forall q \in L_0^2. \end{aligned} \tag{2.4}$$

Here we assume that data $(\boldsymbol{f}, g)$ is given in $\boldsymbol{H}^{-1} \times L_0^2$.

The problem (2.4) has a unique solution $(\boldsymbol{u}, p) \in \boldsymbol{H}_0^1 \times L_0^2$. This follows from standard results for Stokes problem, cf. for example [11]. However, the bound on $(\boldsymbol{u}, p) \in \boldsymbol{H}_0^1 \times L_0^2$ will degenerate as ε tends to zero. In fact, for the reduced problem (2.4) with $\varepsilon = 0$ the space $\boldsymbol{H}_0^1 \times L_0^2$ is not a proper function space for the solution. However, the theory developed in [6] can be applied in this case if we seek $(\boldsymbol{u}, p)$ either in $\boldsymbol{H}_0(\mathrm{div}) \times L_0^2$ or in $\boldsymbol{L}^2 \times (H^1 \cap L_0^2)$, and with data $(\boldsymbol{f}, g)$ in the

proper dual spaces. These results are in fact consequences of standard results for the Poisson equation.

The fact that the regularity of the solution is changed when ε becomes zero strongly suggests that ε–dependent norms and function spaces are required in order to obtain stability estimates independent of ε. Furthermore, since the reduced problem is well posed for two completely different choices of function spaces, this indicates that there are at least two different choices of ε–dependent norms. In present paper we will study the problem (1.1) with respect to an ε–dependent norm which reduces to the norm in $\boldsymbol{H}_0(\mathrm{div}) \times L^2_0$ when $\varepsilon = 0$. Our goal is to derive discretizations which are uniformly stable with respect to ε in this norm. This appears to be the proper choice if we want to study discretizations which also can be generalized to non-mixed approximations of elliptic problems of the form (1.2).

Remark. When we refer to the *reduced system* corresponding to (1.1) we refer to the system (1.1) with $\varepsilon = 0$ and the boundary condition $\boldsymbol{u} = 0$ replaced by $\boldsymbol{u} \cdot \boldsymbol{n} = 0$. This system has a weak formulation given by (2.4), but with the solution space $\boldsymbol{H}^1_0$ replaced by $\boldsymbol{H}_0(\mathrm{div})$. □

The space $\boldsymbol{H}_0(\mathrm{div}) \cap \varepsilon \cdot \boldsymbol{H}^1_0$, with norm $\|\!| \cdot |\!\|_\varepsilon$ given by

$$|\!|\!| \boldsymbol{v} |\!|\!|^2_\varepsilon = \|\boldsymbol{v}\|^2_0 + \|\operatorname{div} \boldsymbol{v}\|^2_0 + \varepsilon^2 \|\boldsymbol{D}\boldsymbol{v}\|^2_0,$$

is equal to $\boldsymbol{H}^1_0$ as a set for $\varepsilon > 0$, but equal to $\boldsymbol{H}_0(\mathrm{div})$ for $\varepsilon = 0$. The system (2.4) can alternatively be written as the system

$$\mathcal{A}_\varepsilon \begin{pmatrix} \boldsymbol{u} \\ p \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} \\ g \end{pmatrix},$$

where the coefficient operator $\mathcal{A}_\varepsilon$ is given by

$$\mathcal{A}_\varepsilon = \begin{pmatrix} \boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta} & -\mathbf{grad} \\ \mathrm{div} & 0 \end{pmatrix}. \tag{2.5}$$

Let $\boldsymbol{X}_\varepsilon$ be the product space $(\boldsymbol{H}_0(\mathrm{div}) \cap \varepsilon \cdot \boldsymbol{H}^1_0) \times L^2_0$ and $\boldsymbol{X}^*_\varepsilon$ the corresponding dual space with respect to the L^2–inner product. This space can also be expressed as

$$\boldsymbol{X}^*_\varepsilon = (\boldsymbol{H}^{-1}(\mathrm{rot}) + \varepsilon^{-1} \boldsymbol{H}^{-1}) \times L^2_0.$$

Here the + sign has the interpretation as the sum of Hilbert spaces, and the space $\boldsymbol{H}^{-1}(\mathrm{rot})$ is given by

$$\boldsymbol{H}^{-1}(\mathrm{rot}) = \{\boldsymbol{v} \in \boldsymbol{H}^{-1} : \operatorname{rot} \boldsymbol{v} \in H^{-1}\}.$$

The operator $\mathcal{A}_\varepsilon$ can be seen to be an isomorphism mapping $\boldsymbol{X}_\varepsilon$ into $\boldsymbol{X}^*_\varepsilon$. Furthermore, the corresponding operator norms

$$||\mathcal{A}_\varepsilon||_{\mathcal{L}(X_\varepsilon, X^*_\varepsilon)} \quad \text{and} \quad ||\mathcal{A}^{-1}_\varepsilon||_{\mathcal{L}(X^*_\varepsilon, X_\varepsilon)}$$

are independent of ε. In fact, with the definitions above, this is also true for $\varepsilon \in [0, 1]$, i.e. the endpoint $\varepsilon = 0$ can be included.

The uniform boundedness of $\mathcal{A}_\varepsilon$ is straightforward to check from the definitions above, while the uniform boundedness of the inverse can be

verified from the two Brezzi conditions, cf. [6]. For the present problem these conditions read:

There are constants $\alpha_0, \beta_0 > 0$, independent of ε, such that

$$\sup_{\boldsymbol{v} \in \boldsymbol{H}_0(\mathrm{div}) \cap \varepsilon \cdot \boldsymbol{H}_0^1} \frac{(q, \mathrm{div}\, \boldsymbol{v})}{\|\!|\boldsymbol{v}|\!\|_\varepsilon} \geq \alpha_0 \|q\|_0 \quad \forall q \in L_0^2, \tag{2.6}$$

and

$$a_\varepsilon(\boldsymbol{v}, \boldsymbol{v}) \geq \beta_0 \|\!|\boldsymbol{v}|\!\|_\varepsilon^2 \quad \forall \boldsymbol{v} \in \boldsymbol{Z}, \tag{2.7}$$

where $\boldsymbol{Z} = \{\boldsymbol{v} \in \boldsymbol{H}_0^1 : \mathrm{div}\, \boldsymbol{v} = 0\}$.

Since it is well known, cf. for example [11, Chapter 1, Corollary 2.4], that condition (2.6) holds for $\varepsilon = 1$ it also holds for all $\varepsilon \in [0, 1]$ with the same constant α_0. Furthermore, condition (2.7) holds trivially with $\beta_0 = 1$ for $\varepsilon \in [0, 1]$.

3. Uniformly stable discretizations

The purpose of this section is to discuss finite element discretizations of the system (1.1). In particular, we shall be interested in discretizations which are stable uniformly in the parameter $\varepsilon \in (0, 1]$.

Let $\boldsymbol{V}_h \subset \boldsymbol{H}_0^1$ and $Q_h \subset L_0^2$ be finite element spaces, where $h \in (0, 1]$ is a discretization parameter. The weak formulation (2.4) leads to the following corresponding finite element discretization:

Find $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_h \times Q_h$ such that

$$\begin{aligned} a_\varepsilon(\boldsymbol{u}_h, \boldsymbol{v}) + (p_h, \mathrm{div}\, \boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_h \\ (\mathrm{div}\, \boldsymbol{u}_h, q) &= (g, q) \qquad \forall q \in Q_h. \end{aligned} \tag{3.1}$$

Remark. Below we shall also encounter several examples of nonconforming approximations of (2.4), i.e. the space $\boldsymbol{V}_h \not\subseteq \boldsymbol{H}_0^1$. In all these examples the bilinear form $a_\varepsilon(\cdot, \cdot)$ is understood to be the sum of the corresponding integrals over each element. No extra jump terms are added. The same remark applies to the energy norm, $\|\!|\cdot|\!\|_\varepsilon$. □

The discretization (3.1) is stable in the sense of [6] if proper discrete analogs of the conditions (2.6) and (2.7) holds. These conditions are:

Stability conditions.

The discretization (3.1) is said to be uniformly stable if there exist constants $\alpha, \beta > 0$, independent of ε and h, such that

$$\sup_{\boldsymbol{v} \in \boldsymbol{V}_h} \frac{(q, \mathrm{div}\, \boldsymbol{v})}{\|\!|\boldsymbol{v}|\!\|_\varepsilon} \geq \alpha \|q\|_0 \quad \forall q \in Q_h, \tag{3.2}$$

and

$$a_\varepsilon(\boldsymbol{v}, \boldsymbol{v}) \geq \beta \|\!|\boldsymbol{v}|\!\|_\varepsilon^2 \quad \forall \boldsymbol{v} \in \boldsymbol{Z}_h, \tag{3.3}$$

where $\boldsymbol{Z}_h = \{\boldsymbol{v} \in \boldsymbol{V}_h : (\mathrm{div}\, \boldsymbol{v}, q) = 0 \quad \forall q \in Q_h\}$.

For the case $\varepsilon = 1$, or more precisely for ε bounded away from zero, the second condition is obvious. In this case there are several choices of pairs of finite element spaces which satisfies (3.2) with α independent

of h. We mention for example the Mini element proposed in [1] or the $P_2 - P_0$ element, i.e. we choose continuous quadratic velocities for $\boldsymbol{V}_h$ and the corresponding space of piecewise constants for Q_h, cf. [10]. For a general review of stable Stokes elements we refer to [8].

However, most of these spaces do *not* lead to discretizations which are stable uniformly in ε. The main reason for this is that when ε approaches zero the second condition is no longer obvious. In fact, for the reduced problem with $\varepsilon = 0$ the condition (3.3) requires

$$\|\boldsymbol{v}\|_0^2 \geq \beta \|\boldsymbol{v}\|_{\text{div}}^2 \quad \forall \boldsymbol{v} \in \boldsymbol{Z}_h.$$

Hence, we must have

$$\| \operatorname{div} \boldsymbol{v}\|_0 \leq c\|\boldsymbol{v}\|_0 \quad \forall \boldsymbol{v} \in \boldsymbol{Z}_h \tag{3.4}$$

for a suitable constant c independent of h, and this condition does not hold for the common conforming stable Stokes elements.

Example 3.1 We consider the problem (1.1) with Ω taken as the unit square. The domain is triangulated by first dividing it into $h \times h$ squares. Then, each square is divided into two triangles by the diagonal with a negative slope. The system is then discretized using the $P_2 - P_0$ element with respect to this triangulation, i.e. $\boldsymbol{V}_h \subset \boldsymbol{H}_0^1$ consists of piecewise quadratic functions, while $Q_h \subset L_0^2$ is the space of discontinuous piecewise constants. This discretization is known to be stable when $\varepsilon > 0$ is fixed, cf. [10]. However, our purpose here is to investigate how the convergence behave as ε becomes small.

We consider the system (1.1) with the function g chosen to be identical zero, while $\boldsymbol{f} = \boldsymbol{u} - \varepsilon^2 \boldsymbol{\Delta u} - \mathbf{grad}\, p$, where $\boldsymbol{u} = \mathbf{curl} \sin^2(\pi x_1) \sin^2(\pi x_2)$ and $p = \sin(\pi x_1)$. Hence, in this example the solution is independent of ε.

In Table 3.1 below we have computed the relative L^2 error in the velocity $\boldsymbol{u}$, i.e. $e(h) = \|\boldsymbol{u} - \boldsymbol{u}_h\|_0 / \|\boldsymbol{u}\|_0$, for different values of ε and h. A third order Gauss-Legendre rule, cf. [17], was used here, and in all the other examples of this section, to perform the necessary integrations. For each fixed ε the convergence rate with respect to h, γ, is estimated by assuming $e(h) = ch^\gamma$, and by computing a least squares fit to this log–linear relation.

$\varepsilon \backslash$ h	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
1	3.84e-2	4.75e-3	6.41e-4	1.04e-4	2.11e-5	2.72
2^{-2}	6.15e-2	1.73e-2	4.65e-3	1.20e-3	3.05e-4	1.92
2^{-4}	4.55e-1	2.10e-1	6.78e-2	1.86e-2	4.79e-3	1.67
2^{-8}	9.31e-1	9.68e-1	9.43e-1	8.14e-1	5.32e-1	0.19
0	9.35e-1	9.84e-1	1.00	1.01	1.02	-0.03

TABLE 3.1. The relative L^2 error in velocity obtained by the $P_2 - P_0$ element.

When $\varepsilon = 1$ the convergence seems to be at least quadratic with respect to h in this case. However, the convergence deteriorates as ε becomes smaller, and for $\varepsilon = 0$ there is no convergence.

Table 3.2 is based on the corresponding relative errors in the energy norm, i.e. the norm $|||\cdot|||_\varepsilon$ for velocity and the L^2 norm for pressure. For simplicity only the estimated convergence rates are given.

ε	1	2^{-2}	2^{-4}	2^{-8}	0
rate, velocity	1.84	1.01	0.70	-0.79	-1.03
rate, pressure	1.06	1.01	1.09	0.13	-0.20

TABLE 3.2. Estimated convergence rates for the velocity and pressure, measured in the energy norm, for the $P_2 - P_0$ element.

These results indicate a similar degenerate behavior with respect to ε. In fact, when $\varepsilon = 0$ the norm, $|||\boldsymbol{u}_h|||_\varepsilon$, seems to grow like h^{-1} as h approach zero. This must be due to the fact that only the projection of $\operatorname{div} \boldsymbol{u}_h$ into piecewise constants is controlled by the method in this case. □

Example 3.2 We repeat the experiment above, but with the difference that we use the nonconforming Crouzeix–Raviart element instead of the $P_2 - P_0$ element, i.e. $\boldsymbol{V}_h$ consists of piecewise linear vector fields which are continuous at the midpoint of each edge of the triangulation, while $Q_h \subset L^2_0$ is the space of piecewise constants. It is well known that for any fixed $\varepsilon > 0$ this element leads to a stable discretization, cf. [10].

In Table 3.3 we have again computed the relative L^2 error in the velocity $\boldsymbol{u}$ for different values of ε and h.

$\varepsilon \backslash$ h	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
1	1.83e-1	4.89e-2	1.26e-2	3.19e-3	8.02e-4	1.96
2^{-2}	2.19e-1	6.89e-2	1.91e-2	4.96e-3	1.26e-3	1.87
2^{-4}	6.42e-1	3.86e-1	1.53e-1	4.58-2	1.21-3	1.45
2^{-8}	9.51e-1	1.00	1.01	9.43e-1	7.44e-1	0.08
0	9.53e-1	1.01	1.04	1.05	1.06	-0.04

TABLE 3.3. The relative L^2 error in velocity obtained by the nonconforming Crouzeix–Raviart element.

The L^2 convergence appears to be quadratic when ε is large. However, also in this case the convergence deteriorates as ε decreases, and for the reduced problem, with $\varepsilon = 0$, the observed values for the relative error is monotonically increasing.

The corresponding estimates of the the convergence rates in energy norm decreases from approximately linear convergence to no convergence as is shown by Table 3.4.

ε	1	2^{-2}	2^{-4}	2^{-8}	0
rate, velocity	0.98	0.97	0.74	0.03	-0.03
rate, pressure	1.00	0.93	0.98	0.12	-0.03

TABLE 3.4. Estimated convergence rates for the velocity and pressure, measured in the energy norm, for the Crouzeix–Raviart element.

In fact, the divergence of the Crouzeix–Raviart element in the case $\varepsilon = 0$ is not surprising. Since the divergence free vector fields in this case can be realized as the curl operator applied to the corresponding Morley space, this behavior of the Crouzeix–Raviart element is closely tied to the divergence of the Morley element for the Poisson equation, cf. [14]. □

The two examples above show that the $P_2 - P_0$ element and the nonconforming Crouzeix–Raviart element, which both are known to be stable for $\varepsilon = 1$, fail to give methods which converge uniformly in ε. The divergence of the $P_2 - P_0$ element for $\varepsilon = 0$ is basically due to the fact that the estimate (3.4) does not hold, and therefore the method is unstable, while the divergence of the Crouzeix–Raviart method is caused by the inconsistency of the method.

Example 3.3 We repeat the experiment above once more, but this time the system (1.1) is discretized by using the Mini element, i.e. $\boldsymbol{V}_h \subset \boldsymbol{H}_0^1$ consists of linear combinations of piecewise linear functions and cubic bubble functions with support on a single triangle, while $Q_h \subset L_0^2$ is the space of continuous piecewise linear functions.

In Table 3.5 below we have computed the relative error in the velocity, with respect to the energy norm $|||\cdot|||_\varepsilon$, for different values of ε and h.

$\varepsilon \backslash$ h	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
1	3.01	1.65	8.42e-1	4.22e-1	2.11e-1	0.96
2^{-2}	2.70	1.55	7.80e-1	3.90e-1	1.95e-1	0.96
2^{-4}	3.71	1.67	7.89e-1	3.87e-1	1.92e-1	1.07
2^{-8}	7.32	4.28	2.79	1.64	6.51e-1	0.84
0	7.44	4.76	3.70	3.39	3.30	0.28

TABLE 3.5. The relative error in velocity, measured in the energy norm, for the Mini element.

When $\varepsilon = 1$ the convergence seems to be linear with respect to h. This agrees with the theoretical results given in [1]. The convergence

deteriorates as ε becomes smaller, and for $\varepsilon = 0$ there seems to be essentially no convergence in the energy norm.

An interesting observation can be made for the Mini element if we consider the corresponding errors for the pressure p. In Table 3.6 below we study the relative error given by $\|p - p_h\|_0 / \|p\|_0$.

$\varepsilon \backslash h$	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
1	8.78	2.81	8.85e-1	2.95e-1	1.02e-1	1.61
2^{-2}	6.09e-1	1.84e-1	5.62e-2	1.85e-2	6.40e-3	1.64
2^{-4}	6.08e-2	1.51e-2	3.88e-3	1.21e-3	4.07e-4	1.81
2^{-8}	3.58e-2	9.93e-3	2.34e-3	4.10e-4	6.00e-5	2.30
0	3.59e-2	1.02e-2	2.75e-3	7.23e-4	1.87e-4	1.90

TABLE 3.6. The relative L^2 error in the pressure obtained by the Mini element.

The surprising observation is that for the pressure the convergence seems to be uniform with respect to ε. In fact, the convergence rate seems to improve as ε tends to zero and for ε small the convergence with respect to h appears to be quadratic. This is a striking difference to what we observed in Examples 3.1 and 3.2. In both these cases the error in the pressure diverges as ε tend to zero, cf. Tables 3.2 and 3.4.

What we have observed here is not special to the present example. The Mini element leads to a discretization which is uniformly stable with respect to ε in a proper ε–dependent norm different from $|||\cdot|||_\varepsilon$. If we define the solution space $\boldsymbol{X}_\varepsilon$ by

$$\boldsymbol{X}_\varepsilon = (\boldsymbol{L}^2 \cap \varepsilon \cdot \boldsymbol{H}_0^1) \times ((\boldsymbol{H}^1 \cap L_0^2) + \varepsilon^{-1} \cdot L^2), \tag{3.5}$$

then it can be shown that the Mini element will in fact produce a uniformly stable discretization in the corresponding energy norm. This norm degenerates to the norm of $\boldsymbol{L}^2 \times H^1$ as ε tends to zero, cf. the discussion in Section 2 above. In order to confirm this behavior we computed the relative error in velocity once more, but this time we used the L^2 norm instead of $|||\cdot|||_\varepsilon$. The results are given in Table 3.7.

$\epsilon \backslash h$	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
1	3.54e-1	1.03e-1	2.64e-2	6.60e-3	1.65e-3	1.95
2^{-2}	3.16e-1	8.79e-2	2.20e-2	5.48e-3	1.37e-3	1.97
2^{-4}	1.90e-1	4.60e-2	1.07e-2	2.59e-3	6.42e-4	2.06
2^{-8}	1.81e-1	7.23e-2	2.87e-2	8.70e-3	1.74e-3	1.64
0	1.82e-1	7.66e-2	3.59e-2	1.76e-2	8.75e-3	1.09

TABLE 3.7. The relative L^2 error in velocity obtained by the Mini element.

We observe that as ε decreases from one to zero the corresponding convergence rate decreases from approximately two to one. However, there is no sign which indicates that the behavior will deteriorate below linear convergence. To complete the picture we have also computed the estimated convergence rares for the pressure in H^1. The results are given in Table 3.8.

ε	1	2^{-2}	2^{-4}	2^{-8}	0
rate	0.61	0.64	0.86	0.99	0.99

TABLE 3.8. Estimated convergence rates for the H^1 error of the pressure obtained by the Mini element.

The estimated convergence rate is clearly below one when $\varepsilon = 1$, while it improves towards one as ε is decreased. This is consistent with the fact that the norm of the pressure component of the product space (3.5) is weaker than the H^1 norm for each $\varepsilon > 0$, but approaches the H^1 norm as ε apprach zero.

The results above seem to confirm that the Mini element leads to a uniformly convergent discretization as long as the error is properly measured. However, as motivated in Section 2 above, in the present paper we are interested in a discretization of the system (1.1) which has a uniform behavior when the error is measured in $(\boldsymbol{H}_0(\mathrm{div}) \cap \varepsilon \cdot \boldsymbol{H}_0^1) \times L_0^2$. Therefore, for our purpose here, the Mini element should not be regarded as a uniformly stable element. □

Let us recall that if a standard conforming Stokes element is not uniformly stable with respect to ε, then this instability must be caused by the failure of the second stability condition (3.3), or equivalently (3.4). Note that the stability condition (3.4) will be trivially satisfied if the spaces $\boldsymbol{V}_h \times Q_h$ are constructed such that all elements of $\boldsymbol{Z}_h$ are divergence free, i.e. $\boldsymbol{Z}_h \subset \boldsymbol{Z}$. In fact, nearly all proposed finite element methods for the reduced problem will have this property. This is for example true for the Raviart–Thomas spaces, cf. [15], and for the Brezzi–Douglas–Marini spaces of [7]. However, in all these cases the spaces $\boldsymbol{V}_h$ will only be a subspace of $\boldsymbol{H}_0(\mathrm{div})$ and not of $\boldsymbol{H}_0^1$, due to the fact that only the normal components of the elements of $\boldsymbol{V}_h$ are required to be continuous across element edges. It is therefore not clear that these spaces will be useful for problems of the form (1.1) with $\varepsilon > 0$.

Example 3.4 We repeat the calculation done in the three examples above, but now we use the lowest order Raviart–Thomas space for the discretization. Hence, for $\varepsilon = 0$ we will expect to obtain linear convergence with respect to h. On the other hand, for $\varepsilon > 0$ the method is nonconforming and there seems to be no reason to expect that the method is convergent in this case. In Table 3.9 we have computed the

estimated convergence rates with respect to h for the relative L^2 errors of the velocity $\boldsymbol{u}$ and the pressure p for different values of ε.

ε	1	2^{-2}	2^{-4}	2^{-8}	0
rate, velocity	-0.07	-0.07	0.28	0.97	0.97
rate, pressure	-0.04	0.08	0.86	1.01	1.01

TABLE 3.9. Estimated convergence rates for the L^2 errors of the velocity and pressure for the Raviart–Thomas element.

As expected, the method appears to be divergent for $\varepsilon > 0$. □

4. A robust nonconforming finite element space

The four examples presented above illustrate that none of the standard elements, proposed for the case $\varepsilon = 1$ or $\varepsilon = 0$, will lead to a discrization of the problem (1.1) with uniform convergence properties with respect to ε, when the error is measured in the norm of the space $(\boldsymbol{H}_0(\mathrm{div}) \cap \varepsilon \cdot \boldsymbol{H}_0^1) \times L_0^2$. The purpose of the rest of this paper is therefore to construct and analyze a new finite element space which has this property.

4.1. **The finite element space.** In order to describe the new finite element space we will first define the proper polynomial space, or shape functions, on a given triangle. Let $T \subset \mathbb{R}^2$ be a triangle and consider the polynomial space of vector fields on T given by

$$\boldsymbol{V}(T) = \{\boldsymbol{v} \in \mathbb{P}_3^2 : \mathrm{div}\, \boldsymbol{v} \in \mathbb{P}_0, \quad (\boldsymbol{v} \cdot \boldsymbol{n})|_e \in \mathbb{P}_1 \quad \forall e \in \mathcal{E}(T)\}.$$

Here $\mathbb{P}_k$ denotes the set of polynomials of degree k and $\mathcal{E}(T)$ denotes the set of the edges of T. Furthermore, $\boldsymbol{n}$ is the unit normal vector on the edge e. Below we will also use $\boldsymbol{t}$ to denote the unit tangent vector on e, while τ denotes the arc length along e.

The space $\mathbb{P}_3^2$ is a vector space of dimension twenty. Furthermore, the conditions

$$\mathrm{div}\, \boldsymbol{v} \in \mathbb{P}_0 \quad \text{and} \quad (\boldsymbol{v} \cdot \boldsymbol{n})|_e \in \mathbb{P}_1 \quad \forall e \in \mathcal{E}(T),$$

represent at most eleven linearly independent constraints on this space. Therefore we must have

$$\dim \boldsymbol{V}(T) \geq 9.$$

In fact, we shall show that $\dim \boldsymbol{V}(T) = 9$.

Lemma 4.1. *The space $\boldsymbol{V}(T)$ is a linear space of dimension nine. Furthermore, an element $\boldsymbol{v} \in \boldsymbol{V}(T)$ is uniquely determined by the following degrees of freedom:*

- $\int_e (\boldsymbol{v} \cdot \boldsymbol{n}) \tau^k \, d\tau \quad k = 0, 1 \quad$ *for all* $e \in \mathcal{E}(T)$.
- $\int_e (\boldsymbol{v} \cdot \boldsymbol{t}) \, d\tau \quad$ *for all* $e \in \mathcal{E}(T)$.

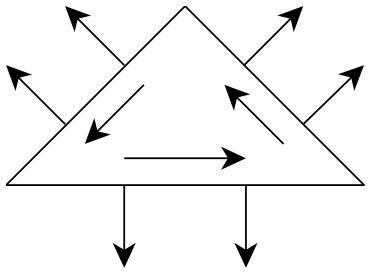

FIGURE 4.1. The degrees of freedom of the new nonconforming element.

Proof. Since $\boldsymbol{V}(T)$ is a vector space of dimension ≥ 9 it is enough to show that elements of $\boldsymbol{V}(T)$ are uniquely determined by the given nine degrees of freedom. Assume that $\boldsymbol{v} \in \boldsymbol{V}(T)$ with all the degrees of freedom equal zero. In particular, this implies that

$$(\boldsymbol{v} \cdot \boldsymbol{n})|_{\partial T} \equiv 0.$$

As a consequence of this

$$\int_T \operatorname{div} \boldsymbol{v}\, dx = \int_{\partial T} \boldsymbol{v} \cdot \boldsymbol{n}\, d\tau = 0.$$

Hence, since $\operatorname{div} \boldsymbol{v} \in \mathbb{P}_0$, we conclude that $\boldsymbol{v}$ is divergence free.

However, since $\boldsymbol{v} \in \mathbb{P}_3^2$ is divergence free we must have $\boldsymbol{v} = \mathbf{curl}\, w$ for a suitable scalar function $w \in \mathbb{P}_4$. Furthermore, since

$$(\mathbf{grad}\, w \cdot \boldsymbol{t})|_e = (\boldsymbol{v} \cdot \boldsymbol{n})|_e = 0$$

for each edge e, we conclude that $\mathbf{grad}\, w \cdot \boldsymbol{t} \equiv 0$ on ∂T. Since w is uniquely determined only up to a constant, we can therefore assume that $w \equiv 0$ on ∂T.

Hence, w is of the form $w = pb$, where $p \in \mathbb{P}_1$ and b is the cubic bubble function with respect to T, i.e. $b = \lambda_1\lambda_2\lambda_3$, where $\lambda_i(x)$ are the barycentric coordinates of x with respect to the three corners of T. In particular, $\frac{\partial b}{\partial \boldsymbol{n}}|_e$ does not change sign on e. Furthermore,

$$\frac{\partial w}{\partial \boldsymbol{n}}|_{\partial T} = p\frac{\partial b}{\partial \boldsymbol{n}}|_{\partial T},$$

and

$$\int_e p\frac{\partial b}{\partial \boldsymbol{n}}\, d\tau = \int_e \frac{\partial w}{\partial \boldsymbol{n}}\, d\tau = \int_e \boldsymbol{v} \cdot \boldsymbol{t}\, d\tau = 0 \quad \forall e \in \mathcal{E}(T).$$

We can therefore conclude that p has a root in the interior of e. However, if $p \in \mathbb{P}_1$ with a root in the interior of each edge of T then $p \equiv w \equiv 0$. □

Let $\{\mathcal{T}_h\}$ be a shape regular family of triangulations of Ω, where h is the maximal diameter. Furthermore, let $\mathcal{E}_h$ be the set of edges of $\mathcal{T}_h$.

Define a finite element space of vector fields $\boldsymbol{V}_h$, associated with the triangulation $\mathcal{T}_h$, as all functions $\boldsymbol{v} \in \boldsymbol{V}_h$ such that

- $\boldsymbol{v}|_T \in \boldsymbol{V}(T)$ for all $T \in \mathcal{T}_h$
- $\int_e (\boldsymbol{v} \cdot \boldsymbol{n}) \tau^k \, d\tau$ is continuous for $k = 0, 1$ for all $e \in \mathcal{E}_h$,
- $\int_e (\boldsymbol{v} \cdot \boldsymbol{t}) \, d\tau$ is continuous for all $e \in \mathcal{E}_h$,

Here we assume that $\boldsymbol{v}$ is extended to be zero outside Ω, i.e. if e is an edge on the boundary of Ω then we require

$$\int_e (\boldsymbol{v} \cdot \boldsymbol{n}) \tau^k \, d\tau = 0 \qquad k = 0, 1 \quad \text{and} \quad \int_e (\boldsymbol{v} \cdot \boldsymbol{t}) \, d\tau = 0.$$

It follows from Lemma 4.1 that any function $\boldsymbol{v} \in \boldsymbol{V}_h$ is uniquely determined by the two lowest order moments of $\boldsymbol{v} \cdot \boldsymbol{n}$ and by the mean value of $\boldsymbol{v} \cdot \boldsymbol{t}$ for all interior edges, cf. Figure 4.1.

If $\boldsymbol{v} \in \boldsymbol{V}_h$ then the normal component $\boldsymbol{v} \cdot \boldsymbol{n}$ is continuous for all interior edges. Therefore, $\boldsymbol{V}_h \subset \boldsymbol{H}_0(\mathrm{div})$. However, the tangential component of $\boldsymbol{v}$ is not continuous, only the mean value with respect to each edge is continuous. Therefore, $\boldsymbol{V}_h \not\subseteq \boldsymbol{H}_0^1$. In addition to the space $\boldsymbol{V}_h$ we let $Q_h \subset L_0^2$ denote the space of scalar piecewise constants with respect to the triangulation $\mathcal{T}_h$.

In the rest of this paper $\boldsymbol{V}_h$ and Q_h will always refer to the finite element spaces just introduced. The corresponding nonconforming finite element approximation of the system (1.1) is defined by the system (3.1).

4.2. **Properties of the new finite element space.** It follows from the definition of $\boldsymbol{V}_h$ that $\mathrm{div}\, \boldsymbol{V}_h \subset Q_h$. Hence, if we define $\boldsymbol{Z}_h \subset \boldsymbol{V}_h$ as the weakly divergence free elements of $\boldsymbol{V}_h$, i.e.

$$\boldsymbol{Z}_h = \{\boldsymbol{v} \in \boldsymbol{V}_h : (\mathrm{div}\, \boldsymbol{v}, q) = 0 \quad \forall q \in Q_h\},$$

then these elements are in fact divergence free.

Remark. It can be seen that

$$\boldsymbol{Z}_h = \mathbf{curl}\, W_h, \tag{4.1}$$

where W_h is an associated nonconforming H^2–element. Locally, on each triangle W_h consists of all $\mathbb{P}_4$ polynomials which reduces to a quadratic on each edge. In addition, $W_h \subset H_0^1$ and the normal derivatives of functions in W_h are continuous on the average on each edge. The finite element space W_h is precisely described and analyzed in [14]. The identity (4.1) was actually the main motivation for the construction of the space $\boldsymbol{V}_h$. More precisely, the spaces W_h, $\boldsymbol{V}_h$ and Q_h are related such that the sequence

$$0 \longrightarrow W_h/\mathbb{R} \xrightarrow{\mathbf{curl}} \boldsymbol{V}_h \xrightarrow{\mathrm{div}} Q_h \longrightarrow 0.$$

is exact. In particular, $\mathrm{div}\, \boldsymbol{V}_h = Q_h$. $\square$

Define an interpolation operator $\boldsymbol{\Pi}_h : \boldsymbol{H}_0^1 \mapsto \boldsymbol{V}_h$ by

$$\int_e (\boldsymbol{\Pi}_h \boldsymbol{v} \cdot \boldsymbol{n}) \tau^k \, d\tau = \int_e (\boldsymbol{v} \cdot \boldsymbol{n}) \tau^k \, d\tau \quad k = 0, 1$$
$$\int_e (\boldsymbol{\Pi}_h \boldsymbol{v} \cdot \boldsymbol{t}) \, d\tau = \int_e (\boldsymbol{v} \cdot \boldsymbol{t}) \, d\tau$$

for all $e \in \mathcal{E}_h$. In addition, let $P_h : L_0^2 \mapsto Q_h$ be the L^2–projection. From the definition of the operator $\boldsymbol{\Pi}_h$ we easily verify the commutativity property

$$\operatorname{div} \boldsymbol{\Pi}_h \boldsymbol{v} = P_h \operatorname{div} \boldsymbol{v} \quad \text{for all } \boldsymbol{v} \in \boldsymbol{H}_0^1. \tag{4.2}$$

In fact, for all $T \in \mathcal{T}_h$

$$\int_T \operatorname{div} \boldsymbol{\Pi}_h \boldsymbol{v} \, dx = \int_{\partial T} (\boldsymbol{\Pi}_h \boldsymbol{v} \cdot \boldsymbol{n}) \, d\tau = \int_{\partial T} (\boldsymbol{v} \cdot \boldsymbol{n}) \, d\tau = \int_T \operatorname{div} \boldsymbol{v} \, dx$$

and hence (4.2) follows.

Since Q_h is the space of piecewise constants the L^2–projection P_h onto Q_h satisfies

$$\|w - P_h w\|_0 \le ch\|w\|_1 \tag{4.3}$$

for all $w \in H^1 \cap L_0^2$, where $c > 0$ is independent of h and w. The operator $\boldsymbol{\Pi}_h$ is well defined on $\boldsymbol{H}_0^1$, it is locally defined on each triangle, and it preserves linear functions locally. Furthermore, the polynomial space $\boldsymbol{V}(T)$ is invariant under affine Piola transformations. More precisely, let $T \in \mathcal{T}_h$ and $\phi(x) = Bx + c$ an affine map of T onto a reference triangle $\hat{T}$. Then the Piola transform, $\boldsymbol{v} \mapsto \hat{\boldsymbol{v}}$, where

$$\hat{\boldsymbol{v}}(\hat{x}) = (\det B)^{-1} B v(x), \qquad \hat{x} = \phi(x),$$

maps $\boldsymbol{V}(T)$ onto $\boldsymbol{V}(\hat{T})$. Therefore, approximation estimates for the operator $\boldsymbol{\Pi}_h$ can be derived from standard scaling arguments utilizing the shape regularity of $\{\mathcal{T}_h\}$. In particular, there exists a constant $c > 0$, independent of h such that

$$\|\boldsymbol{\Pi}_h \boldsymbol{v}\|_{\operatorname{div}} \le \|\boldsymbol{\Pi}_h \boldsymbol{v}\|_{1,h} \le c\|\boldsymbol{v}\|_1. \tag{4.4}$$

In addition, from the Bramble–Hilbert lemma we can further conclude that

$$\|\boldsymbol{\Pi}_h \boldsymbol{v} - \boldsymbol{v}\|_{j,h} \le ch^{k-j}\|\boldsymbol{v}\|_k \qquad \text{for } 0 \le j \le 1 \le k \le 2, \tag{4.5}$$

and for all $\boldsymbol{v} \in \boldsymbol{H}_0^1 \cap \boldsymbol{H}^k$. Here $\|\cdot\|_{j,h}$ denotes the piecewise $\boldsymbol{H}^j$–norm

$$\|\boldsymbol{v}\|_{j,h}^2 = \sum_{T \in \mathcal{T}_h} \|\boldsymbol{v}\|_{j,T}^2.$$

In fact, if $\hat{T}$ is a reference triangle, and $\hat{\boldsymbol{\Pi}} : \boldsymbol{H}^1(\hat{T}) \mapsto \boldsymbol{V}(\hat{T})$ the corresponding interpolation operator, then for all $\boldsymbol{v} \in H^1(\hat{T})$

$$\|\hat{\boldsymbol{\Pi}} \boldsymbol{v}\|_{0,\hat{T}} \le c_1 \|\boldsymbol{v}\|_{0,\partial\hat{T}} \le c_2 \|\boldsymbol{v}\|_{0,\hat{T}}^{1/2} \|\boldsymbol{v}\|_{1,\hat{T}}^{1/2},$$

where c_1 and c_2 only depends on $\hat{T}$. Hence, from a scaling argument we also obtain the low order estimate

$$\|\boldsymbol{\Pi}_h \boldsymbol{v} - \boldsymbol{v}\|_0 \leq c h^{1/2} \|\boldsymbol{v}\|_0^{1/2} \|\boldsymbol{v}\|_1^{1/2} \tag{4.6}$$

for all $\boldsymbol{v} \in \boldsymbol{H}_0^1$.

Next we will verify the stability conditions (3.2) and (3.3) for the product space $\boldsymbol{V}_h \times Q_h$. However, due to the fact that we are considering a nonconforming finite element approximation of the system (1.1), where $\boldsymbol{V}_h \not\subseteq \boldsymbol{H}_0^1$, the norm $|||\cdot|||_\varepsilon$ has to be properly modified. For each $\boldsymbol{v} \in \boldsymbol{V}_h$ we define

$$|||\boldsymbol{v}|||_{\varepsilon,h}^2 = \|\boldsymbol{v}\|_{\mathrm{div}}^2 + \varepsilon^2 \sum_{T \in \mathcal{T}_h} \|\boldsymbol{D}\boldsymbol{v}\|_{0,T}^2.$$

Note that for $\varepsilon = 0$ this norm is simply equal to $\|\cdot\|_{\mathrm{div}}$, while for $\varepsilon = 1$ it is equivalent, uniformly in h, to the piecewise $\boldsymbol{H}^1$–norm $\|\cdot\|_{1,h}$.

Lemma 4.2. *There exists a constant $\alpha_1 > 0$, independent of h, such that*

$$\sup_{\boldsymbol{v} \in \boldsymbol{V}_h} \frac{(q, \operatorname{div} \boldsymbol{v})}{\|\boldsymbol{v}\|_{1,h}} \geq \alpha_1 \|q\|_0 \quad \textit{for all } q \in Q_h.$$

Proof. This follows by a standard argument from the properties of the interpolation operator $\boldsymbol{\Pi}_h$ and the corresponding continuous result (2.6). In fact, since for any $\boldsymbol{v} \in \boldsymbol{H}_0^1$ and $q \in Q_h$ we have

$$(q, \operatorname{div} \boldsymbol{\Pi}_h \boldsymbol{v}) = (q, \operatorname{div} \boldsymbol{v})$$

and

$$\|\boldsymbol{\Pi}_h \boldsymbol{v}\|_{1,h} \leq c_1 \|\boldsymbol{v}\|_1,$$

we can take $\alpha_1 = \alpha_0 / c_1$. □

The following uniform stability result is an immediate consequence of the previous lemma.

Theorem 4.1. *The pair of spaces $(\boldsymbol{V}_h, Q_h)$ satisfies the uniform stability conditions* (3.2) *and* (3.3)*, but with the norm $|||\cdot|||_\varepsilon$ replaced by $|||\cdot|||_{\varepsilon,h}$.*

Proof. The norms $|||\cdot|||_{1,h}$ and $\|\cdot\|_{1,h}$ are equivalent on $\boldsymbol{V}_h$ and $|||\cdot|||_{\varepsilon,h}$ decreases as ε decreases. It follows from Lemma 4.2 that condition (3.2) holds. Since $\boldsymbol{Z}_h \subset \boldsymbol{Z}$ the second condition (3.3) holds with $\beta = 1$. □

5. Error estimates for smooth solutions

Since our new finite element space $(\boldsymbol{V}_h, Q_h)$ satisfies the proper stability conditions (3.2) and (3.3), uniformly with respect to ε, it seems probable that the corresponding finite element method will in fact have uniform convergence properties. In the present section we shall investigate this question under the assumption that the solution (u, p) of the continuous problem is sufficiently smooth, while the effect of the

ε–dependent boundary layers will be taken into account in the next section.

We will start the discussion here with a numerical example which is completely similar to Examples 3.1–3.3.

Example 5.1 We redo the computations done in Examples 3.1–3.3, but this time we use the finite element spaces constructed above. In all the numerical examples with the the new element we used a fifth order Gauss-Legendre method, cf. [17], as integration rule.

In Table 5.1 we have computed the estimated convergence rates with respect to h for the velocity and the pressure.

ε	1	2^{-2}	2^{-4}	2^{-8}	0
rate, velocity in L^2	1.93	1.94	1.94	1.90	1.92
rate, velocity in $\|\!\|\!\| \cdot \|\!\|\!\|_\varepsilon$	0.98	0.99	1.05	1.72	1.92
rate, pressure in L^2	0.98	1.00	1.00	1.00	1.00

TABLE 5.1. Estimated convergence rates for the velocity and the pressure for the new nonconforming element.

We observe that the convergence rates in L^2 appears to be close to quadratic in velocity and linear in pressure uniformly with respect to $\varepsilon \in [0, 1]$, while the convergence in the energy norm appears to be at least linear for each $\varepsilon > 0$. In fact, as ε approaches zero the convergence rate tends to two. This improved convergence is partly due to the fact that the exact solution $\boldsymbol{u}$ is divergence free in this case.

To make a direct comparison between the $P_2 - P_0$ element, the Crouzeix–Raviart element, the Mini element, and the new element when ε is small compared to h, we have plotted the errors in velocity for the different methods as functions of σ, where $h = 2^{-\sigma}$. Here we have chosen $\varepsilon = 2^{-8}$. The errors are plotted, in a logarithmic scale, in Figure 5.1.

To the left the L^2 errors are plotted, while the errors in the energy norm are depicted to the right. We observe that the Mini element and the new element bahaves comparably with respect to the L^2 norm, while the new element clearly is superior to all the other methods with respect to the energy norm. □

The rest of this section will be devoted to establishing error estimates for the new nonconforming finite element method. Throughout this section we will assume that $\boldsymbol{u} \in \boldsymbol{H}^2 \cap \boldsymbol{H}_0^1$, where $(\boldsymbol{u}, p)$ is the weak solution of (2.4). For convenience we also introduce the notation $\|\cdot\|_a$ for the norm on $\boldsymbol{V}_h$ associated the bilinear form a_ε, i.e.

$$\|\boldsymbol{v}\|_a^2 = \|\boldsymbol{v}\|_0^2 + \sum_{T \in \mathcal{T}_h} \|\boldsymbol{D}\boldsymbol{v}\|_{0,T}^2.$$

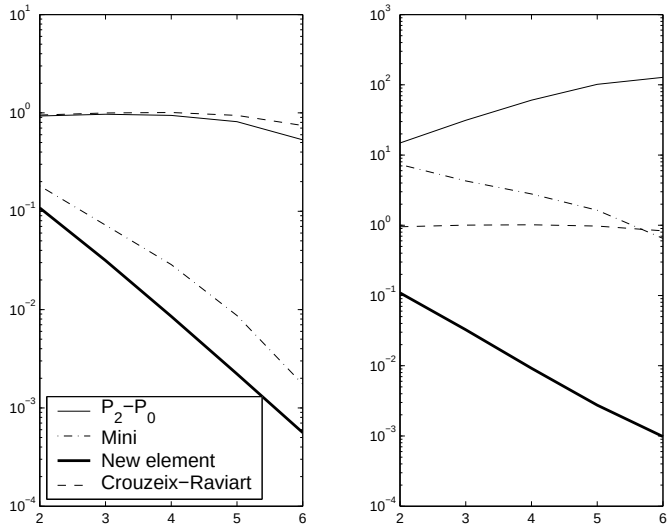

FIGURE 5.1. The errors in velocity, measured in the L^2 norm and the energy norm, as functions of $\sigma = -\log(h)/\log(2)$.

For any $\boldsymbol{v} \in \boldsymbol{V}_h$, we define the consistency error $E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{v})$ by

$$E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{v}) = \varepsilon^2 \sum_{e \in \mathcal{E}_h} \int_e (\operatorname{rot} \boldsymbol{u}) \ [\boldsymbol{v} \cdot \boldsymbol{t}] \, d\tau.$$

Here, if T_- and T_+ are two triangles, sharing an edge e, then $[w] = [w]_e = w|_{T_+} - w|_{T_-}$ denotes the jump of w across e, while $\boldsymbol{t}$ is the unit tangent vector along e corresponding to the clockwise direction on T_+. Since $[\boldsymbol{v} \cdot \boldsymbol{n}]_e = 0$ for any $\boldsymbol{v} \in \boldsymbol{V}_h$ it follows from (2.2) and Green's theorem, in particular from (2.1), that

$$\begin{aligned} a_\varepsilon(\boldsymbol{u}, \boldsymbol{v}) + (p, \operatorname{div} \boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v}) + E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_h, \\ (\operatorname{div} \boldsymbol{u}, q) &= (g, q) \qquad \forall q \in L^2_0, \end{aligned} \tag{5.1}$$

where the term $E_{\varepsilon,h}$ appears due to the fact that $\boldsymbol{V}_h \not\subseteq \boldsymbol{H}^1_0$.

In the error analysis below we will need proper estimates on the consistency error $E_{\varepsilon,h}$. The following bounds are therefore useful.

Lemma 5.1. *If $\boldsymbol{u} \in \boldsymbol{H}^2 \cap \boldsymbol{H}^1_0$ then*

$$\sup_{\boldsymbol{v} \in \boldsymbol{V}_h} \frac{|E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{v})|}{\|\boldsymbol{v}\|_a} \le c\,\varepsilon \begin{cases} h \| \operatorname{rot} \boldsymbol{u} \|_1 \\ h^{1/2} \| \operatorname{rot} \boldsymbol{u} \|_1^{1/2} \| \operatorname{rot} \boldsymbol{u} \|_0^{1/2}, \end{cases}$$

where $c > 0$ is independent of ε and h.

Proof. Let $e \in \mathcal{E}_h$ and $\boldsymbol{v} \in \boldsymbol{H}^1_0 + \boldsymbol{V}_h$. Since the mean value with respect to e of $\boldsymbol{v} \cdot \boldsymbol{t}$ is zero, it follows from a standard scaling argument, cf. for example [5, Section 8.3] or [14, Section 4] for similar arguments, that

for any $\phi \in H^1$

$$\begin{aligned} \int_e \phi[\boldsymbol{v}\cdot\boldsymbol{t}]d\tau &\le \inf_{\lambda,\mu\in\mathbb{R}} \|\phi-\lambda\|_{0,e}\|[\boldsymbol{v}\cdot\boldsymbol{t}-\mu]\|_{0,e} \\ &\le \begin{cases} ch|\phi|_{1,\Omega_e}(|\boldsymbol{v}|_{1,T_-}+|\boldsymbol{v}|_{1,T_+}) \\ ch^{1/2}|\phi|^{1/2}_{1,\Omega_e}\|\phi\|^{1/2}_{0,\Omega_e}(|\boldsymbol{v}|_{1,T_-}+|\boldsymbol{v}|_{1,T_+}). \end{cases} \end{aligned} \tag{5.2}$$

Here T_- and T_+ denote the two triangles meeting the edge e and $\Omega_e = T_- \cup T_+$. Since

$$|E_{\varepsilon,h}(\boldsymbol{u},\boldsymbol{v})| \le \varepsilon^2 \sum_{e\in\mathcal{E}_h} |\int_e (\operatorname{rot}\boldsymbol{u})\ [\boldsymbol{v}\cdot\boldsymbol{t}]\,d\tau|,$$

the desired estimate follows by applying the estimate (5.2) with $\phi = \operatorname{rot}\boldsymbol{u}$, summing over all edges, and using the fact that

$$\sum_{e\in\mathcal{E}_h} |\boldsymbol{v}|^2_{1,T} \le \varepsilon^{-2} a_\varepsilon(\boldsymbol{v},\boldsymbol{v}).$$

□

Let $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_h \times Q_h$ be the approximation of $(\boldsymbol{u}, p)$ derived from the discrete system (3.1). From (3.1) and (5.1) we obtain

$$a_\varepsilon(\boldsymbol{u}-\boldsymbol{u}_h,\boldsymbol{v}) + (p-p_h, \operatorname{div}\boldsymbol{v}) = E_{\varepsilon,h}(\boldsymbol{u},\boldsymbol{v}) \tag{5.3}$$

for all $\boldsymbol{v} \in \boldsymbol{V}_h$. Furthermore,

$$\operatorname{div}\boldsymbol{u}_h = P_h \operatorname{div}\boldsymbol{u} = \operatorname{div}\boldsymbol{\Pi}_h\boldsymbol{u}.$$

Therefore, taking $\boldsymbol{v} = \boldsymbol{\Pi}_h\boldsymbol{u} - \boldsymbol{u}_h$ in (5.3) we obtain

$$a_\varepsilon(\boldsymbol{u}-\boldsymbol{u}_h, \boldsymbol{\Pi}_h\boldsymbol{u}-\boldsymbol{u}_h) = E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{\Pi}_h\boldsymbol{u}-\boldsymbol{u}_h).$$

Since a_ε is an inner product we further have

$$\begin{aligned} \|\boldsymbol{\Pi}_h\boldsymbol{u}-\boldsymbol{u}_h\|^2_a &\le \|\boldsymbol{u}-\boldsymbol{\Pi}_h\boldsymbol{u}\|^2_a + 2a_\varepsilon(\boldsymbol{u}-\boldsymbol{u}_h, \boldsymbol{\Pi}_h\boldsymbol{u}-\boldsymbol{u}_h) \\ &\le \|\boldsymbol{u}-\boldsymbol{\Pi}_h\boldsymbol{u}\|^2_a + 2E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{\Pi}_h\boldsymbol{u}-\boldsymbol{u}_h), \end{aligned}$$

Hence, we conclude that

$$\|\boldsymbol{u}-\boldsymbol{u}_h\|_a \le 2(\|\boldsymbol{u}-\boldsymbol{\Pi}_h\boldsymbol{u}\|_a + \sup_{\boldsymbol{v}\in\boldsymbol{V}_h} \frac{|E_{\varepsilon,h}(\boldsymbol{u},\boldsymbol{v})|}{\|\boldsymbol{v}\|_a}). \tag{5.4}$$

From this basic bound we easily derive the following error estimate.

Theorem 5.1. *If $\boldsymbol{u} \in \boldsymbol{H}^2 \cap \boldsymbol{H}^1_0$ and $p \in H^1 \cap L^2_0$ then the following estimates hold:*

$$\begin{aligned} \|\boldsymbol{u}-\boldsymbol{u}_h\|_0 + \varepsilon\|\operatorname{rot}(\boldsymbol{u}-\boldsymbol{u}_h)\|_0 &\le c(h^2+\varepsilon h)\|\boldsymbol{u}\|_2, \\ \|\operatorname{div}(\boldsymbol{u}-\boldsymbol{u}_h)\|_0 &\le ch\|\operatorname{div}\boldsymbol{u}\|_1 \\ \|p-p_h\|_0 &\le ch(\|p\|_1 + (\varepsilon+h)\|\boldsymbol{u}\|_2). \end{aligned}$$

Here $c > 0$ is a constant independent of ε and h.

Remark: Here, and below, the differential operators $\boldsymbol{D}$ and rot, applied to vector fiels in $\boldsymbol{V}_h$, are defined locally on each triangle of the triangulation $\mathcal{T}_h$. □

Proof. The first estimate is a direct consequence of (4.5), (5.4), and Lemma 5.1. The second estimate follows from the bound (4.3), and the fact that $\operatorname{div} \boldsymbol{u}_h = P_h \operatorname{div} \boldsymbol{u}$.

In order to establish the third estimate we first observe that (4.3) implies that

$$\|p - P_h p\|_0 \leq ch\|p\|_1. \tag{5.5}$$

Hence, it only remains to estimate $P_h p - p_h$. However, from the modified inf–sup condition (3.2), cf. Theorem 4.1, we obtain

$$\|P_h p - p_h\|_0 \leq \alpha^{-1} \sup_{\boldsymbol{v} \in \boldsymbol{V}_h} \frac{(P_h p - p_h, \operatorname{div} \boldsymbol{v})}{\|\!|\boldsymbol{v}|\!\|_{\varepsilon,h}}.$$

Furthermore, for any $\boldsymbol{v} \in \boldsymbol{V}_h$ we have

$$\begin{aligned}(P_h p - p_h, \operatorname{div} \boldsymbol{v}) &= (p - p_h, \operatorname{div} \boldsymbol{v}) \\ &= -a_\varepsilon(\boldsymbol{u} - \boldsymbol{u}_h, \boldsymbol{v}) + E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{v}),\end{aligned}$$

which implies that

$$|(P_h p - p_h, \operatorname{div} \boldsymbol{v})| \leq (\|\boldsymbol{u} - \boldsymbol{u}_h\|_a + \sup_{\boldsymbol{v} \in \boldsymbol{V}_h} \frac{|E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{v})|}{\|\boldsymbol{v}\|_a}) \|\!|\boldsymbol{v}|\!\|_{\varepsilon,h},$$

or

$$\|P_h p - p_h\|_0 \leq \alpha^{-1}(\|\boldsymbol{u} - \boldsymbol{u}_h\|_a + \sup_{\boldsymbol{v} \in \boldsymbol{V}_h} \frac{|E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{v})|}{\|\boldsymbol{v}\|_a}). \tag{5.6}$$

From the previous estimates we therefore obtain

$$\|P_h p - p_h\|_0 \leq c(h^2 + \varepsilon h)\|\boldsymbol{u}\|_2,$$

and together with (5.5) this establishes the desired estimate on the error $\|p - p_h\|_0$. □

Remark: As an alternative to the estimates given in Theorem 5.1 above we can also obtain

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|_0 + \varepsilon \|\operatorname{rot}(\boldsymbol{u} - \boldsymbol{u}_h)\|_0 \leq ch(\|\boldsymbol{u}\|_1 + \varepsilon\|\boldsymbol{u}\|_2) \tag{5.7}$$

and

$$\|p - p_h\|_0 \leq ch(\|p\|_1 + \|\boldsymbol{u}\|_1 + \varepsilon\|\boldsymbol{u}\|_2). \tag{5.8}$$

These modifications are obtained if we use the estimate

$$\|\boldsymbol{u} - \boldsymbol{\Pi}_h \boldsymbol{u}\|_0 \leq ch\|\boldsymbol{u}\|_1,$$

obtained from (4.5), in (5.4) instead of the corresponding quadratic estimate. Even if the modified estimates are weaker for uniformly smooth solutions, they are sometimes preferable for more singular solutions. □

6. Boundary layers and uniform error estimates

In general, we cannot expect that the norm $\|\boldsymbol{u}\|_2$ of the solution of (1.1) is bounded independently of ε. In fact, as ε approach zero even $\|\operatorname{rot}\boldsymbol{u}\|_0$ should be expected to blow up. Hence, the convergence estimates given in Theorem 5.1 will deteriorate as ε becomes small. The following example shows that this behavior of the error is in fact real.

Example 6.1 In this example we study the convergence for an ε dependent solution. Let $\boldsymbol{u} = \varepsilon\,\mathbf{curl}\, e^{-x_1x_2/\varepsilon}$, $p = \varepsilon e^{-x_1/\varepsilon}$, $\boldsymbol{f} = \boldsymbol{u} - \varepsilon^2\boldsymbol{\Delta u} - \mathbf{grad}\, p$ and g identical zero. In fact, $\boldsymbol{u}$ is not the solution of the corresponding system (1.1), since the boundary conditions are not satisfied. However, the adaption of the new method to nonhomogeneous boundary conditions is straightforward.

The significance of the solution $\boldsymbol{u}$ just given is related to the fact that the quantities $\|\operatorname{rot}\boldsymbol{u}\|_0$ and $\varepsilon\|\operatorname{rot}\boldsymbol{u}\|_1$ both are of order $\varepsilon^{-1/2}$ as ε tends to zero. As we will see below, in Lemma 6.1, this behavior is typical for solutions of the singular perturbation problem (1.1). For solutions with this singular behavior the estimates (5.7) and (5.8) leads to error bounds of the form

$$|||\boldsymbol{u} - \boldsymbol{u}_h|||_\varepsilon, \|p - p_h\|_0 \le ch\varepsilon^{-1/2}, \tag{6.1}$$

where c is a constant independent of ε and h. In Table 6.1 below we have computed the absolute error, $|||\boldsymbol{u} - \boldsymbol{u}_h|||_\varepsilon$ for different values of ε and h. For each fixed ε the convergence rate with respect to h is estimated.

$\varepsilon\backslash$ h	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
2^{-2}	7.29e-2	3.60e-2	1.77e-2	8.75e-3	4.36e-3	0.98
2^{-6}	8.89e-2	5.88e-2	3.71e-2	2.06e-2	1.05e-2	0.77
2^{-8}	1.12e-1	6.89e-2	4.07e-2	2.66e-2	1.73e-2	0.67
2^{-10}	1.17e-1	8.16e-2	5.48e-2	3.34e-2	1.93e-2	0.65
2^{-12}	1.17e-1	8.20e-2	5.74e-2	4.02e-2	2.71e-2	0.52

Table 6.1. The absolute error in velocity, measured in the energy norm, obtained by the new nonconforming element.

We observe that for ε sufficiently large the convergence rate is approximately one, but that the estimated rate decreases when ε approach zero. These results seem to confirm the claim that the convergence is linear with respect to h for each fixed ε. However, when h is sufficiently large compared to ε we do not observe this linear rate.

In Table 6.2 we give the corresponding relative L^2 errors for the pressure.

$\varepsilon \backslash$ h	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
2^{-2}	2.32e-2	1.11e-2	5.36e-3	2.64e-3	1.31e-3	1.04
2^{-6}	9.00e-3	5.33e-3	2.62e-3	1.15e-3	4.61e-4	1.07
2^{-8}	5.28e-3	3.24e-3	2.18e-3	1.23e-3	5.97e-4	0.77
2^{-10}	4.93e-3	2.54e-3	1.33e-3	7.93e-4	5.32e-4	0.81
2^{-12}	4.92e-3	2.51e-3	1.24e-3	6.22e-4	3.27e-4	0.98

TABLE 6.2. The L^2 error in the pressure obtained by the new nonconforming element.

Again the estimated convergence rate is approximately one for ε large. Then it starts to decrease with ε as in Table 6.1. However, in this case the convergence rate increases roughly back to one when ε is super close to zero. We will comment on this phenomenon for the error of the pressure at the end of this section.

The estimate 6.1 does not imply uniform convergence with respect to ε for our new finite element method. However, as a consequence of the theory below, we will obtain an improved estimate of the form

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|_\varepsilon, \|p - p_h\|_0 \leq c \min(h^{1/2}, h\varepsilon^{-1/2}), \tag{6.2}$$

for solutions with a singular behavior similar to the solution $\boldsymbol{u}$ studied here. Note that this is in fact consistent with the results of Tables 6.1 and 6.2, where we never observe a convergence rate below a half. □

The main purpose of this section is to establish error estimates which are uniform with respect to the perturbation parameter ε. We shall show a uniform $O(h^{1/2})$ error estimate in the energy norm. We observe that if $g \in H^1 \cap L_0^2$ then it follows directly from Theorem 5.1 that

$$\|\operatorname{div}(\boldsymbol{u} - \boldsymbol{u}_h)\|_0 \leq c\, h \|g\|_1, \tag{6.3}$$

where the constant c is independent of ε and h. Hence, we have uniform linear convergence for the error of the divergence. In contrast to this, the remaining part of the error will be affected by boundary layers as ε becomes small. However, the following uniform convergence estimate will be derived.

Theorem 6.1. *If $\boldsymbol{f} \in \boldsymbol{H}(\mathrm{rot})$ and $g \in H_+^1$ then there is a constant c, independent of $\boldsymbol{f}$, g, ε, and h such that*

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|_0 + \varepsilon \|\operatorname{rot}(\boldsymbol{u} - \boldsymbol{u}_h)\|_0 + \|p - p_h\|_0 \leq c\, h^{1/2}(\|\boldsymbol{f}\|_{\mathrm{rot}} + \|g\|_{1,+}).$$

Here the Sobolev space H_+^1 is a space contained in H^1, with associated norm, $\|\cdot\|_{1,+}$, slightly stronger than $\|\cdot\|_1$. This space will be precisely defined below.

The derivation of the uniform error estimate above will depend heavily on certain regularity estimates for the solution of the system (1.1). For example, we shall estimate the blow up of $\|\operatorname{rot} \boldsymbol{u}\|_1$ as ε approach zero. We shall therefore first derive these regularity estimates.

For convenience of the reader we repeat the system (1.1):

$$\begin{aligned} (\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta})\boldsymbol{u} - \mathbf{grad}\, p &= \boldsymbol{f} \quad \text{in } \Omega, \\ \operatorname{div} \boldsymbol{u} &= g \quad \text{in } \Omega, \\ \boldsymbol{u} &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{6.4}$$

We also repeat that the domain Ω is a polygonal domain in $\mathbb{R}^2$. In fact, in the discussion of this section we shall assume that Ω in addition is *convex*. If $\varepsilon \in (0,1]$, $\boldsymbol{f} \in \boldsymbol{L}^2$ and $g = 0$ then the corresponding weak solution admits the additional regularity that $(\boldsymbol{u}, p) \in (\boldsymbol{H}_0^1 \times L_0^2) \cap (\boldsymbol{H}^2 \times H^1)$. This regularity result follows directly from the result for the corresponding Stokes problem on a convex domain which can be found in [12, Corollary 7.3.3.5]. In fact, the same regularity holds for $g \neq 0$ if we restrict the data g to the space H_+^1.

In order to define this space let $x_1, x_2, \ldots, x_N \in \partial\Omega$ denote the vertices of Ω. The space H_+^1 is given by

$$H_+^1 = \{g \in H^1 \cap L_0^2 : \int_\Omega \frac{|g(x)|^2}{|x - x_j|^2}\, dx < \infty,\ j = 1, 2, \ldots, N\},$$

with associated norm

$$\|g\|_{1,+}^2 = \|g\|_1^2 + \sum_{j=1}^{N} \int_\Omega \frac{|g(x)|^2}{|x - x_j|^2}\, dx.$$

Hence, functions in H_+^1 vanish weakly at each vertex of Ω.

It is established in [2] that

$$\operatorname{div}(\boldsymbol{H}^2 \cap \boldsymbol{H}_0^1) = H_+^1.$$

Furthermore, the divergence operator has a bounded right inverse, $\boldsymbol{R} : H_+^1 \mapsto \boldsymbol{H}^2 \cap \boldsymbol{H}_0^1$, i.e. $\operatorname{div} \boldsymbol{R}g = g$ for all $g \in H_+^1$ and

$$\|\boldsymbol{R}g\|_2 \leq c\|g\|_{1,+}.$$

Note that if $(\boldsymbol{u}, p)$ solves (6.4) then $(\boldsymbol{u} - \boldsymbol{R}g, p)$ solves a corresponding problem with $g = 0$. From the result in the case $g = 0$ we can therefore conclude that $(\boldsymbol{u}, p) \in (\boldsymbol{H}_0^1 \times L_0^2) \cap (\boldsymbol{H}^2 \times H^1)$ for any $(\boldsymbol{f}, g) \in \boldsymbol{L}^2 \times H_+^1$.

The following result gives an upper bound for the blow up of the norm $\|\operatorname{rot} \boldsymbol{u}\|_1$ as ε tends to zero.

Lemma 6.1. *Assume that $\boldsymbol{f} \in \boldsymbol{H}(\mathrm{rot})$, $g \in H_+^1$, and let $(\boldsymbol{u}, p)$ be the corresponding solution of* (6.4). *There exist a constant $c > 0$, independent of ε, $\boldsymbol{f}$ and g, such that*

$$\varepsilon^{1/2}\|\operatorname{rot} \boldsymbol{u}\|_0 + \varepsilon^{3/2}\|\operatorname{rot} \boldsymbol{u}\|_1 \leq c\,(\|\operatorname{rot} \boldsymbol{f}\|_0 + \|g\|_{1,+}). \tag{6.5}$$

Proof. We first construct a function $\hat{\boldsymbol{u}} \in \boldsymbol{H}^2 \cap \boldsymbol{H}_0^1$ such that

$$\operatorname{div} \hat{\boldsymbol{u}} = g, \qquad \text{and} \qquad \operatorname{rot} \boldsymbol{\Delta}\hat{\boldsymbol{u}} = 0. \tag{6.6}$$

In fact, the function $\hat{\boldsymbol{u}}$ can be constructed by defining

$$\hat{\boldsymbol{u}} = \boldsymbol{R}g + \mathbf{curl}\, \psi,$$

with $\psi \in H_0^2$ being the weak solution of the biharmonic equation

$$\begin{aligned} \Delta^2 \psi &= \operatorname{rot} \boldsymbol{\Delta R} g \qquad \text{in } \Omega, \\ \psi = \frac{\partial \psi}{\partial \boldsymbol{n}} &= 0 \qquad \text{on } \partial\Omega. \end{aligned}$$

We observe that, since $\boldsymbol{R}g \in \boldsymbol{H}^2$, the right hand side is in H^{-1}. Therefore, from the regularity of solutions of the biharmonic equation on convex domains, cf. [12, Theorem 7.2.2.3], we have that $\psi \in H^3$, and $\|\psi\|_3 \leq c\| \operatorname{rot} \boldsymbol{\Delta R} g\|_{-1}$. Hence, $\hat{\boldsymbol{u}} \in \boldsymbol{H}^2 \cap \boldsymbol{H}_0^1$, and

$$\|\hat{\boldsymbol{u}}\|_2 \leq c \, \|g\|_{1,+}. \tag{6.7}$$

Furthermore, clearly $\operatorname{div} \hat{\boldsymbol{u}} = \operatorname{div} \boldsymbol{R}g = g$, and for any $\mu \in C_0^\infty$ we have

$$(\boldsymbol{\Delta} \hat{\boldsymbol{u}}, \mathbf{curl}\, \mu) = (\boldsymbol{\Delta R} g, \mathbf{curl}\, \mu) - (\Delta \psi, \Delta \mu) = 0.$$

Hence, the second property in (6.6) also holds.

Define $\boldsymbol{v} = \boldsymbol{u} - \hat{\boldsymbol{u}}$. Then $(\boldsymbol{v}, p) \in (\boldsymbol{H}_0^1 \times L_0^2) \cap (\boldsymbol{H}^2 \times H^1)$ is the weak solution of the problem

$$\begin{aligned} (\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta}) \boldsymbol{v} - \mathbf{grad}\, p &= \hat{\boldsymbol{f}} \quad \text{in } \Omega, \\ \operatorname{div} \boldsymbol{v} &= 0 \quad \text{in } \Omega, \\ \boldsymbol{v} &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{6.8}$$

where $\hat{\boldsymbol{f}} = \boldsymbol{f} + \varepsilon^2 \boldsymbol{\Delta} \hat{\boldsymbol{u}} - \hat{\boldsymbol{u}}$. Clearly, $\hat{\boldsymbol{f}} \in \boldsymbol{L}^2$. In fact, $\hat{\boldsymbol{f}} \in \boldsymbol{H}(\operatorname{rot})$, since

$$\operatorname{rot} \hat{\boldsymbol{f}} = \operatorname{rot} \boldsymbol{f} - \operatorname{rot} \hat{\boldsymbol{u}}.$$

Furthermore, there is a constant c, independent of ε, $\boldsymbol{f}$ and g, such that

$$\| \operatorname{rot} \hat{\boldsymbol{f}}\|_0 \leq c \, (\| \operatorname{rot} \boldsymbol{f}\|_0 + \|g\|_{1,+}). \tag{6.9}$$

Since $\boldsymbol{v} \in \boldsymbol{L}^2$ and $\operatorname{div} \boldsymbol{v} = 0$ there exists $\phi \in H^1$, uniquely determined up to a constant, such that $\boldsymbol{v} = \mathbf{curl}\, \phi$ ([11, Theorem I.3.1]). Hence, since $\boldsymbol{v} \in \boldsymbol{H}^2 \cap \boldsymbol{H}_0^1$, we can choose $\phi \in H^3 \cap H_0^2$. In fact, by applying the rot operator, as a map from $\boldsymbol{L}^2$ to H^{-1}, to the first equation of (6.8) we obtain

$$\begin{aligned} -\Delta \phi + \varepsilon^2 \Delta^2 \phi &= \operatorname{rot} \hat{\boldsymbol{f}} \qquad \text{in } \Omega, \\ \phi = \frac{\partial \phi}{\partial \boldsymbol{n}} &= 0 \qquad \text{on } \partial\Omega. \end{aligned}$$

The function ϕ is uniquely determined by this problem. This singular perturbation problem was in fact studied in [14], where it was established that ([14, Lemma 5.1])

$$\varepsilon^{1/2} \|\phi\|_2 + \varepsilon^{3/2} \|\phi\|_3 \leq c \| \operatorname{rot} \hat{\boldsymbol{f}}\|_0,$$

and as a consequence

$$\varepsilon^{1/2} \| \operatorname{rot} \boldsymbol{v}\|_0 + \varepsilon^{3/2} \| \operatorname{rot} \boldsymbol{v}\|_1 \leq c \| \operatorname{rot} \hat{\boldsymbol{f}}\|_0.$$

Therefore, since $\boldsymbol{u} = \boldsymbol{v} + \hat{\boldsymbol{u}}$, (6.7) and (6.9) implies

$$\begin{aligned}\varepsilon^{1/2}\|\operatorname{rot}\boldsymbol{u}\|_0 + \varepsilon^{3/2}\|\operatorname{rot}\boldsymbol{u}\|_1 &\le c\|\operatorname{rot}\hat{\boldsymbol{f}}\|_0 + \varepsilon^{1/2}(\|\operatorname{rot}\hat{\boldsymbol{u}}\|_0 + \varepsilon\|\operatorname{rot}\hat{\boldsymbol{u}}\|_1)\\ &\le c(\|\operatorname{rot}\boldsymbol{f}\|_0 + \|g\|_{1,+}).\end{aligned}$$

This completes the proof. □

In addition to the ε–dependent bound on the solution $(\boldsymbol{u}, p)$ of (6.4) derived above, we shall also need convergence estimates on how fast these solutions converge to the solution of the reduced system.

The reduced system corresponding to (6.4) is of the form

$$\begin{aligned}\boldsymbol{u}^0 - \mathbf{grad}\, p^0 &= \boldsymbol{f} \quad \text{in } \Omega,\\ \operatorname{div}\boldsymbol{u}^0 &= g \quad \text{in } \Omega,\\ \boldsymbol{u}^0\cdot\boldsymbol{n} &= 0 \quad \text{on } \partial\Omega,\end{aligned} \tag{6.10}$$

A precise weak formulation of this system is given by:

Find $(\boldsymbol{u}^0, p^0) \in \boldsymbol{H}_0(\operatorname{div}) \times L^2_0$ such that

$$\begin{aligned}(\boldsymbol{u}^0, \boldsymbol{v}) + (p^0, \operatorname{div}\boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{H}_0(\operatorname{div})\\ (\operatorname{div}\boldsymbol{u}^0, q) &= (g, q) \qquad \forall q \in L^2_0.\end{aligned} \tag{6.11}$$

If $(\boldsymbol{f}, g) \in \boldsymbol{H}^{-1}(\operatorname{rot}) \times L^2_0$ then this system admits a unique solution. In fact, if $\boldsymbol{f} \in \boldsymbol{H}(\operatorname{rot})$ then $\boldsymbol{u}^0 \in \boldsymbol{H}(\operatorname{rot})$ with $\operatorname{rot}\boldsymbol{u}^0 = \operatorname{rot}\boldsymbol{f}$. Therefore,

$$\boldsymbol{u}^0 \in \boldsymbol{H}_0(\operatorname{div}) \cap \boldsymbol{H}(\operatorname{rot}),$$

and hence, cf. [11, Proposition 3.1, Chap. 1], $\boldsymbol{u}^0 \in \boldsymbol{H}^1$. As a consequence, $p^0 \in H^1$. Furthermore, the corresponding solution map is continuous, i.e. there exist a constant c, independent of $\boldsymbol{f}$ and g, such that

$$\|\boldsymbol{u}^0\|_1 + \|p^0\|_1 \le c\,(\|\boldsymbol{f}\|_{\operatorname{rot}} + \|g\|_0). \tag{6.12}$$

Lemma 6.2. *Assume that $\boldsymbol{f} \in \boldsymbol{H}(\operatorname{rot})$, $g \in H^1_+$, and let $(\boldsymbol{u}, p)$ be the corresponding solution of* (6.4). *There exist a constant $c > 0$, independent of ε, $\boldsymbol{f}$ and g, such that*

$$\|\boldsymbol{u} - \boldsymbol{u}^0\|_0 + \|p - p^0\|_1 \le c\,\varepsilon^{1/2}(\|\boldsymbol{f}\|_{\operatorname{rot}} + \|g\|_{1,+}).$$

Proof. It follows from (2.2), the weak formulation of (6.4), and Green's theorem that for any $\boldsymbol{v} \in \boldsymbol{H}^1 \cap \boldsymbol{H}_0(\operatorname{div})$ the solution $(\boldsymbol{u}, p)$ satisfies

$$\begin{aligned}(\boldsymbol{u}, \boldsymbol{v}) + \varepsilon^2(\operatorname{div}\boldsymbol{u}, \operatorname{div}\boldsymbol{v}) + \varepsilon^2(\operatorname{rot}\boldsymbol{u}, \operatorname{rot}\boldsymbol{v}) + \varepsilon^2\int_{\partial\Omega}(\operatorname{rot}\boldsymbol{u})(\boldsymbol{v}\cdot\boldsymbol{t})d\tau\\ + (p, \operatorname{div}\boldsymbol{v}) = (\boldsymbol{f}, \boldsymbol{v}).\end{aligned}$$

By subtracting from this the first equation of (6.11), we obtain

$$(\boldsymbol{u} - \boldsymbol{u}^0, \boldsymbol{v}) + \varepsilon^2(\operatorname{rot}\boldsymbol{u}, \operatorname{rot}\boldsymbol{v}) + \varepsilon^2\int_{\partial\Omega}(\operatorname{rot}\boldsymbol{u})(\boldsymbol{v}\cdot\boldsymbol{t})d\tau = 0$$

for any $\boldsymbol{v} \in \boldsymbol{H}^1 \cap \boldsymbol{H}_0(\mathrm{div})$ with $\mathrm{div}\, \boldsymbol{v} = 0$. Hence, if we take $\boldsymbol{v} = \boldsymbol{u} - \boldsymbol{u}^0$, and observe that $\mathrm{rot}\, \boldsymbol{u}^0 = \mathrm{rot}\, \boldsymbol{f}$ and $\mathrm{div}(\boldsymbol{u} - \boldsymbol{u}^0) = 0$, we derive the identity

$$\|\boldsymbol{u} - \boldsymbol{u}^0\|_0^2 + \varepsilon^2 \|\mathrm{rot}\, \boldsymbol{u}\|_0^2 = \varepsilon^2 \int_{\partial\Omega} (\mathrm{rot}\, \boldsymbol{u})(\boldsymbol{u}^0 \cdot \boldsymbol{t}) d\tau + \varepsilon^2 (\mathrm{rot}\, \boldsymbol{u}, \mathrm{rot}\, \boldsymbol{f}),$$

which immediately leads to the bound

$$\|\boldsymbol{u} - \boldsymbol{u}^0\|_0^2 + \frac{\varepsilon^2}{2} \|\mathrm{rot}\, \boldsymbol{u}\|_0^2 \leq \varepsilon^2 \|\mathrm{rot}\, \boldsymbol{f}\|_0^2 + \varepsilon^2 \int_{\partial\Omega} (\mathrm{rot}\, \boldsymbol{u})(\boldsymbol{u}^0 \cdot \boldsymbol{t}) d\tau. \tag{6.13}$$

In order to estimate the boundary integral we note that it follows from Lemma 6.1 and [12, Theorem 1.5.1.10] that

$$\|\mathrm{rot}\, \boldsymbol{u}\|_{0,\partial\Omega} \leq c \|\mathrm{rot}\, \boldsymbol{u}\|_0^{1/2} \|\mathrm{rot}\, \boldsymbol{u}\|_1^{1/2} \leq c\varepsilon^{-1} (\|\mathrm{rot}\, \boldsymbol{f}\|_0 + \|g\|_{1,+}).$$

Together with the estimate (6.12) this leads to

$$\begin{aligned} \varepsilon^2 \int_{\partial\Omega} (\mathrm{rot}\, \boldsymbol{u})(\boldsymbol{u}^0 \cdot \boldsymbol{t}) d\tau &\leq \varepsilon^2 \|\mathrm{rot}\, \boldsymbol{u}\|_{0,\partial\Omega} \|\boldsymbol{u}^0\|_1 \\ &\leq c\, \varepsilon (\|\boldsymbol{f}\|_{\mathrm{rot}}^2 + \|g\|_{1,+}^2). \end{aligned}$$

Hence, the estimate

$$\|\boldsymbol{u} - \boldsymbol{u}^0\|_0 + \varepsilon^2 \|\mathrm{rot}\, \boldsymbol{u}\|_0^2 \leq c\, \varepsilon^{1/2} (\|\boldsymbol{f}\|_{\mathrm{rot}} + \|g\|_{1,+}) \tag{6.14}$$

follows.

The estimate for $\|p - p^0\|_1$ is now a direct consequence of the identity

$$\begin{aligned} \mathbf{grad}(p - p^0) &= \boldsymbol{u} - \boldsymbol{u}^0 - \varepsilon^2 \boldsymbol{\Delta} \boldsymbol{u} \\ &= \boldsymbol{u} - \boldsymbol{u}^0 + \varepsilon^2 (\mathbf{curl}\, \mathrm{rot}\, \boldsymbol{u} - \mathbf{grad}\, g) \end{aligned}$$

and the previously established bounds. In fact, it follows from Lemma 6.1 and (6.14) that

$$\begin{aligned} \|\mathbf{grad}(p - p^0)\|_0 &\leq \|\boldsymbol{u} - \boldsymbol{u}^0\|_0 + \varepsilon^2 (\|\mathrm{rot}\, \boldsymbol{u}\|_1 + \|g\|_1) \\ &\leq c\, \varepsilon^{1/2} (\|\boldsymbol{f}\|_{\mathrm{rot}} + \|g\|_{1,+}). \end{aligned}$$

Since $p - p^0 \in L_0^2$, an application of the Poincaré inequality completes the proof. □

The regularity bounds derived above will now be used to prove the uniform convergence estimates.

Proof of Theorem 6.1. Recall that since $\boldsymbol{u} \in \boldsymbol{H}_0^1$ it follows from [11, Proposition 3.1, Chap. 1] that

$$\|\boldsymbol{u}\|_1 \leq c(\|\mathrm{div}\, \boldsymbol{u}\|_0 + \|\mathrm{rot}\, \boldsymbol{u}\|_0).$$

Furthermore, by the standard H^2–regularity for solutions of the Poisson equation on convex domains, and (2.2), we obtain

$$\|\boldsymbol{u}\|_2 \leq c \|\boldsymbol{\Delta} \boldsymbol{u}\|_0 \leq c(\|\mathrm{div}\, \boldsymbol{u}\|_1 + \|\mathrm{rot}\, \boldsymbol{u}\|_1).$$

Hence, from the estimates given in Lemmas 6.1 and 6.2 we conclude that

$$\varepsilon^2\|\boldsymbol{u}\|_2+\varepsilon\|\boldsymbol{u}\|_1+\|\boldsymbol{u}-\boldsymbol{u}^0\|_0+\|p-p^0\|_1 \le c\varepsilon^{1/2}(\|\boldsymbol{f}\|_{\mathrm{rot}}+\|g\|_{1,+}). \tag{6.15}$$

The desired estimate on the velocity error will be derived from (5.4). We will first establish the interpolation estimate

$$\|\boldsymbol{u}-\boldsymbol{\Pi}_h\boldsymbol{u}\|_0+\varepsilon\|\boldsymbol{D}(\boldsymbol{u}-\boldsymbol{\Pi}_h\boldsymbol{u})\|_1 \le c\,h^{1/2}(\|\boldsymbol{f}\|_{\mathrm{rot}}+\|g\|_{1,+}). \tag{6.16}$$

From (4.6), (6.12), and (6.15) we have

$$\begin{aligned}\|\boldsymbol{u}-\boldsymbol{\Pi}_h\boldsymbol{u}\|_0 &\le \|(\boldsymbol{I}-\boldsymbol{\Pi}_h)(\boldsymbol{u}-\boldsymbol{u}^0)\|_0+\|\boldsymbol{u}^0-\boldsymbol{\Pi}_h\boldsymbol{u}^0\|_0\\ &\le ch^{1/2}\left(\|\boldsymbol{u}-\boldsymbol{u}^0\|_0^{1/2}\|\boldsymbol{u}-\boldsymbol{u}^0\|_1^{1/2}+h^{1/2}\,\|\boldsymbol{u}^0\|_1\right)\\ &\le ch^{1/2}(\|\boldsymbol{f}\|_{\mathrm{rot}}+\|g\|_{1,+}).\end{aligned}$$

Furthermore, from (4.4), (4.5), and (6.15),

$$\begin{aligned}\varepsilon\|\boldsymbol{D}(\boldsymbol{u}-\boldsymbol{\Pi}_h\boldsymbol{u})\|_0 \le c\varepsilon\|\boldsymbol{u}\|_1^{1/2}\|\boldsymbol{u}-\boldsymbol{\Pi}_h\boldsymbol{u}\|_1^{1/2} &\le c\varepsilon h^{1/2}\|\boldsymbol{u}\|_1^{1/2}\|\boldsymbol{u}\|_2^{1/2}\\ &\le ch^{1/2}(\|\boldsymbol{f}\|_{\mathrm{rot}}+\|g\|_{1,+}).\end{aligned}$$

The estimate (6.16) is therefore verified.

Similarly, since $\|\boldsymbol{u}\|_1^{1/2}\|\boldsymbol{u}\|_2^{1/2} \le c\varepsilon^{-1}(\|\boldsymbol{f}\|_{\mathrm{rot}}+\|g\|_{1,+})$, we obtain from Lemma 5.1 that

$$\sup_{\boldsymbol{v}\in\boldsymbol{V}_h}\frac{|E_{\varepsilon,h}(\boldsymbol{u},\boldsymbol{v})|}{\|\boldsymbol{v}\|_a} \le ch^{1/2}(\|\boldsymbol{f}\|_{\mathrm{rot}}+\|g\|_{1,+}). \tag{6.17}$$

However, by combining (5.4), (6.3), (6.16), and (6.17), this implies

$$\|\boldsymbol{u}-\boldsymbol{u}_h\|_0+\varepsilon\|\operatorname{rot}(\boldsymbol{u}-\boldsymbol{u}_h)\|_0 \le ch^{1/2}(\|\boldsymbol{f}\|_{\mathrm{rot}}+\|g\|_{1,+}). \tag{6.18}$$

In order to establish the estimate for the $\|p-p_h\|_0$ note that (4.3) and (6.15) implies

$$\|P_hp-p\|_0 \le ch\|p\|_1 \le ch(\|\boldsymbol{f}\|_{\mathrm{rot}}+\|g\|_{1,+}).$$

Finally, by (5.6), (6.17), and (6.18),

$$\|P_hp-p_h\|_0 \le ch^{1/2}(\|\boldsymbol{f}\|_{\mathrm{rot}}+\|g\|_{1,+}).$$

This completes the proof of Theorem 6.1. □

Remark. Even if Lemma 6.2 states that $\|p\|_1$ is uniformly bounded with respect to ε, we are not able to prove that $\|p-p_h\|_0$ converges linearly in h uniformly in ε. The convergence rate is polluted by the blow up of $\boldsymbol{u}$. This seem to agree with what we observed in Example 6.1 above, cf. Table 6.2. □

7. An associated elliptic system

In this section we shall study the elliptic system (1.2) given by

$$
\begin{aligned}
(\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta})\boldsymbol{u} - \delta^{-2}\,\mathbf{grad}(\operatorname{div} \boldsymbol{u} - g) &= \boldsymbol{f} \quad \text{in } \Omega, \\
\boldsymbol{u} &= 0 \quad \text{on } \partial\Omega,
\end{aligned} \tag{7.1}
$$

where $\varepsilon, \delta \in (0, 1]$. Recall that by introducing $p = \delta^{-2} \operatorname{div} \boldsymbol{u}$ this system can be alternatively be written on the mixed form (1.3). Hence, as δ approach zero the system formally reduces to (1.1).

The system (7.1) will be discretized by a standard finite element approach, i.e. the mixed system (1.3) is not introduced in the discretization. Let the bilinear form $b_{\varepsilon,\delta}(\cdot,\cdot)$ be defined by

$$
\begin{aligned}
b_{\varepsilon,\delta}(\boldsymbol{u}, \boldsymbol{v}) &= a_\varepsilon(\boldsymbol{u}, \boldsymbol{v}) + \delta^{-2}(\operatorname{div} \boldsymbol{u}, \operatorname{div} \boldsymbol{v}) \\
&= (\boldsymbol{u}, \boldsymbol{v}) + \varepsilon^2(\boldsymbol{D}\boldsymbol{u}, \boldsymbol{D}\boldsymbol{v}) + \delta^{-2}(\operatorname{div} \boldsymbol{u}, \operatorname{div} \boldsymbol{v}).
\end{aligned}
$$

For a given finite element space $\boldsymbol{V}_h$, the corresponding standard finite element discretization of (7.1) is given by:

Find a $\boldsymbol{u}_h \in \boldsymbol{V}_h$ such that

$$
b_{\varepsilon,\delta}(\boldsymbol{u}_h, \boldsymbol{v}) = (\boldsymbol{f}, \boldsymbol{v}) + \delta^{-2}(g, \operatorname{div} \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_h. \tag{7.2}
$$

Our purpose here is to propose that the finite element space $\boldsymbol{V}_h$, introduced in §4 above, in used this discretization. Since this space is not a subspace of $\boldsymbol{H}_0^1$ this will lead to a nonconforming discretization of the system (7.1). However, before we analyze this discretization, we will present some numerical experiments based on the system (7.1).

Example 7.1 In all the examples presented in this section we consider the system (7.1) with $\boldsymbol{u} = \mathbf{curl} \sin^2(\pi x_1) \sin^2(\pi x_2)$, $g = 0$, and $\boldsymbol{f} = \boldsymbol{u} - \varepsilon^2 \boldsymbol{\Delta} \boldsymbol{u}$. Hence, the solution is independent of ε and δ.

We consider the problem (7.1) with Ω taken as the unit square. The domain is triangulated as described in Example 3.1. The system is then discretized by solving the system (7.2), where the space $\boldsymbol{V}_h$ is the standard space of continuous piecewise linear functions with respect to this triangulation.

In the present example we have used $\varepsilon = 1$, while δ and h varies. In Table 7.1 below we have computed the relative error in the L^2 norm for different values of δ and h.

$\delta \backslash h$	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
1.00	3.87e-1	1.32e-1	3.69e-2	9.52e-3	2.39e-3	1.85
0.10	9.19e-1	7.28e-1	4.34e-1	1.88e-1	6.20e-2	0.97
0.01	1.00	9.96e-1	9.82e-1	9.32e-1	7.88e-1	0.08

Table 7.1. The relative L^2 error using piecewise linear elements, $\varepsilon = 1$.

As expected we observe quadratic convergence with respect to h for $\delta = 1$. However the convergence clearly deteriorates as δ tends to zero. □

Example 7.2 We repeat the experiment above, but we extend the finite element space and use the corresponding velocity space of the Mini element instead of the piecewise linear space. It is interesting to note that the L^2 convergence deteriorates, as δ gets small, also in this case, in contrast to what we have observed in Table 3.7. The relative L^2 error is given in Table 7.2.

$\delta \backslash h$	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
1.00	3.80e-1	1.30e-1	3.62e-2	9.34e-3	2.35e-3	1.85
0.10	9.19e-1	7.28e-1	4.34e-1	1.88e-1	6.20e-2	0.97
0.01	9.99e-1	9.96e-1	9.82e-1	9.33e-1	7.88e-1	0.08

TABLE 7.2. The relative L^2 error using the Mini element, $\varepsilon = 1$.

We observe that the results are almost identical to the ones we obtained in the piecewise linear case. Hence, the extra bubble functions have almost no effect. Of course, the main reason for the difference between the results given here, for δ small, and the results given in Example 3.3, where $\delta = 0$, is that the second equation of the mixed method used previously implicitly introduces a reduced integration in the divergence term. □

Example 7.3 We repeat the experiment above once more, but this time we use the new nonconforming element. In Table 7.3 below we have computed the relative error in the energy norm, i.e. the norm generated by the form $b_{\varepsilon,\delta}$, for different values of δ and h.

$\delta \backslash h$	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
1.00	1.84	9.83e-1	4.98e-1	2.50e-1	1.25e-1	0.97
0.10	1.83	9.66e-1	4.87e-1	2.44e-1	1.22e-1	0.98
0.01	1.83	9.66e-1	4.87e-1	2.44e-1	1.22e-1	0.98

TABLE 7.3. The relative error in energy norm for the new nonconforming element, $\varepsilon = 1$.

In contrast to the other examples above, in this case the convergence seems to be linear with respect to h, uniformly in δ. We also observe that the errors are almost independent of δ.

Next, we reduce ε and take $\varepsilon = 0.01$ and redo the experiment. The results are given in Table 7.4.

We observe that to the given accuracy, the numerical solution is independent of δ, clearly indicating that the numerical solutions are

$\delta\backslash h$	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	rate
1.00	1.04e-1	3.23e-2	8.94e-3	2.21e-3	5.29e-4	1.91
0.10	1.04e-1	3.23e-2	8.94e-3	2.21e-3	5.29e-4	1.91
0.01	1.04e-1	3.23e-2	8.94e-3	2.21e-3	5.29e-4	1.91

TABLE 7.4. The relative error in energy norm for the new nonconforming element, $\varepsilon = 0.01$.

close to a pure curl field independent of δ, which is precisely the form of the exact solution in this case. A similar observation is done if we take $\varepsilon = 0$. □

The numerical experiments just presented indicate that the nonconforming space $\boldsymbol{V}_h$, introduced in §4 above, is well suited for the problem (7.1). We will give a partial theoretical justification for this claim by deriving a generalization of Theorem 5.1.

We assume throughout this section that $\boldsymbol{u} \in \boldsymbol{H}^2 \cap \boldsymbol{H}_0^1$. Let $\|\cdot\|_b$ be the energy norm associated with the system (7.1), i.e.

$$\|\boldsymbol{v}\|_b^2 = b_{\varepsilon,\delta}(\boldsymbol{v}, \boldsymbol{v}).$$

It is a straightforward consequence of the second Strang lemma, cf. [9, Theorem 4.2.2], that there exists a $c > 0$ independent of ε, h and $\boldsymbol{u}$ such that

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|_b^2 \leq \|\boldsymbol{u} - \boldsymbol{\Pi}_h \boldsymbol{u}\|_b^2 + c \sup_{\boldsymbol{v} \in \boldsymbol{V}_h} \frac{|E_{\varepsilon,h}(\boldsymbol{u}, \boldsymbol{v})|^2}{\|\boldsymbol{v}\|_b^2}, \tag{7.3}$$

where the inconsistency error $E_{\varepsilon,h}$ is introduced in §5 above. However, since $\|\boldsymbol{v}\|_b \geq \|\boldsymbol{v}\|_a$, the inconsistency term can be bounded as in Lemma 5.1. Furthermore, (4.5) implies

$$\|\boldsymbol{u} - \boldsymbol{\Pi}_h \boldsymbol{u}\|_a \leq c(h^2 + \varepsilon h)\|\boldsymbol{u}\|_2.$$

As a consequence of the fact that $\operatorname{div} \boldsymbol{\Pi}_h \boldsymbol{u} = P_h \operatorname{div} \boldsymbol{u}$, it is also true that

$$\|\operatorname{div}(\boldsymbol{u} - \boldsymbol{u}_h)\|_0^2 = \|\operatorname{div}(\boldsymbol{u} - \boldsymbol{\Pi}_h \boldsymbol{u})\|_0^2 + \|\operatorname{div}(\boldsymbol{\Pi}_h \boldsymbol{u} - \boldsymbol{u}_h)\|_0^2.$$

Thus, we can conclude from (7.3) that

$$\begin{aligned} \|\boldsymbol{u} - \boldsymbol{u}_h\|_a^2 + \delta^{-2}\|(I - P_h)\operatorname{div} \boldsymbol{u}\|_0^2 + \delta^{-2}\|\operatorname{div}(\boldsymbol{\Pi}_h \boldsymbol{u} - \boldsymbol{u}_h)\|_0^2 \\ \leq c(h^2 + \varepsilon h)^2 \|\boldsymbol{u}\|_2^2 + \delta^{-2}\|(I - P_h)\operatorname{div} \boldsymbol{u}\|_0^2. \end{aligned}$$

We therefore have established the following convergence result.

Theorem 7.1. *If* $\boldsymbol{u} \in \boldsymbol{H}^2 \cap \boldsymbol{H}_0^1$ *then*

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|_0 + \varepsilon \|\operatorname{rot}(\boldsymbol{u} - \boldsymbol{u}_h)\|_0 + \delta^{-1}\|\operatorname{div}(\boldsymbol{\Pi}_h \boldsymbol{u} - \boldsymbol{u}_h)\|_0 \leq c(h^2 + \varepsilon h)\|\boldsymbol{u}\|_2.$$

Here $c > 0$ *is a constant independent of* ε, δ *and* h.

Note that from this result we can conclude that if ε and h are fixed, and δ approach zero, then $\operatorname{div} \boldsymbol{u}_h$ converges in L^2 to $P_h \operatorname{div} \boldsymbol{u}$. Furthermore, the divergence of the error can be controlled by this estimate since

$$\begin{aligned} \|\operatorname{div}(\boldsymbol{u} - \boldsymbol{u}_h)\|_0 &\leq \|(I - P_h)\operatorname{div} \boldsymbol{u}\|_0 + \|\operatorname{div}(\boldsymbol{\Pi}_h \boldsymbol{u} - \boldsymbol{u}_h)\|_0 \\ &\leq ch\|\operatorname{div} \boldsymbol{u}\|_1 + c\delta(\varepsilon^2 + h\varepsilon)\|\boldsymbol{u}\|_2. \end{aligned}$$

Of course, exactly as for the problem (1.1) we can argue that, in general cases, the norm $\|\boldsymbol{u}\|_2$ will not remain bounded as ε and δ approach zero. Hence, ideally we would like to generalize the results of §6 to the problem (7.1). However, this discussion is outside the scope of this paper.

Acknowledgment. The authors are grateful to Professors D.N Arnold, R.S. Falk and Z. Cai for many useful discussions.

References

[1] D.N. Arnold, F. Brezzi and M. Fortin, A stable finite element method for the Stokes equations, Calcolo 21 (1984), pp. 337–344.

[2] D.N. Arnold, L.R. Scott and M. Vogelius, Regular inversion of the divergence operator with Dirichlet boundary conditions on a polygon, Ann. Scuola Norm. Sup. Pisa Cl. Sci.–Serie IV **XV** (1988), pp. 169–192.

[3] J.H. Bramble and J.E. Pasciak, Iterative techniques for time dependent Stokes problem, Comput. Math. Appl. 33 (1997), pp. 13–30.

[4] J. Bergh and J. Löfström, *Interpolation spaces*, Springer Verlag, 1976.

[5] S.C. Brenner and L.R. Scott, *The mathematical theory of finite element methods*, Springer Verlag, 1994.

[6] F. Brezzi, On the existence, uniqueness and approximation of saddle–point problems arising from Lagrangian multipliers, RAIRO Anal. Numér. 8 (1974), pp. 129–151.

[7] F. Brezzi, J. Douglas and L.D. Marini, Two families of mixed finite elements for second order elliptic problems, Numer. Math. 47 (1985), pp. 217–235.

[8] F. Brezzi and M. Fortin, *Mixed and hybrid finite element methods*, Springer Verlag, 1991.

[9] P.G. Ciarlet, *The finite element method for elliptic problems,* North–Holland Publishing Company, 1978.

[10] M. Crouzeix and P.A. Raviart, Conforming and non–conforming finite element methods for solving the stationary Stokes equations, RAIRO Anal. Numér. 7 (1973), pp. 33–76.

[11] V. Girault and P.-A. Raviart, *Finite element methods for Navier–Stokes equations*, Springer Verlag 1986.

[12] P. Grisvard, *Elliptic problems on nonsmooth domains,* Monographs and studies in mathematics vol. 24, Pitman Publishing Inc., 1985.

[13] E. Haug, T. Rusten and H. Thevik, A mathematical model for macrosegregationformation in binary alloy solidification, in *Numerical methods and software tools in industial mathematics,* M. Dæhlen and A. Tveito eds., Birkhäuser, 1997.

[14] T.K. Nilssen, X-C Tai and R. Winther, A robust nonconforming H^2–element, Math. Comp. 70 (2000), pp. 489–505.

[15] P.A. Raviart and J.M. Thomas, A mixed finite element method for second order elliptic problems, *Mathematical aspects of finite element methods*, Lecture Notes in Mathematics 606, Springer–Verlag 1977.
[16] S. Whitaker, Flow in porous media I: A theoretical derivation of Darcy's law, Transport in porous media 1 (1986), pp. 3–25.
[17] O.C. Zienkiewicz and R.L. Taylor, The Finite Element Method, Butterworth-Heinemann, 2000.

Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway
E-mail address: kent-and@ifi.uio.no

Department of Mathematics, University of Bergen, Johannes Brunsgt. 12, 5007 Bergen, Norway
E-mail address: Xue-Cheng.Tai@mi.uib.no

Department of Informatics and Department of Mathematics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway
E-mail address: rwinther@ifi.uio.no

IV

Systems of PDEs and Block Preconditionering

K.-A. Mardal, J. Sundnes, H. P. Langtangen and A. Tveito

In Langtangen and Tveito (eds): *Advanced Topics in Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*, Lecture Notes in Computational Science and Engineering, Springer, 2003.

Systems of PDEs and Block Preconditioning

K.-A. Mardal[1,2], J. Sundnes[1,2], H. P. Langtangen[1,2], and A. Tveito[1,2]

[1] Simula Research Laboratory
[2] Department of Informatics, University of Oslo

Abstract. We consider several examples on systems of PDEs and how block preconditioners can be formulated. Implementation of block preconditioners in Diffpack is in particular explained. We emphasize object-oriented design, where a standard simulator is developed and debugged before being extended with efficient block preconditioners in a derived class. Optimal preconditioners are applied to the Stokes problem, the mixed formulation of the Poisson equation, and the Bidomain model for the electrical activity in the heart.

5.1 Introduction

In this chapter we will see how block preconditioners can be utilized in Diffpack for three different model problems. The chosen model problems are fundamental problems in engineering, medicine and science and they need to be solved in real-life situations, implying very large scale simulations. In such situations the algorithm for solving the matrix equation is of vital importance, and this motivates the use of highly efficient block preconditioners.

A model for the electrical activity in the heart is described in Chapter [14]. We will consider a simplified form of the equations, given by

$$\begin{aligned} \frac{\partial v}{\partial t} &= \nabla \cdot (\sigma_i \nabla v) + \nabla \cdot (\sigma_i \nabla u_e), \ x \in \Omega, \ t > 0, \\ \nabla \cdot (\sigma_i \nabla v) + \nabla \cdot ((\sigma_i + \sigma_e) \nabla u_e) &= 0, \ x \in \Omega, \ t > 0, \end{aligned}$$

where v is the transmembrane potential, u_e is the extracellular potential, and σ_i and σ_e are the intra- and extracellular conductivities, respectively. The two other problems, described in [10], are the Stokes problem and the mixed Poisson problem. It is well known how efficient preconditioners may be constructed for these problems (see e.g. [13,1]). The Stokes problem, modeling creeping Newtonian flow, reads

$$\begin{aligned} -\mu \Delta \boldsymbol{v} + \nabla p &= \boldsymbol{f} \text{ in } \Omega \ \ \text{(equation of motion)}, \\ \nabla \cdot \boldsymbol{v} &= 0 \ \text{ in } \Omega \ \ \text{(mass conservation)}, \end{aligned}$$

where $\boldsymbol{v}$ is the velocity, p is the pressure, μ is the viscosity, and $\boldsymbol{f}$ denotes the body forces. The last example is the mixed Poisson problem,

$$\begin{aligned} \boldsymbol{v} + \lambda \nabla p &= 0 \text{ in } \Omega \ \ \text{(Darcy's law)}, \\ \nabla \cdot \boldsymbol{v} &= g \text{ in } \Omega \ \ \text{(mass conservation)}, \end{aligned}$$

where $\boldsymbol{v}$ is the velocity, p is the pressure, λ is a mobility parameter, and g is a source term. These equations are discretized by the finite element method. In the Stokes problem and the mixed Poisson problem we use mixed elements.

With a suitable numbering of the unknowns the discretization of all the equations given above may be written as

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{c} \end{bmatrix} . \tag{5.1}$$

It is this particular block structure that motivates our preconditioners. We want to utilize the structure of the problems to construct efficient preconditioners. In fact, for all our model problems it is possible to obtain optimal preconditioners by applying simple algebraic operator splitting techniques to (5.1). An optimal preconditioner leads to a number of iterations that is bounded independent of the number of unknowns. This implies that the amount of computational work is proportional to the number of unknowns. This is (order) optimal.

The remainder of this paper is organized as follows. Section 1 introduces block preconditioners from an algebraic point of view and describes some software tools in Diffpack. In Section 2 the Bidomain equations are considered, and an optimal preconditioner is developed together with appropriate software tools. In Section 3 we extend the simulators presented in [10] to utilize block solvers.

5.2 Block Preconditioners in General

In this section we will try to motivate block preconditioners from an algebraic point of view. We first consider the Jacobi method on block form, which is often used to solve coupled systems of PDEs. We then discuss some basic properties of the Jacobi method and motivate why and how it can be used as a preconditioner. We discuss the Jacobi method instead of the slightly faster Gauss-Seidel, because Jacobi is symmetric and can therefore be used as a preconditioner for e.g. the Conjugate-Gradient method. Then we briefly consider Conjugate-Gradient-like methods and see how they can be combined with these block preconditioners. The two last sections deal with software tools to utilize these preconditioners in existing simulators. Readers interested in the mathematical details of iterative methods, e.g., the Conjugate-Gradient method, multigridmethods and classical iterations, are refereed to Hackbusch [8]. Krylov solvers in general, for non-symmetric and indefinite matrices, are dealt with in [3].

5.2.1 Operator Splitting Techniques

Operator slitting techniques are widely used for coupled systems of PDEs, and there are many different techniques that are used for different PDE

problems. Often the operator splitting is used in time, resulting in a splitting on the PDE level. One of the main advantages with these techniques is that a complicated system of coupled PDEs is reduced to simpler equations like the Poisson equation, convection-diffusion equations etc. For such equations a fast solver like multigrid can be employed. Additionally well tested software for such problems can be reused.

However, here we will consider techniques on the linear algebra level, but we should bear in mind that the fast solvers should and will also be used in this approach. The reader interested in the mathematical details is refereed to [8]. Given an algebraic system on the form,

$$\mathcal{A}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{C} & \boldsymbol{D}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix} = \begin{bmatrix}\boldsymbol{b}\\ \boldsymbol{c}\end{bmatrix}, \tag{5.2}$$

the simplest operator splitting technique is the Jacobi method,

$$\boldsymbol{x}^{k+1} = \boldsymbol{A}^{-1}(\boldsymbol{b} - \boldsymbol{B}\boldsymbol{y}^k), \tag{5.3}$$
$$\boldsymbol{y}^{k+1} = \boldsymbol{D}^{-1}(\boldsymbol{c} - \boldsymbol{C}\boldsymbol{x}^k)\,. \tag{5.4}$$

Let $\mathcal{D}$ be the block-wise diagonal of $\mathcal{A}$,

$$\mathcal{D} = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{D}\end{bmatrix}. \tag{5.5}$$

The convergence of Jacobi is determined by the spectral radius of $\mathcal{I} - \mathcal{D}^{-1}\mathcal{A}$, $\rho(\mathcal{I} - \mathcal{D}^{-1}\mathcal{A})$, where $\mathcal{I} = diag(\boldsymbol{I}, \boldsymbol{I})$. A necessary condition for convergence is that

$$\rho(\mathcal{I} - \mathcal{D}^{-1}\mathcal{A}) \leq \delta < 1\,. \tag{5.6}$$

Hence, the standard results of point-wise Jacobi apply also to the block version of Jacobi's method. For coupled systems of PDEs it is often not easy to verify (5.6) a priori. The convergence/divergence determined by δ, is usually found experimentally, and quite often Jacobi diverges. Jacobi is also known to be a rather slow iterative method, hence it might be necessary to iterate quite a number of times. A natural question to ask is how accurate $\boldsymbol{A}^{-1}$ and $\boldsymbol{D}^{-1}$ need to be. Can we use cheap approximations of these matrices, $\tilde{\boldsymbol{A}}^{-1}$ and $\tilde{\boldsymbol{D}}^{-1}$, and speed up the simulation time? A naive inexact version of block Jacobi (5.3)-(5.4) reads,

$$\boldsymbol{x}^{k+1} = \tilde{\boldsymbol{A}}^{-1}(\boldsymbol{b} - \boldsymbol{B}\boldsymbol{y}^k), \tag{5.7}$$
$$\boldsymbol{y}^{k+1} = \tilde{\boldsymbol{D}}^{-1}(\boldsymbol{c} - \boldsymbol{C}\boldsymbol{x}^k)\,. \tag{5.8}$$

However, we see that this approach is dangerous, since we then solve a perturbed system

$$\tilde{\mathcal{A}}\begin{bmatrix}\tilde{\boldsymbol{x}}\\ \tilde{\boldsymbol{y}}\end{bmatrix} = \begin{bmatrix}\tilde{\boldsymbol{A}} & \boldsymbol{B}\\ \boldsymbol{C} & \tilde{\boldsymbol{D}}\end{bmatrix}\begin{bmatrix}\tilde{\boldsymbol{x}}\\ \tilde{\boldsymbol{y}}\end{bmatrix} = \begin{bmatrix}\boldsymbol{b}\\ \boldsymbol{c}\end{bmatrix}, \tag{5.9}$$

Instead one should use the consistent version of the inexact Jacobi[1]

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \tilde{\boldsymbol{A}}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{B}\boldsymbol{y}^k), \tag{5.10}$$

$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \tilde{\boldsymbol{D}}^{-1}(\boldsymbol{c} - \boldsymbol{C}\boldsymbol{x}^k - \boldsymbol{D}\boldsymbol{y}^k)\,. \tag{5.11}$$

The convergence will then be determined by

$$\rho(\mathcal{I} - \tilde{\mathcal{D}}^{-1}\mathcal{A})\,. \tag{5.12}$$

Note that this quantity is even harder to estimate than δ. The method (5.10)-(5.11) can also be seen as a Richardson method preconditioned with $\tilde{\mathcal{D}}^{-1}$. Introducing a damping parameter τ,

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \tau\tilde{\boldsymbol{A}}^{-1}\boldsymbol{r}_x^k, \tag{5.13}$$

$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \tau\tilde{\boldsymbol{D}}^{-1}\boldsymbol{r}_y^k\,. \tag{5.14}$$

where the residual $[\boldsymbol{r}_x^{k+1}, \boldsymbol{r}_y^{k+1}]$ is defined by

$$\boldsymbol{r}_x^k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{B}\boldsymbol{y}^k, \tag{5.15}$$

$$\boldsymbol{r}_y^k = \boldsymbol{c} - \boldsymbol{C}\boldsymbol{x}^k - \boldsymbol{D}\boldsymbol{y}^k \tag{5.16}$$

we can always make the iteration convergent by adjusting[2] τ such that

$$\rho(\mathcal{I} - \tau\tilde{\mathcal{D}}^{-1}\mathcal{A}) < 1\,.$$

However, τ must then usually be found and tuned experimentally for optimal performance. A too small τ will reduce the efficiency, while a too large τ will lead to divergence.

Although we have here focused on the block Jacobi method, a large number of other operator splitting methods exist. Variants that are similar to block Jacobi are the block versions of Gauss-Seidel, SOR, and SSOR, which may all be obtained by modifying the preconditioner. On this 2×2 block system we replace $\mathcal{D}^{-1}$ with the following $\mathcal{B}_{GS}^{-1}$ to obtain the block Gauss-Seidel method,

$$\mathcal{B}_{GS}^{-1} = \begin{bmatrix} \boldsymbol{A}^{-1} & \boldsymbol{0} \\ -\boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{A}^{-1} & \boldsymbol{D}^{-1} \end{bmatrix} \tag{5.17}$$

The following discussion will not only concern with the block-diagonal preconditioner leading to the block Jacobi method, but with preconditioners in general. Hence, the notation $\tilde{\mathcal{D}}^{-1}$ is replaced with $\mathcal{B}$. For all the algorithms the condition number $\kappa(\mathcal{B}\mathcal{A})$ determines the efficiency of the algorithm and whether it converges or not. In practice the estimates of these parameters are

[1] This is what Hackbusch [8] calls the second normal form of a consistent linear iteration. Inexact Jacobi is used to generate $\tilde{\mathcal{D}}^{-1}$.

[2] It can be done by setting $\tau = \frac{1}{\|\tilde{\mathcal{D}}^{-1}\mathcal{A}\|}$.

almost always computed. For the Stokes problem the Uzawa algorithm or the more general Pressure Schur Complement framework (see [16]) is often used. We will see that similar reasoning can be used here. In the next section we will consider more robust methods, that in principle always are convergent. Still, the convergence rate depends depends strongly on $\kappa(\mathcal{BA})$.

5.2.2 Conjugate Gradient-Like Methods

Although there exists a large variety of iterative solvers that may utilize preconditioners based on operator splitting, we will focus on a family of such methods that is particularly interesting. This is the group of Conjugate Gradient-like methods or Krylov Subspace methods. There has been developed a rich set of Krylov solvers for different purposes, many of which are implemented in Diffpack. Some of the most commonly used are `ConjGrad` for symmetric and positive definite (SPD) systems, `Symmlq` and `SymMinRes` for symmetric and indefinite problems, and general purpose methods like `Orthomin`, `GMRES`, `BiCGStab`. The reader is referred to [3] for a general discussion of Krylov solvers and iterative solvers in general, and [4,9] for a presentation of the Krylov solvers implemented in Diffpack.

The Krylov solvers do not need a τ to be chosen to be convergent. However, the efficiency of an appropriate Krylov solver is to a large extent dependent on the condition number of the coefficient matrix $\mathcal{A}$, $\kappa(\mathcal{A})$. We have e.g. that the number of iterations is proportional to $\sqrt{\kappa(\mathcal{A})}$ for the Conjugate Gradient method[3]. For standard second order elliptic problems in 2D the condition number is typically $\mathcal{O}(h^{-2})$, where h is the grid size parameter, and the performance of the Krylov solver will therefore deteriorate as the grid is refined. For sparse matrices arising from the discretization of PDEs, the memory required to store the matrix and vectors is $\mathcal{O}(n)$, where n is the number of unknowns in the linear system. The basic operations in Krylov solvers are matrix-vector products, vector additions and inner products, which are all $\mathcal{O}(n)$ operations. The overall performance is therefore $\mathcal{O}(n^{3/2})$, since the number of iterations is $\sqrt{h^{-2}} \sim n^{1/2}$.

In matrices arising from discretizations of PDEs the condition number increases as the resolution of the grid increases. We are not satisfied with being able to solve the problem on one particular grid. In four years when we have a four times faster computer with four times as much memory[4], we

[3] some assumptions must be made, see Section 5.3.3

[4] A popular version of Moore's law predicts that the processor speed double at least every other year. However, it seems to be slowing down, a doubling is more likely to happen every two year. It is also becoming apparent that the speed of the CPU is not the only factor that determine the total CPU time. The latency of the RAM is also very important. In fact, for the problems we consider here, which are very memory intensive, the CPU usually spend 90% of the time waiting for lookups in RAM, see [5].

want a four times as accurate solution in the same amount of time. Hence, if we want efficient solution methods we need to consider a family of problems,

$$\mathcal{A}_h \begin{bmatrix} \boldsymbol{x}_h \\ \boldsymbol{y}_h \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_h & \boldsymbol{B}_h \\ \boldsymbol{C}_h & \boldsymbol{D}_h \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_h \\ \boldsymbol{y}_h \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_h \\ \boldsymbol{c}_h \end{bmatrix}, \tag{5.18}$$

where h is the most important parameter for the condition number. Our ultimate goal is to construct methods that will solve the problem in $\mathcal{O}(n)$ operations, independently of the grid size h. As mentioned above, this is commonly called an *optimal method*. We are considering iterative methods, hence the demands on a optimal solver can be summarized as follows,

1. The number of operations in each iteration should be proportional to the number of unknowns.
2. The memory requirement should be proportional to the number of unknowns.
3. The number of iterations needed to reach convergence should be bounded independently of the mesh.

As mentioned earlier the convergence of a Krylov solver for symmetric systems can be estimated in terms of the condition number of $\mathcal{A}_h$ and the point 3 is therefore not satisfied. A common way to speed up the Krylov solver is therefore to use a preconditioner, essentially meaning that we multiply the system by a matrix $\mathcal{B}_h$ to obtain another system with the same solution,

$$\mathcal{B}_h \mathcal{A}_h \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \mathcal{B}_h \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{c} \end{bmatrix}. \tag{5.19}$$

The preconditioner $\mathcal{B}_h$ should be a well designed operator, because the efficiency of the Krylov solver then depends on $\kappa(\mathcal{B}_h \mathcal{A}_h)$. The preconditioner $\mathcal{B}_h$ does not need to be formed as a matrix explicitly, it can also be an algorithm like, e.g., multigrid. The only action that is needed is its evaluation on a vector, $v_h = \mathcal{B}_h u_h$. The demands given above can then be stated as properties of the preconditioner $\mathcal{B}_h$:

1. The application of $\mathcal{B}_h$ to a vector should require $\mathcal{O}(n)$ operations.
2. The memory required to store $\mathcal{B}_h$ should be proportional to the number of unknowns.
3. $\kappa(\mathcal{B}_h \mathcal{A})$ should be "small" and bounded independently of h.

Point 3, in the case of symmetric matrices, is often stated as

$$\kappa(\mathcal{B}_h \mathcal{A}_h) = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|} \leq C, \tag{5.20}$$

where λ_i are the eigenvalues and C is a constant that is independently of the discretization parameter h. We will then say that $\mathcal{B}_h \mathcal{A}_h \sim \boldsymbol{I}$. Notice that

the above condition (5.20) differ slightly from the spectral equivalence that is usually wanted for elliptic operators. It is needed for the indefinite systems in Section 5.4.

The "operator splitting" technique in (5.10)-(5.11) can be reused directly. Using the matrix representation in (5.5) with inexact subsolvers, the preconditioner in (5.19) reads

$$\mathcal{B}_h = \begin{bmatrix} \tilde{\boldsymbol{A}}^{-1} & 0 \\ 0 & \tilde{\boldsymbol{D}}^{-1} \end{bmatrix}. \tag{5.21}$$

This is the simplest form of the block preconditioners. Nevertheless it will be the focus of this paper.

5.2.3 Software Tools for Block-Matrices and Numbering Strategies

Equation (5.1) is just an ordinary matrix equation,

$$\mathcal{A}\boldsymbol{u} = \boldsymbol{b},$$

with a particular numbering. That is, we have two unknowns $\boldsymbol{x}$ and $\boldsymbol{y}$ numbered block wise giving the solution vector,

$$\boldsymbol{u} = [x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m].$$

In general $\boldsymbol{x}$ and $\boldsymbol{y}$ define different fields which may be defined on different grids. One example is the Stokes problem which requires mixed elements, e.g. bilinear elements for $\boldsymbol{y}$ and biquadratic for $\boldsymbol{x}$. Diffpack has a general numbering strategy capable of numbering several different fields simultaneously. However, a straightforward use of this numbering (as was explained in Chapter [10]) results in a matrix where the block structure is lost. We want to keep the system on block form, implying that the matrix $\boldsymbol{B}$ in (5.1) needs a numbering consistent with $\boldsymbol{x}$ column-wise, while the row numbering must be the same as in $\boldsymbol{y}$. Hence, $\boldsymbol{B}$ is a $n \times m$ matrix and n may be unequal to m. This is the case for the Stokes and the mixed Poisson problem. Diffpack has several tools for assembling and solving linear system on block-structured form. The coupling of unknown fields (`FieldFE`) and their degree of freedom in the linear system (`LinEqSystem`) is handled by a `DegFreeFE` object, and in case of block-structured systems, we use two `DegFreeFE` objects for each block matrix. `DegFreeFE` has two numbering strategies, the special and the general, we need both. In the first simulators for the Bidomain model it is sufficient to use isoparametric elements and the special numbering can be used. In Section 5.4 we need to use mixed elements, and we must then use the general numbering.

Here is a list of the tools used to facilitate the handling of block-structured systems.

- Each matrix in $\mathcal{A}$ needs two `DegFreeFE` objects to handle the numbering. One is used for the column-wise numbering and one for the row-wise.
- Class `ElmMatVecs` holds a matrix of `ElmMatVec` objects, where each `ElmMatVec` object contains a block matrix and a block vector at the element level.
- Class `SystemCollector` administers the computation of the element block matrices and vectors, as well as the modification of the matrices and vectors due to essential (Dirichlet) boundary conditions. The class contains handles to an `ElmMatVecs` object, all the `DegFreeFE` objects, and functors [9] to hold the integrands for each block matrix and vector. The assembly of element contributions in `LinEqAdmFE::assemble` relies on a `SystemCollector` object.

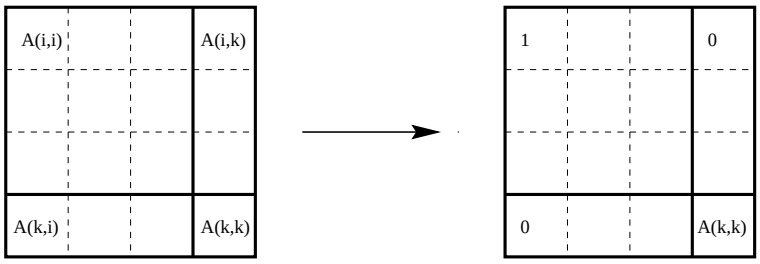

Fig. 5.1. Enforcing boundary condition subject to node i.

Linear systems in Diffpack support block structures. That is, the classes `LinEqAdm`, `LinEqSolver` and `LinEqSystem` and their subclasses always work on `LinEqMatrix` and `LinEqVector` objects. `LinEqMatrix` is a `MatSimplest` (usually 1×1) of pointers (handles) to `Matrix` objects. The `LinEqVector` has a similar structure. This means that the Diffpack Krylov solvers can be used on the mixed system without modifications.

`LinEqSystemPrec` is a subclass of `LinEqSystem` designed for preconditioned linear systems. To handle block preconditioners, this class is extended by the subclass `LinEqSystemBlockPrec`. It has the same interface as `LinEqSystemPrec` plus some more functions associated with the block preconditioner. We refer to the manual page for the full interface. The use of `LinEqSystemBlockPrec` is exemplified in the source code in Chapter 5.3.4.

```
//Make a 2x2 matrix of integrand-functors
integrands.redim(2,2);
integrands(1,1).rebind( new Integrand11(this));
integrands(2,1).rebind( new Integrand21(this));
integrands(1,2).rebind( new Integrand12(this));
integrands(2,2).rebind( new Integrand22(this));

//Make ElmMatVecCalcStd functor to be used from makeSystem
```

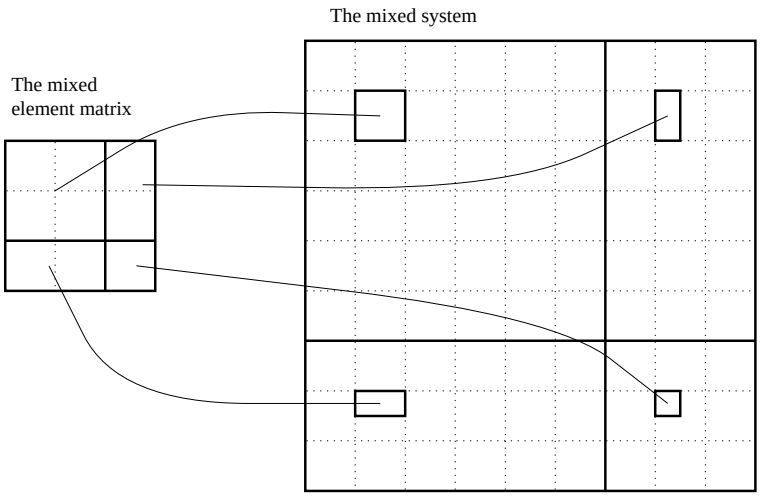

Fig. 5.2. The assembly process of the mixed system.

```
calcelm.rebind (new ElmMatVecCalculator());


//Block matrix of ElmMatVec's
elmats.rebind(new ElmMatVecs());
//SystemCollectorFEM handles the assembly
//the block element matrices into the
//block matrix
elmsys.rebind(new SystemCollectorFEM());

//redim and attach datastructure:
elmsys->redim(2,2);
for (int i=1; i<= 2; i++)
  for (int j=1; j<= 2; j++)
    if (integrands(i,j).ok())
      elmsys->attach(integrands(i,j)(),i,j);

elmsys->attach(*calcelm);

elmsys->attach(*u_dof,1,true);
elmsys->attach(*u_dof,1,false);

elmsys->attach(*v_dof,2,true);
elmsys->attach(*v_dof,2,false);

//lineq is initialized from the DegFreeFE's attached
//to elmsys
lineq->initAssemble(*elmsys);

//attach the ElmMatVec's
elmats->attach(*elmsys);
elmsys->attach(*elmats);
```

Assembly of the right hand side Each of the `integrandMx` functors could have done calculations like

```
elmat.b(i) += ...
```

on the right hand side, as is common in a Diffpack `integrands` function. This would lead to a distributed calculation of the right hand side in several `IntegrandCalc` objects. Instead we have chosen to use the diagonal `IntegrandCalc` objects to handle the computation of the element vector. This gives cleaner and more efficient code. Hence, only

```
integrands(1,1)
integrands(2,2)
```

will be used in the computation of the right hand side.

5.2.4 Software Tools for Block Structured Preconditioners

As indicated above we will in this paper focus on block diagonal preconditioners, which may be written as

$$\mathcal{B} = \begin{bmatrix} \boldsymbol{M} & 0 \\ 0 & \boldsymbol{N} \end{bmatrix}, \tag{5.22}$$

where $\boldsymbol{M}$ and $\boldsymbol{N}$ are operators that are constructed in some way reflecting the structure of the linear system matrix $\mathcal{A}$. The software is not restricted to 2×2 block systems, any block system can be used. In this chapter we will however stick to 2×2 systems, for simplicity. In Diffpack such preconditioners can be made by defining the menu interface `MenuSystem`,

```
menu.setCommandPrefix("M prec");
Precond_prm ::defineStatic(menu, level+1);
menu.unsetCommandPrefix();
menu.setCommandPrefix("N prec");
Precond_prm::defineStatic(menu, level+1);
menu.unsetCommandPrefix()
```

scan the parameters,

```
menu.setCommandPrefix("M prec");
M_prec_prm.rebind( Precond_prm::construct());
M_prec_prm->scan(menu);
menu.unsetCommandPrefix ();
menu.setCommandPrefix("N prec");
N_prec_prm.rebind( Precond_prm::construct());
N_prec_prm->scan(menu);
menu.unsetCommandPrefix ();
```

and attach the preconditioners to the linear system, represented by the class `LinEqSystemBlockPrec`.

```
LinEqSystemBlockPrec& lins = CAST_REF(lineq->getLinEqSystem(),
     LinEqSystemBlockPrec);
lins.attach(*M_prec_prm,1);
lins.attach(*N_prec_prm,2);
```

`LinEqSystemBlockPrec` supply the routines

```
virtual void applyLeftBlockDiagPrec
(
  const LinEqVector& c,
        LinEqVector& d,
        TransposeMode tpmode = NOT_TRANSPOSED
);

// apply right block preconditioner Cright, d=Cright*c
virtual void applyRightBlockDiagPrec
(
  const LinEqVector& c,
        LinEqVector& d,
        TransposeMode tpmode = NOT_TRANSPOSED
);
```

which are used by the Krylov solvers. The user should make sure that the matrix, the vector and the corresponding preconditioner are all constructed using the same `DegFreeFE`-object. This will ensure that the numbering of all components are consistent.

This completes our general discussion of block preconditioners and the corresponding software tools. In the remainder of the paper we will demonstrate how the proposed solution method may be applied to specific problems.

5.3 The Bidomain Equations

As a first introduction to problems that lead to block structured linear systems, we consider a set of partial differential equations modeling the electrical activity in the heart. Solving these equations enables us to simulate the propagation of the electrical field that causes the cardiac cells to contract and thus initiates each heartbeat. Measurements of this electrical field on the surface of the body are known as electrocardiograms or ECG measurements, and are an important tool for identifying various pathological conditions. Performing realistic computer simulations may increase our understanding of this process, and may thus be a valuable tool for research and educational purposes.

5.3.1 The Mathematical Model

To obtain a set of partial differential equations describing the propagation of the electrical field, we introduce the quantities u_i and u_e, denoting the intracellular and extracellular potential, respectively. To avoid treating the

complex cellular structure of the tissue explicitly, we consider the heart muscle to be a continuum consisting of both intracellular and extracellular space. The potentials u_i and u_e are then volume averaged quantities defined at every point in the domain. To easily relate the electrical potentials to ionic currents crossing the cell membranes, which are extremely important for the propagation of the potentials, we introduce the transmembrane potential v, defined as $v = u_i - u_e$. If we further denote the intra- and extracellular conductivities by σ_i and σ_e respectively, the dynamics of the electrical potentials are governed by the following equations,

$$\begin{aligned} \chi C_m \tfrac{\partial v}{\partial t} + \chi I_{\text{ion}}(v, s) &= \nabla \cdot (\sigma_i \nabla v) + \nabla \cdot (\sigma_i \nabla u_e), && x \in \text{Heart}(\Omega), \\ \nabla \cdot ((\sigma_i + \sigma_e) \nabla u_e) &= -\nabla \cdot (M_i \nabla v), && x \in \text{Heart}(\Omega)\,. \end{aligned}$$

This is the Bidomain model, introduced by Geselowitz [6] in the late 70s. The term I_{ion} is the transmembrane ionic current, C_m is the capacitance of the cell membrane and χ is a parameter relating the area of the cell membrane to the tissue volume. A detailed derivation of these equations is given in Chapter [14]. The vector s in the ionic current term I_{ion} is a state vector characterizing the conductivity properties of the cell membrane. Realistic models of cardiac cells require this state vector to be determined from a complex set of ordinary differential equations. The solution of these ODEs contributes significantly to the complexity of solving the Bidomain equations numerically. Simulations based on the complete equations are presented in Chapter [14]. In the present section we want the Bidomain problem to serve as a simple model problem to introduce the concepts of block systems and block preconditioning. To obtain a problem that is more suitable for this purpose we disregard the ionic current term I_{ion} and assume that $\chi C_m = 1$. We then get the simplified problem

$$\frac{\partial v}{\partial t} = \nabla \cdot (\sigma_i \nabla v) + \nabla \cdot (\sigma_i \nabla u_e), \;\; x \in \Omega, \tag{5.23}$$

$$\nabla \cdot (\sigma_i \nabla v) + \nabla \cdot ((\sigma_i + \sigma_e) \nabla u_e) = 0, \;\; x \in \Omega\,. \tag{5.24}$$

For the present study, we assume that the heart muscle is immersed in a non-conductive media, so that no current leaves the heart tissue. The current is assumed to be given by Ohms law, so this gives

$$n \cdot (\sigma_i \nabla u_i) = 0$$

and

$$n \cdot (\sigma_e \nabla u_e) = 0$$

on the surface of the heart, which defines the boundary conditions for (5.23)-(5.24). By utilizing that $u_i = v + u_e$ we may express these conditions in terms of our primary variables v and u_e. Combining this with equations (5.23) and

(5.24) gives the following initial-boundary value problem

$$\frac{\partial v}{\partial t} = \nabla \cdot (\sigma_i \nabla v) + \nabla \cdot (\sigma_i \nabla u_e), \;\; x \in \Omega, \;\; t > 0 \tag{5.25}$$

$$\nabla \cdot (\sigma_i \nabla v) + \nabla \cdot ((\sigma_i + \sigma_e) \nabla u_e) = 0, \;\; x \in \Omega, \;\; t > 0 \tag{5.26}$$

$$n \cdot (\sigma_i \nabla v + \sigma_i \nabla u_e) = 0, \;\; x \in \partial\Omega, \;\; t > 0 \tag{5.27}$$

$$n \cdot (\sigma_e \nabla u_e) = 0, \;\; x \in \partial\Omega, \;\; t > 0 \tag{5.28}$$

$$v(x, 0) = v_0(x), \;\; x \in \Omega, \;\; t = 0, \tag{5.29}$$

where v_0 is a given initial distribution for the transmembrane potential.

5.3.2 Numerical Method

Time Discretization We let Δt be the (constant) time step and denote by v^l and u^l approximations to v an u_e, respectively, at time step t_l. More precisely we have

$$v^l(x) \approx v(x, l\Delta t),$$
$$u^l(x) \approx u_e(x, l\Delta t)\,.$$

A time discretization of equation (5.25) may be obtained from a simple implicit Euler scheme, leading to the following equations to be solved at each time step,

$$\frac{v^l - v^{l-1}}{\Delta t} = \nabla \cdot (\sigma_i \nabla v^l) + \nabla \cdot (\sigma_i \nabla u^l), \;\; x \in \Omega, \tag{5.30}$$

$$\nabla \cdot (\sigma_i \nabla v^l) \nabla \cdot ((\sigma_i + \sigma_e) \nabla u^l) = 0, \;\; x \in \Omega\,. \tag{5.31}$$

To simplify the notation we introduce the inner products

$$a_i(v, \phi) = \Delta t \int_\Omega \sigma_i \nabla v \cdot \nabla \phi dx, \tag{5.32}$$

$$a_{i+e}(v, \phi) = \Delta t \int_\Omega (\sigma_i + \sigma_e) \nabla v \cdot \nabla \phi dx\,. \tag{5.33}$$

If we now define a suitable function space $V(\Omega)$, a weak formulation of equations (5.30)-(5.31) is to find $v^l, u^l \in V(\Omega)$ which satisfies

$$(v^l, \phi) + a_i(v^l, \phi) + a_i(u^l, \phi) = (v^{l-1}, \phi), \quad \forall \phi \in V(\Omega), \tag{5.34}$$

$$a_i(v^l, \phi) + a_{i+e}(u^l, \phi) = 0, \qquad \forall \phi \in V(\Omega), \tag{5.35}$$

where $(\cdot, \cdot)$ denotes the L^2 inner product. All boundary integral terms in the weak equations vanish because of the homogeneous boundary conditions (5.27)-(5.28).

We now introduce a discrete subspace $V_h(\Omega)$ of $V(\Omega)$, spanned by basis functions $\phi_i(x), i = 1, \ldots, n$. We approximate v^l and u^l by

$$v^l(x) \approx \sum_{j=1}^{n} \phi_j(x) v_j, \tag{5.36}$$

$$u^l(x) \approx \sum_{j=1}^{n} \phi_j(x) u_j, \tag{5.37}$$

where $\phi_i(x), i = 1, \ldots, n$ is a suitable set of basis functions defined in Ω. For the application of the finite element method, the basis functions are usually piecewise polynomials defined over a grid Ω_h. The refinement of the grid is characterized by the discretization parameter h. We have $n \sim h^{-d}$, where d is the number of physical space dimensions. Applying a standard Galerkin finite element discretization to the weak formulation (5.34)-(5.35), we obtain a system of linear equations given by

$$\sum_{j=1}^{l} (\phi_j, \phi_i) v_j + \sum_{j=1}^{l} a_i(\phi_j, \phi_i) v_j \tag{5.38}$$

$$+ \sum_{j=1}^{l} a_i(\phi_j, \phi_i) u_j = (v^{l-1}, \phi_i), \quad \text{for } i = 1, \ldots, n, \tag{5.39}$$

$$\sum_{j=1}^{l} a_i(\phi_j, \phi_i) v_j + \sum_{j=1}^{l} a_{i+e}(\phi_j, \phi_i) u_j = 0, \quad \text{for } i = 1, \ldots, n. \tag{5.40}$$

In matrix form, this system may be written as

$$\mathcal{A} \begin{bmatrix} v \\ u \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}, \tag{5.41}$$

where the matrix blocks are defined by

$$A_{ij} = (\phi_i, \phi_j) + a_i(\phi_i, \phi_j) \tag{5.42}$$

$$B_{ij} = a_i(\phi_i, \phi_j) \tag{5.43}$$

$$C_{ij} = a_{i+e}(\phi_i, \phi_j) \tag{5.44}$$

$$\alpha_i = (v^l, \phi). \tag{5.45}$$

5.3.3 Solution of the Linear System

It is of course possible to solve the linear system (5.41) without considering the block structure of the problem. Direct and iterative methods exist that will handle this problem fairly well. We may, however, improve the efficiency significantly by utilizing our knowledge of the system structure. The focus of this section is the construction of a highly efficient block preconditioner for this system.

A Preconditioner In this section we will consider the linear system to be solved at each time step as a stationary problem. We do not make any assumptions on the right hand side, but consider (5.41) with an arbitrary right hand side. The problem is given by

$$\mathcal{A}\begin{bmatrix}\boldsymbol{v}\\ \boldsymbol{u}\end{bmatrix} = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{B}^T & \boldsymbol{C}\end{bmatrix}\begin{bmatrix}\boldsymbol{v}\\ \boldsymbol{u}\end{bmatrix} = \begin{bmatrix}\boldsymbol{a}\\ \boldsymbol{b}\end{bmatrix}, \tag{5.46}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are arbitrary vectors. The reason for this is simply that the preconditioner needs to be able to work on any right hand side, since we do not have any a priori information about the behavior of the residual. Our suggested preconditioner for this linear system is

$$\mathcal{B}_h = \begin{bmatrix}(\boldsymbol{A}^{-1})\widehat{\ } & 0\\ 0 & (\boldsymbol{C}^{-1})\widehat{\ }\end{bmatrix},$$

where $(\cdot)\widehat{\ }$ denotes multigrid approximations to the respective inverse. Since the blocks $\boldsymbol{A}$ and $\boldsymbol{C}$ are similar to the discrete Laplace operator, it is well known from multigrid theory that such approximations can be made that are spectrally equivalent with the true inverse, see e.g. [17].

The Solver Since our linear system is symmetric and positive definite, we use the conjugate gradient method to solve the preconditioned system. If we denote the start vector by x^0 and the exact solution x, the initial error is given by $e^0 = x - x^0$. It can be shown that a reduction of the initial error with a factor $\epsilon \ll 1$ is achieved after at most

$$\frac{1}{2}\ln\frac{2}{\epsilon}\sqrt{\kappa} \tag{5.47}$$

iterations, see e.g. [3], where κ is the condition number of $\mathcal{BA}$. The condition number of $\mathcal{A}$ without preconditioning is $\kappa \sim h^{-2}$, and the number of iterations is thus proportional to h^{-1}. Preliminary analysis indicates that the preconditioner $\mathcal{B}$ introduced in the previous section is spectrally equivalent to the inverse of $\mathcal{A}$, and the condition number of the preconditioned system is hence bounded independently of h. According to (5.47), the number of iterations is thus independent of h, i.e. independent of the number of unknowns n.

5.3.4 A Simulator for the Bidomain Model

We first make a simulator for the Bidomain model without considering the block structure, primarily for debugging purposes. The complete source code for this simulator is located in `$NOR/doc/mixed/Heart`.

```
class Heart : public FEM
{
```

```
public:
  Handle(GridFE)        grid;    // underlying finite element grid
  Handle(FieldFE)       u;
  Handle(FieldFE)       u_prev;
  Handle(FieldFE)       v;
  Handle(DegFreeFE)     dof;
  Handle(TimePrm)       tip;
  Vec(real)             linsol;
  Handle(LinEqAdmFE)    lineq;
  Handle(SaveSimRes)    database;
  MatSimple(real)  sigma_i;
  MatSimple(real)  sigma_e;
  bool dump;


  Handle(FieldFE)       error;

  // used to partition scan into manageable parts:
  virtual void sigma_scan();
  void scanGrid();
  virtual void initFieldsEtc();

  virtual void timeLoop();
  virtual void solveAtThisTimeStep();
  virtual void setIC();

  // standard functions:
  virtual void fillEssBC ();
  virtual void fillEssBC4U (DegFreeFE& dof, int U);
  virtual void fillEssBC4V (DegFreeFE& dof, int V);

  virtual void calcElmMatVec
    (int e, ElmMatVec& elmat, FiniteElement& fe);

  virtual void integrands
    (ElmMatVec& elmat, const FiniteElement& fe);


public:

  Handle(FieldFunc)    usource;
  Handle(FieldFunc)    uic;
  Heart ();
 ~Heart ();

  virtual void adm (MenuSystem& menu);
  virtual void define (MenuSystem& menu, int level = MAIN);
  virtual void scan ();

  virtual void solveProblem ();
  virtual void saveResults ();
  virtual void resultReport ();
};
```

This simulator should be thought of as the main simulator. It is in charge of initialization of all the datastructure related to the physical properties of

the model problem. One should also use this simulator to validate the model and verify new simulators that utilize more efficient solution techniques, like the block and multigrid solvers that will be presented below. To ease the process of debugging the simulator, we also introduce a sub-class named `HeartAnalytic` which is intended purely for debugging purposes. The class solves the problem with very simple boundary- and initial conditions, and compares the numerical solution with analytical results. This simulator is also useful to find appropriate convergence criteria.

It is evident from the header file that the structure of this simulator class is not very different from the simplest Diffpack simulators introduced in [9]. The major difference is that we have two unknown fields, and one `DegFreeFE` object, based on the grid and the number of unknowns per node,

```
dof.rebind (new DegFreeFE (*grid, 2));
```

The fields of unknowns are made accordingly,

```
coll.rebind(new FieldsFE(*grid, "coll"));
u.rebind (coll()(1));
v.rebind (coll()(2));
```

Hence, the fields are made as vector fields, similar to the way they are made in the `Elasticity1` simulator described in [9]. When we use the special (or the general) numbering technique the block structure of the matrices will be lost. Usually one wants to avoid this block structure because it results in a very large bandwidth of the matrices and for factorization methods small bandwidths are desirable. We employ these numbering strategies within each block, such that each block has a very small bandwidth.

To be able to reuse the `fillEssBC` routine in the block simulators we split it into two functions:

```
fillEssBC4U(DegFreeFE& dof, int U)
fillEssBC4V(DegFreeFE& dof, int V)
```

The main idea is that a `DegFreeFE& dof` and a field number `U` is sufficient to fill the boundary conditions with `dof.fields2dof(· , U)`. This can be done in both the block simulators, multigrid and this simulator. Consult `$NOR/doc/mixed/Heart/HeartAnalytic.cpp` for the complete code.

The `HeartAnalytic` class is used to verify that the simulator produces correct results. A case with very simple geometry and initial conditions is considered, and the numerical results are compared to the analytical solution. The next step is to develop a simulator that utilizes the block structure in the solution algorithm. The purpose of this simulator is of course to produce the exact same results as the general `Heart` class, but to speed up the solution process by applying specialized block preconditioners. We first make a simple block structured simulator `HeartBlocks`, without any multigrid functionality. The process of debugging the simulators is substantially simplified by this incremental development, as we limit the amount of new functionality introduced in each step.

```
class HeartBlocks : public HeartAnalytic
{
public:
  Handle(Precond_prm)           prec_prm1;
  Handle(Precond_prm)           prec_prm2;
  Handle(FieldsFE)   u_coll;
  Handle(FieldsFE)              v_coll;
  Handle(DegFreeFE)  u_dof;
  Handle(DegFreeFE)             v_dof;
  Handle(LinEqVector)           linsol_block;
  Integrands_type               integrand_type;
  int nsd;

public:
  HeartBlocks();
 ~HeartBlocks();

  MatSimplest(Handle(IntegrandCalc)) integrands;
  Handle(ElmMatVecCalc)   calcelm;
  Handle(ElmMatVecs)     elmat;  // all blocks at the element level
  Handle(SystemCollector) elmsys;  // main utility for block adm.

  virtual void adm (MenuSystem& menu);
  virtual void timeLoop();
  virtual void define (MenuSystem& menu, int level = MAIN);
  virtual void scan ();
  void initBlocks();
  void initBlockDofs();
  virtual void solveAtThisTimeStep();
  virtual void solveProblem();
  virtual void resultReport ();
  virtual void fillEssBC();
  virtual void makePreconditioner();
  virtual void makePreconditioner4U();
  virtual void makePreconditioner4V();

  void makeBlockSystem
  (
   LinEqAdmFE&      lineq,
   SystemCollector& elmsys,
   FieldsFE&        coll,
   ElmMatVecs&     elmat,
   int              itg_man
  );

  void makeSystem
  (
    DegFreeFE& dof,
    FieldsFE& coll,
    IntegrandCalc& integrand,
    ElmMatVecCalc& emv,
    Matrix(NUMT)& A,
    int itgrule_man
  );

};
```

The `LinEqAdmFE` handle `lineq` introduced in the main simulator `Heart` is a general structure for linear systems, which is capable of handling block structured systems. The default version has only one block, but in this case we want to construct a 2×2 block system corresponding to the two primary unknowns. The call

```
lineq.rebind (new LinEqAdmFE(2,2));
```

in the `scan` function handles this basic initialization. This call constructs a `LinEqSystemBlockPrec` with a 2×2 block structure of the matrices, preconditioners and vectors. The preconditioners are attached as described in Section 5.2.3.

The main simulator class `Heart` was derived from the base class `FEM` which offers a virtual function `integrands`, which is used to make the linear system in `makeSystem`. This is the common way to construct the linear system in a Diffpack simulator. However, Diffpack also offers other possibilities, which are more suitable for handling block structured systems. Instead of redefining the provided `integrands` function, we may introduce the integrands as derived subclasses of `IntegrandCalc`, which offers the `integrands (ElmMatVec& elmat, const FiniteElement& fe)`, similar to `FEM`. The base class `FEM` then offers general `makeSystem` routines like,

```
virtual void makeSystem
  (DegFreeFE& dof, FieldsFE& coll, IntegrandCalc* integrand,
  ElmMatVecCalc& emv, Matrix(NUMT)& A),
```

which only uses the datastructures submitted in the function call. This is in fact the function we will use to make the algebraic preconditioners, as we will see later.

For a block system we need several integrand classes, all sub-classes of `IntegrandCalc`, which are typically organized in a 2-dimensional array.

```
MatSimplest(Handle(IntegrandCalc)) integrands;
```

The matrix is initialized according to the structure of the system, which in our case is a 2×2 system.

```
integrands.redim(2,2);
integrands(1,1).rebind( new Integrand11(this));
integrands(2,1).rebind( new Integrand21(this));
integrands(1,2).rebind( new Integrand12(this));
integrands(2,2).rebind( new Integrand22(this));
```

Each element in `integrands` is derived of `IntegrandCalc` and needs to define an `integrands` function. In addition they usually contain a pointer to a `Heart` object, submitted as `this`, to be able to fetch various data from the main simulator.

The four `integrands` functions will correspond to the four distinct parts of the `integrands` function implemented in the `Heart` class. The following one, corresponding to the block $\boldsymbol{C}$ in (5.41), is a typical function for Poisson type equations (compare with the `Poisson1` example).

```
void Integrand22::integrands
  (ElmMatVec& elmat, const FiniteElement& fe)
{

  real nabla;
  real detJxW = fe.detJxW();
  int nbf = fe.getNoBasisFunc();
  int nsd = fe.getNoSpaceDim();
  int i,j,k;
  real dt = data->tip->Delta(); //data is a Heart pointer
  for (i = 1; i <= nbf; i++) {
    for (j = 1; j <= nbf; j++) {
      nabla = 0;
      for (k = 1; k <= nsd; k++)
  nabla += (data->sigma_e(k,k)+data->sigma_i(k,k))
*fe.dN(i,k)*fe.dN(j,k);
      elmat.A(i,j) += dt*nabla*detJxW;
    }
    elmat.b(i) += 0.0;
  }
}
```

Since all the `calcElmMatVec` functions are identical in this application they are all represented as one `ElmMatVecCalculator` object. The class `ElmMatVecCalculator` is derived from `ElmMatVecCalc` which supply functions for calculating the element matrices. The following is a typical declaration of such a function, further details are found in the manual pages for the `FEM` class [9].

```
virtual void calcElmMatVec
  (
   int            elm_no,
   ElmMatVec&     elmat,
   FiniteElement& fe,
   IntegrandCalc& integrand,
   FEM*           fem
  );
```

We also need some temporary data structures to store the element matrices and administer the computation.

```
elmsys.rebind(new SystemCollector());
elmat.rebind(new ElmMatVecs());
```

`ElmMatVecs` is simply a matrix of `ElmMatVec`objects, in the present case a 2×2 system of element matrices. This class also enforces the essential boundary conditions in the different element matrices simultaneously.

The `SystemCollector` object `elmsys` is the manager that connects the block structured `LinEqMatrix` to the `DegFreeFE` objects and the classes containing the integrands for the system. It should be fed with a consistent set of `DegFreeFE`s and `IntegrandCalc` objects.

```
elmsys->redim(2,2);
for (int i=1; i<= 2; i++)
```

```
    for (int j=1; j<= 2; j++)
      if (integrands(i,j).ok())
        elmsys->attach(integrands(i,j)(),i,j);

  elmsys->attach(*calcelm);

  elmsys->attach(*u_dof,1,true);
  elmsys->attach(*u_dof,1,false);

  elmsys->attach(*v_dof,2,true);
  elmsys->attach(*v_dof,2,false);

  lineq->initAssemble(*elmsys);
  elmat->attach(*elmsys);
  elmsys->attach(*elmat);
  elmsys->attach(this);
```

The assembly of the linear system is performed by a function corresponding to the `makeSystem` call in common Diffpack simulators.

```
makeBlockSystem(*lineq,*elmsys,*coll,*elmat,1);
```

A suitable definition of this function can be found in `$NOR/mixed/Heart/block/HeartBlocks.cpp`.

Having now constructed the block linear system, we need to build the preconditioners. We will use the block Jacobi preconditioner introduced in sections 5.2.1 and 5.2.2 , but as stated above we will not consider multigrid preconditioners in this version of the simulator. We will instead use simple algebraic preconditioners to approximate the block inverses. The matrices needed by the preconditioner are assembled using the `IntegrandCalc` subclasses defined earlier. The following code segment handles the assembly of the preconditioner corresponding to block A in (5.41).

```
  Handle(IntegrandCalc) integrand;
  LinEqSystemBlockPrec& lins =
CAST_REF(lineq->getLinEqSystem(), LinEqSystemBlockPrec);
  MenuSystem& menu = SimCase::getMenuSystem();
  integrand = new Integrand11(this);

  Handle(Matrix_prm(NUMT)) mat_prec_prm;
  ...
  //initialize mat_prec_prm somehow
  ...
  Handle(Matrix(NUMT)) mat_prec;
  mat_prec= mat_prec_prm().create();

  makeSystem(u_dof(), *coll, *integrand, *calcelm, *mat_prec, 1);
```

The assembled matrix is attached to the preconditioner and then to the preconditioned linear system represented by the `LinEqSystemBlockPrec` `lineq`.

```
Handle(PrecBasis) prec_basis = new PrecBasis();
prec_basis->attach(*mat_prec);
prec_prm1->print(s_o);
lins.attach(*prec_prm1, *prec_basis,1);
```

Numerical results for this simulator are shown in Table 5.1. We have used the Conjugate Gradient method with RILU block preconditioners. We see that the number of iterations increases when the problem size increases. Because the cost of one iteration is proportional to the number of unknowns, the work of solving the problem is $\mathcal{O}(n^p)$, with $p > 1$.

Table 5.1. Numerical results for the Bidomain solver with RILU block preconditioning.

Unknowns	Iterations	p
882	34	-
3362	39	1.1
13122	56	1.3
51842	86	1.3
206082	150	1.3

5.3.5 A Simulator with Multigrid

The final step in the process of speeding up the simulator is to replace the simple algebraic preconditioners with efficient multigrid algorithms. The Multigrid tools in Diffpack are documented in [11] and the reader is referred to this chapter for more details concerning the `MGtools` class. In the present section we will see how the toolbox can be utilized to easily employ multigrid for block systems. The `HeartBlocksMG` simulator is a derived sub-class of the `HeartBlocks` class. In our problem we need two `MGtools` objects, one for each block on the diagonal, and we thus introduce the vector `VecSimplest(Handle(MGtools)) mg`. We also need to rewrite some of the functions in the `HeartBlocks`. The complete class declaration is as follows.

```
class HeartBlocksMG : public HeartBlocks
{
 public:
  int no_of_grids;
  VecSimplest(Handle(MGtools)) mg;
  Handle(FieldCollector) fieldcoll2;

  HeartBlocksMG();
 ~HeartBlocksMG();

  virtual void define (MenuSystem& menu, int level = MAIN);
  virtual void initFieldsEtc();
  virtual void scan ();
  virtual void solveAtThisTimeStep();
  virtual void fillEssBC(SpaceId space);
  virtual void mgFillEssBC(SpaceId space);
  virtual void getDataStructureOnGrid(SpaceId space);
```

```
  virtual void resultReport();
  virtual void makePreconditioner();
  virtual void saveResults();
  virtual void setIC();
};
```

As usual in Diffpack we have an extensive menu-interface which makes it easy to test various parameters. We use the command prefixes `block1` and `block2` to allow separate menu input for the two blocks.

```
menu.setCommandPrefix("block1");
Precond_prm ::defineStatic(menu, level+1);
menu.unsetCommandPrefix();
menu.setCommandPrefix("block2");
Precond_prm::defineStatic(menu, level+1);
menu.unsetCommandPrefix();
```

The `MGtools` menu is also initialized according to the Diffpack standard presented in [9].

```
MGtools::defineStatic(menu,level+1);
```

In the `scan` function the preconditioners are initialized and attached to the linear system.

```
menu.setCommandPrefix("block1");
prec_prm1.rebind( Precond_prm::construct());
prec_prm1->scan(menu);
menu.unsetCommandPrefix ();
menu.setCommandPrefix("block2");
prec_prm2.rebind( Precond_prm::construct());
prec_prm2->scan(menu);
menu.unsetCommandPrefix ();


LinEqSystemBlockPrec& lins =
    CAST_REF(lineq->getLinEqSystem(), LinEqSystemBlockPrec);
lins.attach(*prec_prm1,1);
lins.attach(*prec_prm2,2);
```

We also need to initialize the `VecSimplest` holding the `MGtools`. The length of the vector is set and each `MGtools` object is initialized according to the procedure described in [11].

```
mg.redim(2);
for (int i=1; i<= 2; i++){
  if ( i== 1){
mg(i).rebind(new MGtools(*this, oform("mg toolbox",i)));
  }
  else{
   //use the same grid hierarchy as mg(1) in mg(2):
mg(i).rebind(new MGtools(*this,mg(1)->gridcoll(),
       oform("mg toolbox",i)));
  }
  mg(i)->attach(lineq());
```

```
      mg(i)->scan(menu); //make grid etc.
      mg(i)->initStructure();
      if ( lins.getPrec(i).description().contains("multilevel")){
PrecML& p = CAST_REF(lins.getPrec(i), PrecML);
p.attach(mg(i)->getMLSolver());
      }
```

Both `MGtools` objects are now initialized properly and attached to the linear system `lineq`. The only remaining task in the construction of the preconditioners is to make the linear systems for the multigrid algorithms.

```
void HeartBlocksMG:: makePreconditioner() {
    int i;
    integrand_type = PRECONDITIONER;
    for (i=1; i<= 2; i++) {
      mg(i)->makeSystemMl(*integrands(i,i), *calcelm, true);
    }
}
```

The code segments presented here include all the major steps in the process of including the block multigrid preconditioner in the simulator. Because the simulator was already prepared for block linear systems and block preconditioning the addition of the multigrid preconditioner was fairly straightforward. The source code for the simulator is found in `$NOR/mixed/Heart/block/` in the files `HeartBlocksMG.h` and `HeartBlocksMG.cpp`.

Numerical results for the simulator with multigrid are shown in Table 5.2. We see that the number of iterations does not increase when the problem size increases. Because the number of iterations is constant and the cost of one iteration is proportional to the number of unknowns, the total work for one simulation is $\mathcal{O}(n)$.

Table 5.2. Numerical results for the Bidomain solver with multigrid block preconditioning.

Unknowns	Iterations
882	17
3362	17
13122	16
51842	15
206082	15

5.3.6 A Solver Based on Operator Splitting

The block structured preconditioners presented in this paper are all based on operator splitting techniques. These techniques may also be used as solvers

for the linear system. The simplest operator splitting algorithm is the block Jacobi method given by (5.3)-(5.4). A very similar algorithm is the block Gauss-Seidel iteration, defined by

$$\begin{aligned} \boldsymbol{x}^{k+1} &= x^k + \boldsymbol{A}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{B}\boldsymbol{y}^k) && (5.48)\\ &= \boldsymbol{A}^{-1}(\boldsymbol{b} - \boldsymbol{B}\boldsymbol{y}^k), && (5.49)\\ \boldsymbol{y}^{k+1} &= y^k + \boldsymbol{D}^{-1}(\boldsymbol{c} - \boldsymbol{C}\boldsymbol{x}^{k+1} - \boldsymbol{D}\boldsymbol{y}^k) && (5.50)\\ &= \boldsymbol{D}^{-1}(\boldsymbol{c} - \boldsymbol{C}\boldsymbol{x}^{k+1})\,. && (5.51) \end{aligned}$$

Here we have inverted $\boldsymbol{A}$ and $\boldsymbol{D}$ exactly. This leads to the simplified expression.

The major work of one iteration is the solution of the linear sub-systems (5.49) and (5.51), the efficiency of the algorithm depends on our ability to solve these linear systems efficiently. For these systems it is possible to use multigrid directly as a solver, by simply repeating the multigrid sweeps until convergence is reached. A slightly different approach is to use multigrid as preconditioner for the conjugate gradient method. We want to implement the new simulator as a sub-class of the `HeartBlocksMG` class described in the previous section, with as much reuse as possible, and thus the second approach seems like a good choice. To maximize the possibility for reuse, we employ the Gauss-Seidel method on linear algebra level. This means that we assemble the global linear system first, before it is decomposed into the sub-systems in (5.49)-(5.51).

A different, and perhaps more common, approach would be to split the equations on the PDE level and assemble the two linear systems separately. The main advantage of our approach is that we are able to reuse the code from the `HeartBlocksMG` simulator. Other advantages include a simple and efficient update of the right hand sides for each iteration, and that the residual for the complete linear system is easily available, to be used as a convergence test for the Gauss-Seidel iterations. The main disadvantage of the chosen approach is that the assembly of the complete linear system is more costly than assembling the two smaller sub-systems, and because of the need to store all the matrix blocks, the memory requirement is also increased.

Most of the datastructures and functions of the `HeartBlocksMG` class are reused unaltered in the new `HeartBlocksGS` simulator. The main addition to the class is that we need two new `LinEqAdmFE` objects to handle the linear systems (5.49)-(5.51). These linear systems are initialized and preconditioners attached by the following code segment in the scan function.

```
lineq1.rebind (new LinEqAdmFE(EXTERNAL_STORAGE));
lineq2.rebind (new LinEqAdmFE(EXTERNAL_STORAGE));

//use the same menu input for both lineqs
lineq1->scan(menu);
lineq2->scan(menu);

LinEqSystemPrec& lins1 =
```

```
   CAST_REF(lineq1->getLinEqSystem(), LinEqSystemPrec);
lins1.attach(*prec_prm1);

LinEqSystemPrec& lins2 =
   CAST_REF(lineq2->getLinEqSystem(), LinEqSystemPrec);
lins2.attach(*prec_prm1);

if ( lins1.getPrec().description().contains("multilevel")){
  PrecML& p = CAST_REF(lins1.getPrec(), PrecML);
  p.attach(mg(1)->getMLSolver());
}

if ( lins2.getPrec().description().contains("multilevel")){
  PrecML& p = CAST_REF(lins2.getPrec(), PrecML);
  p.attach(mg(2)->getMLSolver());
}
```

In addition to the additions made to the scan function, we need to change the function `solveAtThisTimeStep` to incorporate the new solution procedure. After assembling the global system, the following `do`-loop handles the Gauss-Seidel iterations.

```
do {
  //extract the u blocks as a separate system:
  Matrix(real)& block12 = lineq->Al().mat(1,2);

  block12.prod(linsol_block->getVec(2), rhs1);

  rhs1.add(lineq->bl().getVec(1),'-',rhs1);

  lineq1->attach(lineq->Al().mat(1,1),
      linsol_block->getVec(1), rhs1);

  //attach preconditioner for u:
  mg(1)->attach(lineq1->Al().mat(), no_of_grids);
  mg(1)->attachSol(lineq1->xl().vec(), no_of_grids);
  mg(1)->attachRhs(lineq1->bl().vec(), no_of_grids, false);
  //solve the linear system for u:
  lineq1->solve();

  //extract the v blocks as a separate system
  Matrix(real)& block21 = lineq->Al().mat(2,1);

   block21.prod(linsol_block->getVec(1), rhs2);
  rhs2.add(lineq->bl().getVec(2),'-',rhs2);

  lineq2->attach(lineq->Al().mat(2,2),
      linsol_block->getVec(2), rhs2);

  //attach preconditioner for v:
  mg(2)->attach(lineq2->Al().mat(), no_of_grids);
  mg(2)->attachSol(lineq2->xl().vec(), no_of_grids);
  mg(2)->attachRhs(lineq2->bl().vec(), no_of_grids, false);
  //solve the linear system for v
  lineq2->solve();                 // solve linear system

  lineq->getLinEqSystem().residual(res);
```

```
    r = res.norm();
    its ++;
  } while (r/r0 > tol);
```

We use the relative residual `r/r0` of the complete system to monitor the convergence. The same residual was used for the block preconditioned solvers described previously, and the performance of the simulators should thus be directly comparable. The source code for this simulator is found in `NOR/doc/mixed/Heart/block` in the files `HeartBlocksGS.h` and `HeartBlocksGS.cpp`. Results from numerical experiments with the Gauss-Seidel based simulator are presented in Table 5.3. With the given structure of the two sub-problems and the results achieved in the previous section, we expect multigrid to be an optimal preconditioner for solving the two sub-problems with the Conjugate-Gradient method. This is confirmed by the non-increasing iteration count for these sub-problems when the problem size increases. It is not quite as obvious that the number of Gauss-Seidel iterations is also independent of the problem size, but this is confirmed by the presented results. The present solution method is thus $\mathcal{O}(n)$, exactly as we had for the block-preconditioned solver with Conjugate Gradient iterations. Still, experiments indicate that the block preconditioned solver is significantly faster, with CPU times approximately 25% of the Gauss-Seidel based solver.

Table 5.3. Numerical results for the Bidomain solver with multigrid-based Gauss-Seidel iterations.

Unknowns	CG its u system	CG its v system	GS iterations
882	6	6	19
3362	6	5	16
13122	7	5	14
51842	7	6	12
206082	7	6	10

5.4 Two Saddle Point Problems

In the next sections we will consider two well known saddle point problems; the Stokes problem and the mixed formulation of the Poisson equation. These problems may both be written on the form

$$\mathcal{A}_h \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \mathbf{g} \end{bmatrix} . \tag{5.52}$$

This is a special case of (5.2) where $\boldsymbol{C}$ is $\boldsymbol{0}$ and $\boldsymbol{C}^{-1}$ is undefined. Therefore we cannot use the Jacobi method as preconditioner. But as we will see

the preconditioner will be block diagonal and is constructed similarly. These systems are indefinite and the discretization of these model problems require mixed elements as discussed in [10].

How should this preconditioner be constructed? Such systems have been studied in [13] and we follow their conclusion. The best preconditioner is of course $\mathcal{A}^{-1}$, which is obviously not an option. Our mixed system is indefinite, but we see that if we use the best preconditioner possible, $\mathcal{A}^{-1}$, we get a positive definite system. Hence, a natural question to ask, is whether the preconditioned system should be indefinite or definite. We seek an "approximation of the inverse" and an approximate transformation from an indefinite to a positive system may be dangerous. We may end up with eigenvalues close to zero, causing a breakdown of the basic iterative method. The cure is to make a preconditioner such that the preconditioned system is also an indefinite problem. Guided by [13] we consider block preconditioners on the form $\mathcal{B} = diag(\boldsymbol{M}, \boldsymbol{N})$ such that (5.19) has the following form

$$\mathcal{B}_h \mathcal{A}_h = \begin{bmatrix} \boldsymbol{MA} & \boldsymbol{MB}^T \\ \boldsymbol{NB} & \mathbf{0} \end{bmatrix}. \tag{5.53}$$

The preconditioners $\mathbf{M}$ and $\mathbf{L}$ are positive and therefore the preconditioned system will still be indefinite. The best possible such saddle point problem is

$$\mathcal{BA} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{Q}^T \\ \boldsymbol{Q} & \mathbf{0} \end{bmatrix}, \text{ where } \boldsymbol{QQ}^T = \boldsymbol{I}. \tag{5.54}$$

This system has eigenvalues, $1, 1/2 \pm 1/2\sqrt{5}$ (both positive and negative). Hence, the preconditioner is not similar to the inverse of the coefficient matrix. To construct a system similar to (5.54) we should have block operators such that,

$$\boldsymbol{MA} \sim \boldsymbol{I}, \tag{5.55}$$

$$\boldsymbol{NBMB}^T \sim \boldsymbol{I}. \tag{5.56}$$

If we are able to construct blocks with these properties, we will bound the eigenvalues of $\boldsymbol{BA}$ above and away from zero, independent of the mesh size h.

Pressure Schur Complement methods. In [16] they have formulated a general framework, a basic iteration on the Pressure Schur Complement, where the Projection method, Pressure correction methods, Uzawa methods and the Vanka smoother are special cases. Such a framework can be developed by using the block utilities in Diffpack in a similar way that is done in Section 5.3.6.

5.4.1 Stokes Problem

Mathematical Model The Stokes problem fits elegantly in the setup described in the previous section. We briefly recapture the equations from [10],

$$-\mu\Delta\boldsymbol{v} + \nabla p = \boldsymbol{f} \text{ in } \Omega, \quad \text{(equation of motion)}, \tag{5.57}$$

$$\nabla \cdot \boldsymbol{v} = 0 \text{ in } \Omega, \quad \text{(mass conservation)}, \tag{5.58}$$

$$\boldsymbol{v} = \boldsymbol{h} \text{ on } \partial\Omega_E, \tag{5.59}$$

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{n}} + p\boldsymbol{n} = 0 \text{ on } \partial\Omega_N\,. \tag{5.60}$$

Numerical Method We remember the discrete coefficient operator for the Stokes problem,

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}^T \\ \boldsymbol{B} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} -\mu\Delta & \nabla \\ \nabla^T & \mathbf{0} \end{bmatrix}. \tag{5.61}$$

To obtain a well posed discrete and continuous problem the Babuska-Brezzi condition have to be fulfilled [7], this means that we need to use the mixed elements described in [10]. Examples of elements are the Taylor-Hood, Mini and Crouzeix-Raviart.

Solution of the Linear System We can choose $\mathbf{M}$ as an efficient preconditioner for $\boldsymbol{A}$ since $\mathbf{M}\boldsymbol{A} \sim \boldsymbol{I}$. Such preconditioner can be made by multigrid because $\boldsymbol{A}$ is an elliptic operator. The Schur complement $\mathbf{B}\mathbf{M}\mathbf{B}^T$ is already well conditioned, thus $\boldsymbol{N}$ can be chosen as $\boldsymbol{I}$ (the mass matrix). We present some numerical experiments of this preconditioner in Section 5.4.1, similar experiments are shown in [13].

A Simulator for Stokes problem In [10] a Stokes simulator, using mixed finite elements, was developed. We will now extend this simulator to utilize efficient block preconditioners. Most of the steps to do this are the same as for the Bidomain problem presented in the previous section, but we then only used isoparametric elements. For the Stokes problem and the mixed Poisson problem discussed later, we need to use mixed elements or else (5.52) will be singular[5] [2]. The transition from isoparametric elements to mixed is first of all a query-replace,

`FEM`, `FiniteElement`, `integrand`, `makeSystem`, `IntegrandCalc`

is replaced with

[5] Notice that the pressure is only determined up to a constant and the system is therefore singular with a one dimensional kernel. However, when not using certain mixed elements, the kernel will be larger.

`MxFEM`, `MxFiniteElement`, `integrandMx`, `makeSystemMx`, `IntegrandCalcMx`.

In addition, it is important to know that the *general numbering* strategy of `DegFreeFE` is used instead of the *special numbering* which is used for isoparametric elements. The way `DegFreeFE` is constructed will determine which strategy it will use. A `DegFreeFE` made like,

```
GridFE grid;
DegFreeFE dof;
...
//make grid somehow
...
dof.rebind(new DegFreeFE(grid, 2)); //dof is made
```

results in the special numbering whereas, the general numbering is used if `DegFreeFE` is made like,

```
FieldsFE fields;
DegFreeFE dof;
fields_grid.rebind (new BasisFuncGrid (*grid));
String fields_elm_tp = menu.get ("fields element");
fields_grid->setElmType (fields_grid->getElmType(), fields_elm_tp);
fields.rebind (new FieldsFE (*fields_grid, nsd, "fields"));

dof.rebind(new DegFreeFE(fields)); //dof is made
```

The general numbering will be used even though isoparametric elements are used in `FieldsFE`. We turn on the mixed element utilities in `MGtools` by

```
MGtools::useFieldCollector(true,no_fields);
```

where `no_fields` is an integer. The fields of unknowns will then be collected in a `FieldCollector` object managed by the `MGtools`. When needed the fields should be fetched from there,

```
Handle(FieldsFE) v_coll;
v_coll.rebind( mg->fieldcoll->getFields(no_of_grids));
```

In contrast to `HeartBlocksMG` we made the fields ourselves by

```
coll.rebind(new FieldsFE(*grid, "coll"));
u.rebind (coll()(1));
v.rebind (coll()(2));
u_prev.rebind (new FieldFE (*grid, "u_prev"));
```

The implementation of the multigrid tools for the velocity preconditioner is more or less the same as for the Bidomain problem and we therefore skip the details. The source code can be found in `$NOR/doc/mixed/Stokes/block`.

However, now we also must make a preconditioner similar to $\boldsymbol{I}$, the mass matrix. It can be misleading to use the term identity when using finite element methods. However, it is the natural counterpart to the identity in finite difference methods. We therefore describe the term in greater detail. Assume

we have a function f and we want a finite element representation, $\sum_j u_j N_j$, of this function, then u_j is determined by

$$(\sum_j u_j N_j, N_i) = f_i = (f, N_i)\,. \tag{5.62}$$

Hence, we see that u_i and f_i are scaled differently. Inside the integration the coefficient u_j is multiplied roughly with N_j^2, whereas f_i is f multiplied by N_i. We make a preconditioner to account for this different scaling, namely the inverse of a lumped mass matrix. To do this we derive a preconditioner from `FEM` and `PrecUserDefProc`,

```
class PressurePrec : public PrecUserDefProc, public MxFEM {
  public:
  StokesBlocks* solver;
  Handle(Precond)    mass_prec;
  Handle(Precond_prm) mass_prm;
  Handle(Matrix_prm(NUMT)) mat_prm;
  Handle(DegFreeFE)  p_dof;
  Handle(Matrix(NUMT))          mass;
  bool               inited;

  PressurePrec(Precond_prm& prm):PrecUserDefProc(prm) {
    inited= false;
  }
  ~PressurePrec() {}

  virtual void scan(MenuSystem& menu);
  static void defineStatic(MenuSystem& menu, int level);
  virtual void init();
  virtual void attachPDof(DegFreeFE& dof) { p_dof.rebind(dof); }
  virtual void makeMassPrec();
  virtual void apply(
      const LinEqVector& c,
      LinEqVector& d,
      TransposeMode tpmode = NOT_TRANSPOSED
  );

  bool ok() { return inited; }

};
```

Any derived class of `PrecUserDefProc` must supply an `apply` function, which in our case simply is,

```
  void PressurePrec:: apply ( const LinEqVector& c_, LinEqVector& d_,
                              TransposeMode tpmode )
  {
    if  (!inited) init();
    mass->forwBack (( Vector(NUMT)&)c_.vec(), d_.vec());
  }
```

The `init` function should make the mass matrix and factorize it.

```
void PressurePrec:: init(){
  makeMassPrec();
  inited = true;
}
```

The mass matrix is made from the finite element fields related to the pressure, a `DegFreeFE` object `p_dof` containing the sufficient information and is fetched from the `Stokes` class. The mass matrix is made as follows,

```
void PressurePrec:: makeMassPrec() {
  if (p_dof->noEssBC()) {
    warningFP("PressurePrec:: makeMassPrec",
              "p_dof has no essential bc.");
  }
  mass.rebind(new MatDiag(NUMT)(p_dof->getTotalNoDof()));
  mass()=0.0;
  Handle(IntegrandMassP) integrand = new IntegrandMassP();
  integrand->P = 1;
  FEM::makeSystemMx(*p_dof, integrand.getPtr(), *mass);
  FactStrategy fact; fact.fact_tp=LU_FACT;
  mass->factorize(fact);
}
```

We make a diagonal mass matrix for fast inversion. It is sufficient to use the diagonal version since this matrix only should deal with the scaling in (5.62). The following `integrands` function makes the mass matrix.

```
void IntegrandMassP::integrandsMx
  (ElmMatVec& elmat, const MxFiniteElement& mfe)
{
  const int nsd = mfe.getNoSpaceDim();
  const int npbf = mfe(P).getNoBasisFunc();

  const real detJxW = mfe.detJxW();
  int   i,k,j;
  for (i = 1; i <= npbf; i++){
    for (j = 1; j <= npbf; j++){
       elmat.A(i,i) += mfe(P).N(i)*mfe(P).N(j)*detJxW;
    }
  }
}
```

Numerical Experiments In this section we test the efficiency of different block preconditioners. We fix $\boldsymbol{L}$ as the inverse of a lumped mass matrix. The comparison of $\boldsymbol{M}$ made be multigrid and RILU is shown in Table 5.4. We have used the Krylov method `SymMinRes` (Minimum residual method for symmetric and indefinite problems), with a random start vector. The convergence criterion is set as,

$$\frac{(\boldsymbol{B}r_k, r_k)}{(\boldsymbol{B}r_0, r_0)} \leq 10^{-4},$$

where r_k is the residual at iteration k and $\boldsymbol{B}$ is the preconditioner. This convergence monitor is implemented in Diffpack as `CMRelMixResidual`. The multigrid preconditioner is constructed as a V-cycle with SSOR with relaxation parameter 1.0 as smoother. On the coarsest grid we use Gauss elimination. As expected the multigrid preconditioner ensures convergence in (roughly) a fixed number of iterations, while the efficiency of the RILU preconditioner deteriorates as the number of unknowns increases. Similar results are obtained with the Crouzeix-Raviart and the Mini element (see the examples that follow the source code.).

Table 5.4. Numerical results for the Stokes problem with Taylor-Hood elements.

Unknowns	Iterations with RILU	Iterations with MG
187	25	16
659	34	19
2467	49	19
9539	72	21
37507	120	21
148739	158	21

5.4.2 Mixed Poisson Problem

Mathematical Model The mixed Poisson problem is

$$\boldsymbol{v} - \lambda \nabla p = 0 \text{ in } \Omega \quad \text{(Darcy's law)}, \tag{5.63}$$

$$\nabla \cdot \boldsymbol{v} = g \text{ in } \Omega \quad \text{(mass conservation)}\,. \tag{5.64}$$

The sign of p is changed to obtain a symmetric system with more efficient solution algorithms. More details on this problem and its implementation in Diffpack can be found in [10].

Numerical Method The coefficient operator is,

$$\mathcal{A}_h = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & -\nabla \\ \nabla\cdot & \boldsymbol{0} \end{bmatrix}\,. \tag{5.65}$$

Examples of elements that can be used are the Raviart-Thomas (see, e.g., [2,10]) and the new element [10,12], which we refered to as the robust element, in lack of a better name. Some usual Stokes elements, with continuous pressure elements, can also be used at the expense of loss of accuracy in the velocity field.

Solution of the Linear System Analogous with the preconditioner made in Section 5.4.1 we can choose $\boldsymbol{M}$ as the inverse of a lumped mass matrix, since $\boldsymbol{A} = \boldsymbol{I}$. The preconditioner $\boldsymbol{L}$ should then be made such that $\boldsymbol{LBMB}^T \sim \boldsymbol{I}$ and this means that $\boldsymbol{L} \sim \Delta^{-1}$. This preconditioner can be used in connection with the Mini and Taylor-Hood, because the pressure elements are continuous. We have implemented this and numerical experiments are shown in Section 5.4.2.

However, we are primarily interested in using discontinuous pressure elements, since the elements of particular interest are the Raviart-Thomas and the robust element. These elements are formulated as subspaces of $\boldsymbol{H}(div;\Omega)\times L^2(\Omega)$. The point is that the gradient on the pressure is transfered to a divergence on the velocity. The Δ-operator applied on these pressure elements is not straightforward.

One alternative is to use the auxiliary space technique [19]. This technique allows us to make a multigrid preconditioner based on standard continuous linear elements and employ it on the piecewise constant elements. We refer to [18] for the mathematical description and the requirements on the operators involved. Notice also that the numerical experiments done in this section have not been verified theoretically.

Let C and L be the spaces of piecewise constant elements and continuous linear elements, respectively. Furthermore, let $P : L \to C$ be an interpolation or projection operator, and P^* is the adjoint operator with respect to the L_2 inner product. The preconditioner is then,

$$L_C = S_C + P\, L_L\, P^* \,. \tag{5.66}$$

The basic ingredients are

- A multigrid preconditioner for continuous linear elements, L_L.
- Transfer operators mapping between the two different element spaces, P and P^*.
- A smoother for the piecewise constants, S_C.

The multigrid preconditioner made for the the Mini and the Taylor–Hood elements can be reused. The smoother is implemented as the inverse of a mass matrix multiplied with τh^2. This is similar to the $\boldsymbol{L}$-preconditioner we made for the Stokes problem (except that we multiply with τh^2). The transfer operators are made in a class `Interpolator`, which implements a general L_2 projection. The L_2 projection involves the making and inversion of mass matrices. This might seem like overkill, but the mass matrix has a small condition number and can be inverted sufficiently accurate quite efficiently. A few Conjugate Gradient iterations, with SSOR as the preconditioner, is usually sufficient. In our case the mass matrix is diagonal, because of the piecewise constants. Therefore, we can simply use on SSOR iteration to invert the mass matrix exactly.

The NCPressurePrecMG implements the above described preconditioner. We will comment the key points. The NCPressurePrecMG is a subclass of PressurePrecMG, which is a multigrid preconditioner similar to the one we implemented for the Stokes problem.

The new software in this preconditioner is the L_2 projection, Interpolator, which we will describe below. The Interpolator has a standard menu interface and can be made as follows,

```
interpolator.rebind(new Interpolator());
interpolator->scan(menu);
```

Additionally, it has to be initialized with the fields that we will transfer between.

```
interpolator->init(
   solver->p_dof().fields()(1),
   mgtools_laplace->getDof(no_of_grids).fields()(1));
```

These two steps are all that must be done to initialize the L_2 projection. We now apply this projection. The mapping from the piecewise constant elements onto the continuous linear element is then,

```
void NCPressurePrecMG::cons2linear(LinEqVector& from,
 LinEqVector& to)
{
   interpolator->interpolate(
     from.getVec(), DUAL, FIRST,
     to.getVec(), DUAL, SECOND);
}
```

The enum variable DUAL means that a right-hand side is transfered. A left-hand side transfer operation use the NODAL flag. The FIRST and SECOND flags mean that the vector from and to corresponds to the first and second field used in the initialization, Interpolator::init. The mapping from linear to constant elements is similar,

```
void NCPressurePrecMG::linear2cons(
 LinEqVector& from, LinEqVector& to)
{
   interpolator->interpolate(
     from.getVec(), NODAL, SECOND,
     to.getVec(), NODAL, FIRST);
}
```

The whole preconditioner in (5.66) is implemented as

```
void NCPressurePrecMG:: apply (
   const LinEqVector& c_,
   LinEqVector& d_,
   TransposeMode tpmode )
{
  if  (!inited) init();
```

```
  // init vectors
  LinEqVector& c_not_const = (LinEqVector&)c_;
  c->attach(c_not_const.getVec(1));
  d_ = 0.0;
  d->attach(d_.vec(1));
  tmp_vec->vec(1).fill(d->getVec(1));

  //lapace prec on linear elements
  cons2linear(*c, *lrhs);
  laplace_prec->apply(*lrhs, *llhs);
  linear2cons(*llhs, *tmp_vec);

  //smoother
  ssor_vec.fill( d->getVec());
  mass->SSOR1it(d->vec(), ssor_vec, c->vec(), 1.0);
  d->mult(tau*h_char*h_char);

  //add together
  d->add(*tmp_vec, *d);
}
```

There are several other alternatives to construct optimal preconditioners for the mixed Poisson problem (see, e.g., [15,1]).

Numerical Experiments We now show the results of some numerical experiments that document the efficiency of the preconditioner. We have tested the Taylor-Hood, Mini, Raviart-Thomas and the robust element. Notice that the results with the Raviart-Thomas and the robust elements have not been verified theoretically.

Table 5.5. Numerical results for the mixed Poisson problem with Taylor-Hood elements.

Unknowns	Iterations with RILU	Iterations with MG
187	26	22
659	26	25
2467	35	25
9539	58	24
37507	73	23
148739	50	21

The number of iterations to achieve the convergence criterion $\|\boldsymbol{r}_k\|/\|\boldsymbol{r}_0\| \leq 10^{-4}$ with the Taylor-Hood element is shown in Table 5.5. We obtain similar behavior with the Mini element, where we use the same preconditioner. The number of iterations needed with the RILU preconditioner increases as the number of unknowns increases. The multigrid preconditioner ensures a fixed

Table 5.6. Numerical results for the mixed Poisson problem with Raviart-Thomas elements.

Unknowns	Iterations with RILU	Iterations with MG
88	55	35
336	100	37
1312	195	44
5184	318	50
20608	442	52
82176	678	56

convergence rate independent of the number of unknowns. The input files for the Taylor-Hood experiment is `b_th.i` and `mg_th.i`, while the Mini element is tested with the `b_mini.i` and the `mg_mini.i` input files.

The preconditioner Raviart-Thomas and the robust element is on the form (5.66). Table 5.6 shows the results with the Raviart-Thomas element, it seems that the preconditioner is about as effiecient as in the case of the Taylor-Hood elements. The robust element behaves the same way. The Raviart-Thomas element is tested in `b_rt.i` and `mg_rt.i`, while the robust element is tested in `b_new.i` and `mg_new.i`. The complete source code for the block preconditioned mixed Poisson problem is in `$NOR/doc/mixed/PoissonMx/block`.

References

1. D. N. Arnold, R. S. Falk, and R. Winther. Preconditioning in H(div) and applications. *Math. Comp. 66*, 1997.
2. F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Methods.* Springer-Verlag, 1991.
3. A. M. Bruaset. *A Survey of Preconditioned Iterative Methods.* Addison-Wesley Pitman, 1995.
4. A. M. Bruaset and H. P. Langtangen. Object-oriented design of preconditioned iterative methods in Diffpack. *Transactions on Mathematical Software*, 23:50–80, 1997.
5. Craig C. Douglas, Jonathan Hu, Markus Kowarschik, and Ulrich Rude. Cache based algorithms. In *MGNet Virtual Proceedings 2001*, *http://www.mgnet.org/mgnet-cm2001.html*, 2001.
6. D. B. Geselowitz and W. T. Miller. A bidomain model for anisotropic cardiac muscle. *Annals of Biomedical Engineering*, 11:191–206, 1983.
7. V. Girault and P. A. Raviart. *Finite Element Methods for Navier-Stokes Equations.* Springer-Verlag, 1986.
8. W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations.* Springer-Verlag, 1994.
9. H. P. Langtangen. *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming.* Textbooks in Computational Science and Engineering. Springer, 2nd edition, 2003.

10. K.-A. Mardal and H. P. Langtangen. Mixed finite elements in Diffpack. In *Computational Partial Differential Equations using Diffpack*. Springer, 2003.
11. K.-A. Mardal, H. P. Langtangen, and G.W. Zumbusch. Multigrid methods in diffpack. In *Computational Partial Differential Equations using Diffpack*. Springer, 2003.
12. K.-A. Mardal, X.-C. Tai, and R. Winther. Robust finite elements for Darcy–Stokes flow.
13. T. Rusten and R. Winther. A preconditioned iterative method for saddlepoint problems. *SIAM J. Matrix Anal.*, 1992.
14. J. Sundnes, G. T. Lines, P. Grøttum, and A. Tveito. Numerical methods and software for modeling the electrical activity in the human heart. In H. P. Langtangen and A. Tveito, editors, *Computational Partial Differential Equations using Diffpack - Advanced Topics*. Springer, 2002.
15. P. S. Vassilevski T. Rusten and R. Winther. Interior penalty preconditioners for mixed finite element approximations of elliptic problems. *Mathematics of Computation*, 1996.
16. S. Turek. *Efficient Solvers for Incompressible Flow Problems*. Springer, 1999.
17. C. Oosterlee U. Trottenberg and A. Schuller. *Multigrid*. Academic Press, 2001.
18. J. Xu. Iterative methods for space decomosision and subspace correction. *SIAM Review*, 1992.
19. J. Xu. The auxiliary space method and optimal multigrid preconditioning techniques forunstructured grids. *Computing*, 56:215–235, 1996.

V

Uniform Preconditioners for the Time Dependent Stokes Problem

K.-A. Mardal and R. Winther

Submitted to *Numerische Mathematik*.

UNIFORM PRECONDITIONERS FOR THE TIME DEPENDENT STOKES PROBLEM

KENT ANDRE MARDAL AND RAGNAR WINTHER

ABSTRACT. Implicit time stepping procedures for the time dependent Stokes problem lead to stationary singular perturbation problems at each time step. These singular perturbation problems are systems of saddle point type, which formally approach a mixed formulation of the Poisson equation as the time step tends to zero. The purpose of the present paper is to design preconditioners for discrete analogs of these systems, where the spatial discretization is derived from standard finite element procedures. By using standard positive definite elliptic preconditioners as building blocks we construct preconditioners which lead to condition numbers which are bounded uniformly with respect to the time step and the spatial discretization.

1. INTRODUCTION

Let $\Omega \subset \mathbb{R}^n$, with n=2 or 3, be a bounded polygonal domain with boundary $\partial\Omega$. Consider the corresponding initial value problem for the time dependent Stokes problem given by:

$$\begin{aligned} \boldsymbol{u}_t - \boldsymbol{\Delta u} - \mathbf{grad}\, p &= \boldsymbol{f} \quad \text{in } \Omega \times \mathbb{R}^+, \\ \operatorname{div} \boldsymbol{u} &= 0 \quad \text{in } \Omega \times \mathbb{R}^+, \\ \boldsymbol{u} &= 0 \quad \text{on } \partial\Omega \times \mathbb{R}^+, \\ \boldsymbol{u} &= \boldsymbol{u}_0 \quad \text{on } \Omega \times \{t = 0\}. \end{aligned} \tag{1.1}$$

If this initial value problem is discretized by an implicit time stepping procedure we are lead to stationary singular perturbation problems of the form:

$$\begin{aligned} (\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta})\boldsymbol{u} - \mathbf{grad}\, p &= \boldsymbol{f} \quad \text{in } \Omega, \\ \operatorname{div} \boldsymbol{u} &= 0 \quad \text{in } \Omega, \\ \boldsymbol{u} &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{1.2}$$

Here $\varepsilon > 0$ is the square root of the time step, while p is related to the original pressure by a scaling with the factor ε^2. The new right-hand side $\boldsymbol{f}$ represents a combination of $\boldsymbol{u}$ at the previous time step and the original forcing term. When such implicit time stepping procedures are combined with a finite element discretization of the spatial variables, then at least one discrete analog of the system (1.2) has to be solved for each time step. Hence, the efficiency of such solution strategies may depend critically on the development of iterative solvers for discretizations of systems of the form (1.2).

We recall that many semi–implicit time stepping schemes for the full nonlinear, incompressible Navier–Stokes equation use discrete analogs of linear systems of the form (1.2) as building blocks. These systems are then combined with a proper method for the nonlinear convection process by a fractional step strategy, cf. for example [11], or by an approach using Lagrangian coordinates "to remove" the convective term, cf. [17].

The main purpose of the present paper is to discuss preconditioners for discrete analogs of the system (1.2) when the perturbation parameter ε is allowed to be arbitrary small. More precisely, we shall assume that $\varepsilon \in (0, 1]$, and our goal is to design precondioners which leads to condition numbers which are bounded uniformly with respect to both ε and the discretization parameter h.

We note that when ε is not too small the system (1.2) is similar to the stationary Stokes problem, but with an additional lower order term. However, if ε approaches zero then the system formally tends to a mixed formulation of the Poisson equation. This observation can potentially indicate some problems for the corresponding discrete systems, since standard stable finite elements for the Stokes problem may not be stable for the mixed Poisson system. On the other hand, most stable elements for the mixed Poisson system, like the Raviart–Thomas elements, lack some of the continuity conditions required for conforming approximations of the Stokes system. In fact, this issue was discussed in great detail in [14], where systems of the form (1.2) was motivated as models for "averaged fluid flow." It was established that if standard $\boldsymbol{H}(\mathrm{div}) \times L^2$ norms was used for the mixed Poisson system then none of the most common Stokes elements appeared to be stable, and as consequence, these elements did not perform well for ε small. Motivated by this observation a new family of finite elements were constructed, with stability properties which are uniform with respect to the perturbation parameter ε.

However, for our study here the starting point is rather different. When the system (1.2) is derived from implicit time stepping procedures for the time dependent Stokes system the parameter ε is not a physical parameter. Difficulties which may occur as a consequence of ε being small should therefore be seen as instabilities created by the time stepping procedure, and not as instabilities created by the spatial discretizations of the time dependent Stokes system. We shall therefore in this paper study preconditioners for the system (1.2), discretized by standard Stokes elements. Uniform preconditioners with respect to ε and h will be derived, even if these systems have the instability properties indicated above. The main tool for deriving these preconditioners will be proper stability estimates in ε–dependent norms, but these norms do not degenerate to the norm of $\boldsymbol{H}(\mathrm{div}) \times L^2$ for $\varepsilon = 0$. Instead, the norm for the reduced case will correspond to $(\boldsymbol{u}, p) \in \boldsymbol{L}^2 \times H^1$.

The results of this paper are closely related to the results of [3]. In [3] a class of uniform block diagonal preconditioners for the system (1.2) was constructed in a multilevel setting. A tight connection between the perturbation parameter ε, the mesh parameter h, and the multigrid algorithm was utilized in order to obtain uniform preconditioners. In contrast to this, the theory developed here makes no explicit use of a multilevel framework. We will basically construct block diagonal preconditioners, where each block is composed of standard second order elliptic preconditioners which can be constructed by any algorithm. The preconditioners analyzed in this paper resemble preconditioners in [19] for the full incompressible Navier–Stokes equation, in the sense that also here we construct the preconditioners by composing several simpler operators derived from proper subproblems.

In §2 below we introduce some useful notation and describe basic properties of the system (1.2). Uniform preconditioners for the continuous system is derived in §3 as a consequence of a uniform inf–sup property. In §4 we then use numerical experiments to study discrete versions of this preconditioner for some choices of finite element spaces with continuous pressures. We observe that these preconditioners seems to result in preconditioned systems which are well conditioned, uniformly with respect to the perturbation parameter ε and the mesh parameter h. In §5 we perform a detailed theoretical analysis of these discrete preconditioners. Finally, in §6 we investigate the properties of related preconditioners for finite elements with discontinuous pressures.

2. Preliminaries

For any Banach space X the associated norm will be denoted $\|\cdot\|_X$. If $H^m = H^m(\Omega)$ is the Sobolev space of functions on Ω with m derivatives in $L^2 = L^2(\Omega)$ we use the simpler notation $\|\cdot\|_m$ instead of $\|\cdot\|_{H^m}$. The space H_0^m is the closure in H^m of $C_0^\infty = C_0^\infty(\Omega)$. The dual space of H_0^m with respect to the L^2 inner product will be denoted by H^{-m}. Furthermore, L_0^2 will denote the space of L^2 functions with mean value zero. A space written in boldface denotes a n–vector valued analog of the corresponding scalar space, where n=2 or 3. The notation $(\cdot,\cdot)$ is used to denote the L^2 inner product on scalar, vector, and matrix valued functions, and to denote the duality pairing between H_0^m and H^{-m}. The gradient of a vector field $\boldsymbol{v}$ is denoted $\boldsymbol{Dv}$, i.e. $\boldsymbol{Dv}$ is the $n \times n$ matrix with elements

$$(\boldsymbol{Dv})_{i,j} = \partial \boldsymbol{v}_i / \partial x_j \quad 1 \le i, j \le n.$$

Hence, for any $\boldsymbol{u} \in \boldsymbol{H}^2$ and $\boldsymbol{v} \in \boldsymbol{H}_0^1$ we have

$$-(\boldsymbol{\Delta u}, \boldsymbol{v}) = (\boldsymbol{Du}, \boldsymbol{Dv}) \equiv \int_\Omega \boldsymbol{Du} : \boldsymbol{Dv}\, dx,$$

where the colon denotes the scalar product of matrix fields.

Below we shall encounter the intersection and sum of Hilbert spaces. We therefore recall the basic definitions of these concepts. If X and Y are Hilbert spaces, both continuously contained in some larger Hilbert spaces, then the intersection $X \cap Y$ and the sum $X + Y$ are themselves Hilbert spaces with the norms

$$\|z\|_{X\cap Y} = (\|z\|_X^2 + \|z\|_Y^2)^{1/2}$$

and

$$\|z\|_{X+Y} = \inf_{\substack{z=x+y \\ x\in X,\, y\in Y}} (\|x\|_X^2 + \|y\|_Y^2)^{1/2}.$$

Furthermore, if $X \cap Y$ is dense in both X and Y then

$$(X \cap Y)^* = X^* + Y^*. \tag{2.1}$$

Finally, if T is a bounded linear operator mapping X_1 to Y_1 and X_2 to Y_2, respectively, then

$$T \in \mathcal{L}(X_1 \cap X_2, Y_1 \cap Y_2) \cap \mathcal{L}(X_1 + X_2, Y_1 + Y_2).$$

In particular, we will later use the bound

$$\|T\|_{\mathcal{L}(X_1+X_2,Y_1+Y_2)} \leq \max(\|T\|_{\mathcal{L}(X_1,Y_1)}, \|T\|_{\mathcal{L}(X_2,Y_2)}). \tag{2.2}$$

We refer to [5, Chapter 2] for these results.

Throughout this paper $\varepsilon \in (0, 1]$, $a_\varepsilon(\cdot,\cdot) : \boldsymbol{H}^1 \times \boldsymbol{H}^1 \mapsto \mathbb{R}$ will denote the bilinear form

$$a_\varepsilon(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}, \boldsymbol{v}) + \varepsilon^2(\boldsymbol{D}\boldsymbol{u}, \boldsymbol{D}\boldsymbol{v}),$$

and $\boldsymbol{I} - \varepsilon^2\boldsymbol{\Delta} : \boldsymbol{H}_0^1 \mapsto \boldsymbol{H}^{-1}$ the corresponding operator, i.e.

$$((\boldsymbol{I} - \varepsilon^2\boldsymbol{\Delta})\boldsymbol{u}, \boldsymbol{v}) = a_\varepsilon(\boldsymbol{u}, \boldsymbol{v}) \qquad \forall \boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{H}_0^1.$$

A weak formulation of problem (1.2), slightly generalized to allow for a nonhomogeneous right hand side in the second equation, is given by:

Find $(\boldsymbol{u}, p) \in \boldsymbol{H}_0^1 \times L_0^2$ such that

$$\begin{aligned} a_\varepsilon(\boldsymbol{u}, \boldsymbol{v}) + (p, \operatorname{div} \boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v}) \qquad && \forall \boldsymbol{v} \in \boldsymbol{H}_0^1, \\ (\operatorname{div} \boldsymbol{u}, q) &= (g, q) \qquad && \forall q \in L_0^2. \end{aligned} \tag{2.3}$$

Here we assume that data $(\boldsymbol{f}, g)$ is given in $\boldsymbol{H}^{-1} \times L_0^2$.

The problem (2.3) has a unique solution $(\boldsymbol{u}, p) \in \boldsymbol{H}_0^1 \times L_0^2$. This follows from standard results for Stokes problem, cf. for example [10]. However, the bound on $(\boldsymbol{u}, p) \in \boldsymbol{H}_0^1 \times L_0^2$ will degenerate as ε tends to zero. In fact, for the reduced problem (2.3), with $\varepsilon = 0$ and the boundary condition modified such that only zero normal component is required, the space $\boldsymbol{H}_0^1 \times L_0^2$ is not a proper function space for the solution. However, the theory developed in [6] can be applied in this case if we seek $(\boldsymbol{u}, p)$ either in $\boldsymbol{H}_0(\operatorname{div}) \times L_0^2$ or in $\boldsymbol{L}^2 \times (H^1 \cap L_0^2)$, and with data $(\boldsymbol{f}, g)$ in the proper dual spaces. These results are in fact consequences of standard results for the Poisson equation. Here the space $\boldsymbol{H}_0(\operatorname{div})$ denotes the set of square integrable vector fields, with a

square integrable divergence, and with zero normal component on the boundary.

The fact that the regularity of the solution is changed when ε becomes zero strongly suggests that ε–dependent norms and function spaces are required in order to obtain stability estimates independent of ε. Furthermore, since the reduced problem is well posed for two completely different choices of function spaces, this indicates that there are at least two different choices of ε–dependent norms. These are the norms of the spaces $(\boldsymbol{H}_0(\mathrm{div}) \cap \varepsilon \cdot \boldsymbol{H}_0^1) \times L_0^2$ and $(\boldsymbol{L}^2 \cap \varepsilon \cdot \boldsymbol{H}_0^1) \times ((H^1 \cap L_0^2) + \varepsilon^{-1} \cdot L_0^2)$. Note that for any $\varepsilon > 0$ both these spaces are equal to $\boldsymbol{H}_0^1 \times L_0^2$ as a set, but as ε approaches zero the corresponding norms degenerates to the norm of $\boldsymbol{H}_0(\mathrm{div}) \times L_0^2$ or $\boldsymbol{L}^2 \times (H^1 \cap L_0^2)$, respectively.

In [14] it was established that most standard Stokes elements are not uniformly stable in the norm of $(\boldsymbol{H}_0(\mathrm{div}) \cap \varepsilon \cdot \boldsymbol{H}_0^1) \times L_0^2$. Therefore, if we want uniform stability estimates for such elements it seems more natural to use the norm induced by the space

$$\boldsymbol{X}_\varepsilon := (\boldsymbol{L}^2 \cap \varepsilon \cdot \boldsymbol{H}_0^1) \times ((H^1 \cap L_0^2) + \varepsilon^{-1} \cdot L_0^2).$$

This is the approach taken in this paper.

The norm of the space $\boldsymbol{L}^2 \cap \varepsilon \cdot \boldsymbol{H}_0^1$, will be denoted $\|\!| \cdot |\!\|_\varepsilon$, i.e.

$$|\!|\!| \boldsymbol{v} |\!|\!|_\varepsilon^2 = \|\boldsymbol{v}\|_0^2 + \varepsilon^2 \|\boldsymbol{D}\boldsymbol{v}\|_0^2,$$

while the norm in $(H^1 \cap L_0^2) + \varepsilon^{-1} \cdot L_0^2$ will be simplified to $|\cdot|_\varepsilon$. More precisely, we define, $|q|_\varepsilon$ by

$$|q|_\varepsilon = \inf_{\substack{q = q_1 + q_2 \\ q_1 \in H^1 \cap L_0^2,\, q_2 \in L_0^2}} (\|\operatorname{\mathbf{grad}} q_1\|_0^2 + \varepsilon^{-2} \|q_2\|_0^2)^{1/2}.$$

This notation for the norm in $(H^1 \cap L_0^2) + \varepsilon^{-1} \cdot L_0^2$ is convenient, but slightly unusual, since $|q|_0 = \|\operatorname{\mathbf{grad}} q\|_0$ is equivalent to $\|q\|_1$ on $H^1 \cap L_0^2$, while $|q|_1$ is equivalent to $\|q\|_0$.

Let $H^* \supset L_0^2$ denote the dual space of $H^1 \cap L_0^2$, and define the operator $(I - \varepsilon^2 \Delta)^{-1} : H^* \mapsto H^1 \cap L_0^2$ by a standard weak formulation, i.e. $p = (I - \varepsilon^2 \Delta)^{-1} g$ if p satisfies

$$(p, q) + \varepsilon^2 (\operatorname{\mathbf{grad}} p, \operatorname{\mathbf{grad}} q) = (g, q) \qquad \forall q \in H^1 \cap L_0^2.$$

By using this operator a more explicit characterization of $|q|_\varepsilon$ can be given. For $q \in L_0^2$ let $q_1 = (I - \varepsilon^2 \Delta)^{-1} q \in H^1 \cap L_0^2$. Note, in particular, that $\Delta q_1 \in L_0^2$. Furthermore, a straightforward computation shows that the solution of the minimization problem in the definition of $|q|_\varepsilon$ is given by q_1 and $q_2 = -\varepsilon^2 \Delta q_1 = q - q_1$. Hence, we obtain

$$|q|_\varepsilon^2 = \|\operatorname{\mathbf{grad}} q_1\|_0^2 + \varepsilon^{-2} \|q - q_1\|_0^2. \tag{2.4}$$

The system (2.3) can alternatively be written as

$$\mathcal{A}_\varepsilon \begin{pmatrix} \boldsymbol{u} \\ p \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} \\ g \end{pmatrix}, \tag{2.5}$$

where the coefficient operator $\mathcal{A}_\varepsilon$ is given by

$$\mathcal{A}_\varepsilon = \begin{pmatrix} \boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta} & -\mathbf{grad} \\ \operatorname{div} & 0 \end{pmatrix}. \tag{2.6}$$

Here $-\mathbf{grad} : L_0^2 \mapsto \boldsymbol{H}^{-1}$ is the dual of the divergence operator, $\operatorname{div} : \boldsymbol{H}_0^1 \mapsto L_0^2$.

Let $\boldsymbol{X}_\varepsilon^*$ be the dual space of $\boldsymbol{X}_\varepsilon$. Because of (2.1) this space can be expressed as

$$\boldsymbol{X}_\varepsilon^* = (\boldsymbol{L}^2 + \varepsilon^{-1}\boldsymbol{H}^{-1}) \times (\varepsilon \cdot L_0^2 \cap H^*).$$

We shall show below that the operator $\mathcal{A}_\varepsilon$ is an isomorphism mapping $\boldsymbol{X}_\varepsilon$ into $\boldsymbol{X}_\varepsilon^*$. Furthermore, the corresponding operator norms

$$||\mathcal{A}_\varepsilon||_{\mathcal{L}(X_\varepsilon, X_\varepsilon^*)} \quad \text{and} \quad ||\mathcal{A}_\varepsilon^{-1}||_{\mathcal{L}(X_\varepsilon^*, X_\varepsilon)} \quad \text{are independent of } \varepsilon. \tag{2.7}$$

In fact, with the definitions above, this is also true for $\varepsilon \in [0, 1]$, i.e. the endpoint $\varepsilon = 0$ can be included. However, in the discussion below we will for simplicity always assume that $\varepsilon > 0$.

The uniform boundedness of $\mathcal{A}_\varepsilon$ is straightforward to check from the definitions above. For example, if $p = p_1 + p_2$, where $p_1 \in H^1 \cap L_0^2, p_2 \in L_0^2$, then the term $(p, \operatorname{div} \boldsymbol{v})$ in (2.3) can be bounded by

$$\begin{aligned} |(p, \operatorname{div} \boldsymbol{v})| &= |(p_1 + p_2, \operatorname{div} \boldsymbol{v})| \\ &= |-(\mathbf{grad}\, p_1, \boldsymbol{v}) + (p_2, \operatorname{div} \boldsymbol{v})| \\ &\le (\|p_1\|_1 \|\boldsymbol{v}\|_0 + \|p_2\|_0 \| \operatorname{div} \boldsymbol{v}\|_0) \\ &\le (\|p_1\|_1^2 + \varepsilon^{-2}\|p_2\|_0^2)^{1/2} |\!|\!| \boldsymbol{v} |\!|\!|_\varepsilon. \end{aligned}$$

Hence,

$$|(p, \operatorname{div} \boldsymbol{v})| \le |p|_\varepsilon |\!|\!| \boldsymbol{v} |\!|\!|_\varepsilon \quad \forall \boldsymbol{v} \in \boldsymbol{H}_0^1, p \in L_0^2. \tag{2.8}$$

The uniform boundedness of $\mathcal{A}_\varepsilon^{-1}$ can be verified from the two Brezzi conditions, cf. [6]. For the present problem these conditions read:

There are constants $\alpha_0 > 0, \beta_0 > 0$, independent of ε, such that

$$\sup_{\boldsymbol{v} \in \boldsymbol{H}_0^1} \frac{(q, \operatorname{div} \boldsymbol{v})}{|\!|\!| \boldsymbol{v} |\!|\!|_\varepsilon} \ge \alpha_0 |q|_\varepsilon \quad \forall q \in L_0^2 \tag{2.9}$$

and

$$a_\varepsilon(\boldsymbol{v}, \boldsymbol{v}) \ge \beta_0 |\!|\!| \boldsymbol{v} |\!|\!|_\varepsilon^2 \quad \forall \boldsymbol{v} \in \boldsymbol{H}_0^1. \tag{2.10}$$

Condition (2.10) obviously holds with $\beta_0 = 1$, while condition (2.9) will be verified in the next section.

3. Mapping properties and uniform preconditioners

As explained above the main purpose of the present paper is to construct uniform preconditioners for discrete analogs of the system (1.2) when this system has been discretized by standard Stokes elements. The presentation of these preconditioners will be given in the next section. However, in order to motivate these preconditioners we will in

this section explain how to precondition the continuous problem. Similar discussions, where preconditioners for various discrete systems are motivated from mapping properties of the corresponding continuous systems, can for example be found in [2] and [13].

We will first establish the uniform inf–sup condition (2.9). If $\varepsilon = 1$ then this condition, up to equivalence of norms, reduces to

$$\sup_{\boldsymbol{v}\in\boldsymbol{H}_0^1} \frac{(q, \operatorname{div} \boldsymbol{v})}{\|\boldsymbol{v}\|_1} \geq \alpha_1 \|q\|_0 \quad \forall q \in L_0^2, \tag{3.1}$$

where $\alpha_1 > 0$. This is the standard inf–sup condition for the Stokes system which is well–known to hold. This result was established for a Lipschitz domain by Nečas [15], cf. also [10, Chapter 1, Corollary 2.4]. The uniform inf–sup condition (2.9) is now a simple consequence of (2.2), (3.1), and Poincaré's inequality.

Lemma 3.1. *The uniform inf–sup condition* (2.9) *holds.*

Proof. Observe that (3.1) can be written in the form

$$\|\operatorname{\mathbf{grad}} q\|_{-1} \geq \alpha_1 \|q\|_0 \quad \forall q \in L_0^2.$$

Hence, if we define $\boldsymbol{G}_0 = \operatorname{\mathbf{grad}}(L_0^2)$ then $\boldsymbol{G}_0$ is a closed subspace of $\boldsymbol{H}^{-1}$, and we can define $\operatorname{\mathbf{grad}}^{-1} : \boldsymbol{G}_0 \mapsto L_0^2$ such that

$$\|\operatorname{\mathbf{grad}}^{-1}\|_{\mathcal{L}(\boldsymbol{G}_0, L_0^2)} \leq \alpha_1^{-1}.$$

In addition, Poincaré's inequality states that there is a constant $c = c(\Omega)$ such that

$$\|q\|_1 \leq c \|\operatorname{\mathbf{grad}} q\|_0 \quad \forall q \in H^1 \cap L_0^2,$$

or $\|\operatorname{\mathbf{grad}}^{-1}\|_{\mathcal{L}(\boldsymbol{G}_1, H^1\cap L_0^2)} \leq c$, where $\boldsymbol{G}_1 = \operatorname{\mathbf{grad}}(H^1 \cap L_0^2)$. We therefore conclude from (2.2) that

$$\|\operatorname{\mathbf{grad}}^{-1}\|_{\mathcal{L}(\boldsymbol{G}_1+\varepsilon^{-1}\boldsymbol{G}_0, (H^1\cap L_0^2)+\varepsilon^{-1}L_0^2)} \leq \max(c, \alpha_1^{-1}).$$

Hence, letting $\alpha_0 = \min(\alpha_1, c^{-1})$ we obtain

$$\|\operatorname{\mathbf{grad}} q\|_{\boldsymbol{L}^2+\varepsilon^{-1}\boldsymbol{H}^{-1}} \geq \alpha_0 |q|_\varepsilon \quad \forall q \in L_0^2,$$

which is (2.9). □

Let $\mathcal{B}_\varepsilon : \boldsymbol{X}_\varepsilon^* \mapsto \boldsymbol{X}_\varepsilon$ be the diagonal operator

$$\mathcal{B}_\varepsilon = \begin{pmatrix} (\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta})^{-1} & 0 \\ 0 & \varepsilon^2 I + (-\Delta)^{-1} \end{pmatrix}. \tag{3.2}$$

When restricted to $\boldsymbol{L}^2 \times L_0^2$ this operator is symmetric and positive definite. We observe that $\mathcal{B}_0 = \operatorname{diag}(\boldsymbol{I}, (-\Delta)^{-1})$, while $\mathcal{B}_1$ has the same mapping property as $\operatorname{diag}((-\boldsymbol{\Delta})^{-1}, I)$. In fact, it follows directly from the definitions of the spaces $\boldsymbol{X}_\varepsilon$ and $\boldsymbol{X}_\varepsilon^*$ that the operator norms

$$||\mathcal{B}_\varepsilon||_{\mathcal{L}(X_\varepsilon^*, X_\varepsilon)} \quad \text{and} \quad ||\mathcal{B}_\varepsilon^{-1}||_{\mathcal{L}(X_\varepsilon, X_\varepsilon^*)} \quad \text{are independent of } \varepsilon. \tag{3.3}$$

Hence, the composition

$$\mathcal{B}_\varepsilon \mathcal{A}_\varepsilon : \quad \boldsymbol{X}_\varepsilon \xrightarrow{\mathcal{A}_\varepsilon} \boldsymbol{X}_\varepsilon^* \xrightarrow{\mathcal{B}_\varepsilon} \boldsymbol{X}_\varepsilon \tag{3.4}$$

maps $\boldsymbol{X}_\varepsilon$ into itself. In particular, we can conclude from (2.7) and (3.3) that the operator norms

$$||\mathcal{B}_\varepsilon \mathcal{A}_\varepsilon||_{\mathcal{L}(X_\varepsilon, X_\varepsilon)}, \; ||(\mathcal{B}_\varepsilon \mathcal{A}_\varepsilon)^{-1}||_{\mathcal{L}(X_\varepsilon, X_\varepsilon)} \quad \text{are independent of } \varepsilon. \tag{3.5}$$

Furthermore, observe that the symmetric positive definite operator $\mathcal{B}_\varepsilon^{-1}$ defines an inner product on $\boldsymbol{X}_\varepsilon$, and that that the operator $\mathcal{B}_\varepsilon \mathcal{A}_\varepsilon$ is symmetric with respect to this inner product.

Consider now the preconditioned version of the system (2.3), or (2.5), given by

$$\mathcal{B}_\varepsilon \mathcal{A}_\varepsilon \begin{pmatrix} \boldsymbol{u} \\ p \end{pmatrix} = \mathcal{B}_\varepsilon \begin{pmatrix} \boldsymbol{f} \\ g \end{pmatrix}, \tag{3.6}$$

where the operator $\mathcal{B}_\varepsilon$, introduced above, is a preconditioner. This preconditioned differential system has a symmetric and bounded coefficient operator. Therefore, the system (3.6) can, in theory, be solved by an iterative method like the minimum residual method (cf. for example [12]) or the conjugate gradient method applied to the normal equations. These methods are well defined as long as $\mathcal{B}_\varepsilon \mathcal{A}_\varepsilon$ maps $\boldsymbol{X}_\varepsilon$ into itself, and the convergence in the norm of $\boldsymbol{X}_\varepsilon$ can be bounded by the spectral condition number

$$\kappa(\mathcal{B}_\varepsilon \mathcal{A}_\varepsilon) = ||\mathcal{B}_\varepsilon \mathcal{A}_\varepsilon||_{\mathcal{L}(X_\varepsilon, X_\varepsilon)} \cdot ||(\mathcal{B}_\varepsilon \mathcal{A}_\varepsilon)^{-1}||_{\mathcal{L}(X_\varepsilon, X_\varepsilon)}.$$

Hence, property (3.5) ensures uniform convergence with respect to ε.

If the system (2.3) is replaced by a discrete analog, with a discrete coefficient operator $\mathcal{A}_{\varepsilon,h}$, then the corresponding preconditioner $\mathcal{B}_{\varepsilon,h}$ should also be constructed on discrete spaces. However, the continuous discussion given above suggests clearly the structure of these preconditioners. Of course, in order to obtain computational efficiency, the inverse operators appearing in the blocks of $\mathcal{B}_\varepsilon$ should then be replaced by proper elliptic preconditioners.

4. The discrete preconditioners

A standard finite element discretization of (2.3) leads to discrete indefinite systems approximating (2.3). Motivated by the continuous discussion above we will propose preconditioners for these discrete systems. The behavior of these preconditioners will then be investigated by numerical experiments, while a theoretical discussion is given in the next section.

4.1. **Finite element discretization.** Let $\{\boldsymbol{V}_h \times Q_h\}_{h\in(0,1]} \subset \boldsymbol{H}_0^1 \times L_0^2$ be finite element spaces, where the parameter h represents the scale of the discretization. Given the spaces $\boldsymbol{V}_h$ and Q_h the corresponding finite element discretization of the system (2.3) is given by:

Find $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_h \times Q_h$ such that

$$\begin{aligned} a_\varepsilon(\boldsymbol{u}_h, \boldsymbol{v}) + (p_h, \operatorname{div} \boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_h, \\ (\operatorname{div} \boldsymbol{u}_h, q) &= (g, q) \qquad \forall q \in Q_h. \end{aligned} \tag{4.1}$$

Standard stable Stokes elements satisfies a Babuska–Brezzi condition of the form

$$\sup_{\boldsymbol{v}\in\boldsymbol{V}_h} \frac{(q, \operatorname{div} \boldsymbol{v})}{\|\boldsymbol{v}\|_1} \geq \alpha_1 \|q\|_0 \quad \forall q \in Q_h, \tag{4.2}$$

where the positive constant α_1 is independent of the mesh parameter h. For a review of such finite element spaces we refere for example to the texts [7] and [10]. We note that (4.2) is a discrete version of (2.9) in the case when the perturbation parameter ε is bounded away from zero. This condition will imply, in particular, that the discrete system (4.1) has a unique solution.

The discrete system (4.1) can alternatively be written as a discrete analog of (2.5),

$$\mathcal{A}_{\varepsilon,h} \begin{pmatrix} \boldsymbol{u}_h \\ p_h \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_h \\ g_h \end{pmatrix}, \tag{4.3}$$

where the discrete coefficient operator $\mathcal{A}_{\varepsilon,h} : \boldsymbol{V}_h \times Q_h \mapsto \boldsymbol{V}_h \times Q_h$ is defined by

$$(\mathcal{A}_{\varepsilon,h} \begin{pmatrix} \boldsymbol{u} \\ p \end{pmatrix}, \begin{pmatrix} \boldsymbol{v} \\ q \end{pmatrix}) = a_\varepsilon(\boldsymbol{u}, \boldsymbol{v}) + (p, \operatorname{div} \boldsymbol{v}) + (\operatorname{div} \boldsymbol{u}, q) \tag{4.4}$$

for all $(\boldsymbol{u}, p), (\boldsymbol{v}, q) \in \boldsymbol{V}_h \times Q_h$. Hence, the operator $\mathcal{A}_{\varepsilon,h}$ is an L^2–symmetric, but indefinite, operator mapping the product space $\boldsymbol{V}_h \times Q_h$ into itself.

Our goal is to construct efficient positive definite, block diagonal preconditioners for the operator $\mathcal{A}_{\varepsilon,h}$, i.e. we will construct block diagonal operators $\mathcal{B}_{\varepsilon,h} : \boldsymbol{V}_h \times Q_h \mapsto \boldsymbol{V}_h \times Q_h$ such that the condition numbers of the operators $\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}$ are bounded uniformly in the perturbation parameter ε and the discretization parameter h. The preconditioners $\mathcal{B}_{\varepsilon,h}$, constructed below, are designed as proper discrete analogs of the operator $\mathcal{B}_\varepsilon$ introduced above.

Motivated by (3.2), the preconditioner $\mathcal{B}_{\varepsilon,h}$ will be constructed on the form

$$\mathcal{B}_{\varepsilon,h} = \begin{pmatrix} \boldsymbol{M}_{\varepsilon,h} & 0 \\ 0 & \varepsilon^2 I_h + N_h \end{pmatrix}. \tag{4.5}$$

Here $\boldsymbol{M}_{\varepsilon,h} : \boldsymbol{V}_h \mapsto \boldsymbol{V}_h$ is a preconditioner for the discrete version of the differential operator $\boldsymbol{I} - \varepsilon^2\boldsymbol{\Delta}$ with Dirichlet boundary conditions, while the operator $N_h : Q_h \mapsto Q_h$ is a corresponding preconditioner for the

discrete negative Laplacian with natural boundary conditions. Finally, the operator $I_h : Q_h \mapsto Q_h$ is the identity operator if the space Q_h consists of discontinuous functions or an operator spectrally equivalent to the identity on C^0–elements. In fact, in the present section we will only consider finite elements spaces with continuous approximations of the pressure. The reason for this is that the presence of the negative Laplacian preconditioner N_h, defined on Q_h, seems to demand that $Q_h \subset H^1$, at least as long as conforming approximations of the Laplacian are used. Hence, below we shall consider the Mini element [1] and the classical Taylor–Hood elements [7], [10].

The most efficient iterative method for the preconditioned system

$$\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}\begin{pmatrix}\boldsymbol{u}_h\\ p_h\end{pmatrix} = \mathcal{B}_{\varepsilon,h}\begin{pmatrix}\boldsymbol{f}_h\\ g_h\end{pmatrix}$$

is probably the preconditioned minimum residual method, cf. [12], [16], [18]. Alternatively, we can use the conjugate gradient method applied to the normal equations of the preconditioned system.

4.2. **Numerical experiments.** In order to test the behavior of the discrete preconditioners of the form (4.5) we will consider the system (4.1) with the domain Ω taken as the unit square in $\mathbb{R}^2$. A sequence of rectangular meshes is constructed by uniform refinements of a 2×2 partition of the unit square, and a triangular mesh is constructed by dividing each rectangle into two triangles by the diagonal with negative slope. The number of unknowns in the experiments below will typically range from order 10^2 to order 10^5.

In the two examples below the pressure space Q_h consists of continuous piecewise linear functions. The preconditioner $N_h : Q_h \mapsto Q_h$ is a standard V–cycle operator with a symmetric Gauss–Seidel smoother, while the approximate identity I_h on Q_h simply consists of one symmetric Gauss–Seidel iteration. The condition numbers for the operators $N_h(-\Delta_h)$ and I_h, where $\Delta_h : Q_h \mapsto Q_h$ is the corresponding discrete Laplace operator, can be estimated by a standard procedure from the preconditioned conjugate gradient method, where we have chosen an oscillatory random vector as a start vector. The iteration is terminated when the residual is reduced by a factor of 10^{-17}.

A similar approach is used to estimate the condition number of $\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}$, where we recall that $\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}$ are symmetric with respect to the inner product generated by $\mathcal{B}_{\varepsilon,h}^{-1}$. These estimates for $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}$ are based on the Conjugate Gradient method applied to the normal system

$$\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}\begin{pmatrix}\boldsymbol{u}_h\\ p_h\end{pmatrix} = \mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}\mathcal{B}_{\varepsilon,h}\begin{pmatrix}\boldsymbol{f}_h\\ g_h\end{pmatrix}.$$

Estimates for the condition numbers of $N_h(\Delta_h)$ and I_h are given in Table 4.1.

h	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
$\kappa(N_h(-\Delta_h))$	1.71	1.50	1.47	1.47	1.47	1.47
$\kappa(I_h)$	1.66	1.62	1.61	1.60	1.60	1.60

TABLE 4.1. Condition numbers for the operators $N_h(-\Delta_h)$ and I_h.

We observe that these operators cleary behave well as the mesh paramer h is decreased. In the examples below the preconditioners N_h and I_h are combined with proper operators $\boldsymbol{M}_{\varepsilon,h}$ to build the complete block diagonal preconditioner $\mathcal{B}_{\varepsilon,h}$ of the form (4.5).

Example 4.1 Consider the discretization given by the Mini element proposed in [1], i.e. $\boldsymbol{V}_h \subset \boldsymbol{H}_0^1$ consists of piecewise linear functions and cubic bubble functions supported on a single triangle, while $Q_h \subset H^1 \cap L_0^2$ is the space of continuous piecewise linear functions. The preconditioner $\boldsymbol{M}_{\varepsilon,h} : \boldsymbol{V}_h \mapsto \boldsymbol{V}_h$, approximating $(\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta}_h)^{-1}$, where $\boldsymbol{\Delta}_h : \boldsymbol{V}_h \mapsto \boldsymbol{V}_h$ is the corresponding discrete Laplacian on $\boldsymbol{V}_h$, is again constructed as a standard V–cycle operator with symmetric Gauss–Seidel as a smoother. However, the high frequency bubble functions are only present in the finest grid. On all the coarser grids we simply use piecewise linear functions.

This approach seems to be efficient as seen by the condition number estimates in Table 4.2

$h\backslash\varepsilon$	0	0.001	0.01	0.1	0.5	1.0
2^{-2}	2.67	2.66	2.02	1.00	1.10	1.12
2^{-3}	2.79	2.73	1.35	1.05	1.14	1.16
2^{-4}	2.92	2.68	1.11	1.10	1.18	1.19
2^{-5}	2.94	2.22	1.02	1.15	1.20	1.21
2^{-6}	2.95	1.54	1.04	1.18	1.22	1.22
2^{-7}	2.95	1.14	1.11	1.20	1.23	1.23

TABLE 4.2. Condition numbers for $\kappa(\boldsymbol{M}_{\varepsilon,h}(\boldsymbol{I} - \varepsilon^2 \boldsymbol{\Delta}_h))$ obtained from the Mini element.

As expected these condition numbers appears to bounded uniformly with respect to ε and h. Finally, we construct the complete operator $\mathcal{B}_{\varepsilon,h}$ of the form (4.5) and estimate the condition numbers of $\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}$. The results are given in Table 4.3.

These results seem to indicate that the condition numbers $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$ are indeed independent of ε and h. This will be theoretically verified in the next section. □

Example 4.2 We repeat the experiment above, but this time we replace the Mini element by the Taylor–Hood element. Hence, $\boldsymbol{V}_h \subset \boldsymbol{H}_0^1$

$h\backslash\varepsilon$	0	0.001	0.01	0.1	0.5	1.0
2^{-2}	4.06	4.04	3.77	10.00	17.89	18.33
2^{-3}	4.32	4.23	3.59	13.87	19.18	19.43
2^{-4}	4.58	4.21	5.27	16.81	19.66	19.77
2^{-5}	4.65	3.50	8.56	18.53	19.83	19.88
2^{-6}	4.66	3.35	12.49	19.36	19.91	19.92
2^{-7}	4.67	4.52	15.74	19.72	19.93	19.93

TABLE 4.3. Condition numbers for $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$ obtained from the Mini elements.

consists of piecewise quadratics, while, as above, the space Q_h is the space of continuous piecewise linears. The multigrid preconditioner $\boldsymbol{M}_{\varepsilon,h} : \boldsymbol{V}_h \mapsto \boldsymbol{V}_h$ is again a standard V–cycle operator with a symmertic Gauss–Seidel smoother. The estimates for the condition numbers $\kappa(\boldsymbol{M}_{\varepsilon,h}(\boldsymbol{I} - \varepsilon^2\boldsymbol{\Delta}_h))$, given in Table 4.4, clearly indicates a bound independent of ε and h.

$h\backslash\varepsilon$	0	0.001	0.01	0.1	0.5	1.0
2^{-2}	1.10	1.10	1.07	1.05	1.19	1.20
2^{-3}	1.11	1.11	1.03	1.14	1.22	1.22
2^{-4}	1.11	1.10	1.01	1.21	1.24	1.14
2^{-5}	1.11	1.09	1.03	1.23	1.24	1.24
2^{-6}	1.11	1.05	1.12	1.24	1.24	1.24
2^{-7}	1.11	1.02	1.20	1.24	1.24	1.24

TABLE 4.4. Condition numbers for $\kappa(\boldsymbol{M}_{\varepsilon,h}(\boldsymbol{I} - \varepsilon^2\boldsymbol{\Delta}_h))$ obtained from the Taylor–Hood element.

As in the case of the Mini element we use the operator $\boldsymbol{M}_{\varepsilon,h}$, together with the operators N_h and I_h introduced above, to construct the complete operator $\mathcal{B}_{\varepsilon,h}$ of the form (4.5). The estimates for the condition numbers of $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$ are given in Table 4.5.

$h\backslash\varepsilon$	0	0.001	0.01	0.1	0.5	1.0
2^{-2}	5.98	5.99	6.28	11.63	14.77	14.92
2^{-3}	6.03	6.05	6.92	13.42	15.25	15.32
2^{-4}	6.06	6.10	8.41	14.55	15.50	15.53
2^{-5}	6.07	6.23	10.62	15.14	15.59	15.61
2^{-6}	6.08	6.64	12.77	15.42	15.63	15.64
2^{-7}	6.08	7.81	14.18	15.55	15.64	15.65

TABLE 4.5. Condition numbers for $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$ using the Taylor–Hood elements.

Again these condition numbers appears to be independent of ε and h. Similar results are also obtained if the replace the $P_2 - P_1$ element with the corresponding element on rectangles, i.e. the $Q_2 - Q_1$ element. □

5. A theoretical discussion in the discrete case

The purpose of this section is to present a theoretical analysis of the preconditioners studied experimentally above. We assume that Ω is a bounded polygonal domain in $\mathbb{R}^2$ and that $\{\mathcal{T}_h\}$ is a shape regular and quasi–uniform family of triangulations of Ω, where h is the maximum diameter of a triangle in $\mathcal{T}_h$.

Below we shall give a precise analysis of the conditioning of the operator $\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}$ when the spaces $\boldsymbol{V}_h$ and Q_h are given either by the Mini element or the Taylor–Hood element. However, first we will make some remarks in the general case. For this discussion we just assume that $\boldsymbol{V}_h \subset \boldsymbol{H}_0^1$ and $Q_h \subset H^1 \cap L_0^2$ is a pair of finite element spaces.

Let $\mathcal{A}_{\varepsilon,h} : \boldsymbol{V}_h \times Q_h \mapsto \boldsymbol{V}_h \times Q_h$ be defined by (4.4) and let $\mathcal{B}_{\varepsilon,h} : \boldsymbol{V}_h \times Q_h \mapsto \boldsymbol{V}_h \times Q_h$ be a corresponding L^2 symmetric and positive definite, block diagonal preconditioner on the form (4.5). Our goal is to establish bounds on the spectral condition number, $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$, which are independent of the perturbation parameter ε and the mesh parameter h. To establish this we will use the characterization

$$\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}) = \frac{\sup|\lambda|}{\inf|\lambda|},$$

where the supremum and infimum is taken over the spectrum of $\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}$. The saddle point theory of [6] will be used to obtain an upper bound on $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$. Observe that if $\lambda \in \mathbb{R}$ is an eigenvalue of $\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}$, with corresponding eigenfunction $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_h \times Q_h$, then the equations

$$\begin{aligned} a_\varepsilon(\boldsymbol{u}_h, \boldsymbol{v}) + (p_h, \operatorname{div} \boldsymbol{v}) &= \lambda(\boldsymbol{M}_h^{-1}\boldsymbol{u}_h, \boldsymbol{v}), \\ (\operatorname{div} \boldsymbol{u}_h, q) &= \lambda((\varepsilon^2 I_h + N_h)^{-1} p_h, q), \end{aligned} \tag{5.1}$$

holds for all $\boldsymbol{v} \in \boldsymbol{V}_h, q \in Q_h$.

In all the examples below the operator $\boldsymbol{M}_{\varepsilon,h} : \boldsymbol{V}_h \mapsto \boldsymbol{V}_h$ will be a uniform preconditioner for the corresponding discrete version of the operator $\boldsymbol{I} - \varepsilon^2\boldsymbol{\Delta}$. In other words, the bilinear forms $a_\varepsilon(\cdot,\cdot)$ and $(\boldsymbol{M}_{\varepsilon,h}^{-1}\cdot,\cdot)$ are uniformly spectrally equivalent, i.e. there are constants c_1 and c_2, independent of ε and h, such that

$$c_1 a_\varepsilon(\boldsymbol{v}, \boldsymbol{v}) \le (\boldsymbol{M}_{\varepsilon,h}^{-1}\boldsymbol{v}, \boldsymbol{v}) \le c_2 a_\varepsilon(\boldsymbol{v}, \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_h. \tag{5.2}$$

Furthermore, the operator $I_h : Q_h \mapsto Q_h$ will be spectrally equivalent to the identity operator, and $N_h : Q_h \mapsto Q_h$ is spectrally equivalent to the discrete Laplacian, i.e. there are constants c_3 and c_4, independent of h, such that

$$c_3\|q\|_1^2 \le (N_h^{-1}q, q) \le c_4\|q\|_1^2 \qquad \forall q \in Q_h. \tag{5.3}$$

As we observed in the experiments discussed in §4 above, these requirements on $\boldsymbol{M}_{\varepsilon,h}$, N_h and I_h were easily fulfilled for the examples we studied there.

Recall that the norm $|q|_\varepsilon \equiv \|q\|_{(H^1\cap L^2_0)+\varepsilon^{-1}L^2_0}$ is characterized by

$$|q|^2_\varepsilon = \inf_{q_1\in H^1\cap L^2_0}[\|\mathbf{grad}\, q_1\|^2_0 + \varepsilon^{-2}\|q-q_1\|^2_0].$$

In fact, the optimal choice is $q_1 = (I-\varepsilon^2\Delta)^{-1}q$, where Neumann boundary conditions are implicitly assumed. For functions in Q_h the corresponding discrete norm is given by

$$|q|_{\varepsilon,h} = \inf_{q_1\in Q_h}[\|\mathbf{grad}\, q_1\|^2_0 + \varepsilon^{-2}\|q-q_1\|^2_0].$$

It is obvious that $|q|_\varepsilon \le |q|_{\varepsilon,h}$ on Q_h. However, due to the quasi–uniformity of the triangulations $\{\mathcal{T}_h\}$, the two norms are equivalent, uniformly in h. To see this note that if $q\in Q_h$ and $q_1 = (I-\varepsilon^2\Delta)^{-1}q$ then

$$|q|^2_{\varepsilon,h} \le \|\mathbf{grad}\, q_{1,h}\|^2_0 + \varepsilon^{-2}\|q-q_{1,h}\|^2_0,$$

where $q_{1,h}\in Q_h$ is the L^2 projection of q_1. However, $\|q-q_{1,h}\|_0 \le \|q-q_1\|_0$ and, by quasi-uniformity, $\|\mathbf{grad}\, q_{1,h}\|_0 \le c_0\|\mathbf{grad}\, q_1\|_0$, where the constant c_0 is independent of h. Hence, we conclude that

$$|q|_{\varepsilon,h} \le c_0|q|_\varepsilon.$$

In addition, (5.3) further implies that $|q|_{\varepsilon,h}$ is equivalent to the norm

$$\inf_{q_1\in Q_h}[(N_h^{-1}q_1,q_1) + \varepsilon^{-2}\|q-q_1\|_0]^{1/2}. \tag{5.4}$$

Finally, from the general properties of sums and intersections of linear spaces, cf. §2 above, it follows that the norm (5.4) is equivalent to the norm $((\varepsilon^2 I + N_h)^{-1}q,q)^{1/2}$.

To summarize the discussion so far we state the following result.

Lemma 5.1. *If $I_h, N_h : Q_h \mapsto Q_h$ are L^2 symmetric operators, such that I_h is spectrally equivalent to the identity and N_h satisfies property (5.3), then the norms $|q|_\varepsilon$, $|q|_{\varepsilon,h}$, and $((\varepsilon^2 I_h + N_h)^{-1}q,q)^{1/2}$ are equivalent, uniformly in ε and h.*

Assume that the finite element spaces $\{\boldsymbol{V}_h\times Q_h\}$ satisfies the uniform Babuska–Brezzi condition

$$\inf_{q\in Q_h}\sup_{\boldsymbol{v}\in\boldsymbol{V}_h}\frac{(q,\operatorname{div}\boldsymbol{v})}{\|\boldsymbol{v}\|_\varepsilon|q|_\varepsilon} \ge \alpha > 0, \tag{5.5}$$

where α is independent of ε and h. By using the theory of [6], cf. Proposition 1.1 of that paper, this condition will imply that $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$ is bounded independently of ε and h. In fact, it is an immediate consequence of this theory, the upper bound (2.8), the property (5.2) of the operator $\boldsymbol{M}_{\varepsilon,h}$, and the norm eqivalence given in Lemma 5.1, that $|\lambda|$, where λ is an eigenvalue of (5.1), is bounded from below and above,

uniformly in ε and h. We can therefore conclude with the following result.

Theorem 5.1. *Assume that $\boldsymbol{V}_h \subset \boldsymbol{H}_0^1$, $Q_h \subset H^1 \cap L_0^2$ and let $\boldsymbol{M}_{\varepsilon,h} : \boldsymbol{V}_h \mapsto \boldsymbol{V}_h$ be a L^2 symmetric, positive definite preconditioner satisfying* (5.2). *Furthermore, assume that the operators N_h and I_h on Q_h are as in Lemma 5.1. If the uniform Babuska–Brezzi condition* (5.5) *holds then the condition numbers $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$ are bounded uniformly in ε and h.*

Hence, for any particular choice of spaces $\{\boldsymbol{V}_h \times Q_h\}$ our main task is to verify the uniform inf–sup condition (5.5).

5.1. **The Mini element.** We recall that the velocity space, $\boldsymbol{V}_h$, consists of linear combinations of continuous piecewise linear vector fields and local cubic bubbles. More precisely, $\boldsymbol{v} \in \boldsymbol{V}_h$ if and only if

$$\boldsymbol{v} = \boldsymbol{v}^1 + \sum_{T \in \mathcal{T}_h} \boldsymbol{c}_T b_T,$$

where $\boldsymbol{v}^1$ is a continuous piecewise linear vector field, $\boldsymbol{c}_T \in \mathbb{R}^2$, and b_T is the scalar cubic bubble function with respect to T, i.e. the unique cubic function vanishing on ∂T and with $\int_T b_T \, dx = 0$. The pressure space Q_h is the standard space of continuous piecewise linear scalar fields.

The uniform Babuska–Brezzi condition (5.5) will be established in this case by constructing an interpolation operator $\boldsymbol{\Pi}_h : \boldsymbol{L}^2 \mapsto \boldsymbol{V}_h$ such that

$$(\operatorname{div} \boldsymbol{\Pi}_h \boldsymbol{v}, q) = (\operatorname{div} \boldsymbol{v}, q) \qquad \forall \boldsymbol{v} \in \boldsymbol{H}_0^1, q \in Q_h, \tag{5.6}$$

and

$$\|\boldsymbol{\Pi}_h \boldsymbol{v}\|_\varepsilon \leq c \, \|\boldsymbol{v}\|_\varepsilon \qquad \forall \boldsymbol{v} \in \boldsymbol{H}_0^1, \tag{5.7}$$

where the constant c is independent of ε and h. The condition (5.5) will then follow from these two properties of the operator $\boldsymbol{\Pi}_h$ and the corresponding continuous Babuska–Brezzi condition (2.9).

In order to define the operator $\boldsymbol{\Pi}_h$ we will utilize the fact that the space $\boldsymbol{V}_h$ can be decomposed into two subspaces, $\boldsymbol{V}_h^b$, consisting of all functions which are identical zero on all edges, i.e. $\boldsymbol{V}_h^b$ is the span of the bubble functions, and $\boldsymbol{V}_h^1$ consisting of continuous piecewise linear vector fields. Let $\boldsymbol{\Pi}_h^b : \boldsymbol{L}^2 \mapsto \boldsymbol{V}_h^b$ be defined by,

$$(\boldsymbol{\Pi}_h^b \boldsymbol{v}, \boldsymbol{z}) = (\boldsymbol{v}, \boldsymbol{z}) \qquad \forall \boldsymbol{z} \in \boldsymbol{Z}_h,$$

where $\boldsymbol{Z}_h$ denotes the space of piecewise constants vector fields. Clearly this uniquely determines $\boldsymbol{\Pi}_h^b$. Furthermore, a scaling argument, utilizing equivalence of norms, shows that the local operators $\boldsymbol{\Pi}_h^b$ are uniformly bounded, with respect to h, in L^2.

The operator $\boldsymbol{\Pi}_h^b$ will satisfy property (5.6) since for all $\boldsymbol{v} \in \boldsymbol{H}_0^1$ and $q \in Q_h$, we have

$$(\operatorname{div} \boldsymbol{\Pi}_h^b \boldsymbol{v}, q) = -(\boldsymbol{\Pi}_h^b \boldsymbol{v}, \mathbf{grad}\, q) = -(\boldsymbol{v}, \mathbf{grad}\, q) = (\operatorname{div} \boldsymbol{v}, q),$$

where we have used that $\mathbf{grad}\, Q_h \subset \boldsymbol{Z}_h$.

The desired operator $\boldsymbol{\Pi}_h : \boldsymbol{V}_h \mapsto \boldsymbol{V}_h$ will be of the form

$$\boldsymbol{\Pi}_h = \boldsymbol{\Pi}_h^b(\boldsymbol{I} - \boldsymbol{R}_h) + \boldsymbol{R}_h,$$

where $\boldsymbol{R}_h : \boldsymbol{L}^2 \mapsto \boldsymbol{V}_h^1$ will be specified below. Note that

$$\boldsymbol{I} - \boldsymbol{\Pi}_h = (\boldsymbol{I} - \boldsymbol{\Pi}_h^b)(\boldsymbol{I} - \boldsymbol{R}_h),$$

and therefore

$$(\operatorname{div}(\boldsymbol{I} - \boldsymbol{\Pi}_h)\boldsymbol{v}, q) = (\operatorname{div}(\boldsymbol{I} - \boldsymbol{\Pi}_h^b)(\boldsymbol{I} - \boldsymbol{R}_h)\boldsymbol{v}, q) = 0$$

for all $q \in Q_h$. Hence, the operator $\boldsymbol{\Pi}_h$ satisfies (5.6).

We take $\boldsymbol{R}_h$ to be the Clement interpolant onto piecewise linear vector fields, cf. [8]. Hence, in particular, $\boldsymbol{R}_h$ satisfies

$$\|(\boldsymbol{I} - \boldsymbol{R}_h)\boldsymbol{v}\|_j \leq c h^{k-j} \|\boldsymbol{v}\|_k, \qquad 0 \leq j \leq k \leq 1, \tag{5.8}$$

where the constant c is independent of h and $\boldsymbol{v}$. Since $\boldsymbol{\Pi}_h^b$ is uniformly L^2–bounded we derive, using (5.8) and a standard inverse estimate, that for $j = 0, 1$

$$\begin{aligned}
\|\boldsymbol{\Pi}_h \boldsymbol{v}\|_j &\leq \|\boldsymbol{\Pi}_h^b(\boldsymbol{I} - \boldsymbol{R}_h)\boldsymbol{v}\|_j + \|\boldsymbol{R}_h \boldsymbol{v}\|_j \\
&\leq c(h^{-j}\|\boldsymbol{\Pi}_h^b(\boldsymbol{I} - \boldsymbol{R}_h)\boldsymbol{v}\|_0 + \|\boldsymbol{v}\|_j) \\
&\leq c(h^{-j}\|(\boldsymbol{I} - \boldsymbol{R}_h)\boldsymbol{v}\|_0 + \|\boldsymbol{v}\|_j) \\
&\leq c\|\boldsymbol{v}\|_j.
\end{aligned}$$

This implies (5.7).

The uniform inf–sup condition (5.5) has therefore been established. As explained above, this implies that $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$ is bounded, uniformly with respect to ε and h. The analysis given here is therefore in agreement with the observations done in Example 4.1.

5.2. **The Taylor–Hood element.** We recall that the velocity space, $\boldsymbol{V}_h$, consists of continuous piecewise quadratic vector fields, while, as above, the discrete pressures in Q_h are continuous and piecewise linear. In this case we shall use a slightly different strategy than above to establish the uniform Babuska–Brezzi condition (5.5). The argument we will present resembles the continuous argument given in the proof of Lemma 3.1.

We first recall that it is well–known that the Taylor–Hood element satisfies (4.2), cf. [20] or Chapter 6 of [7]. On the other hand, for $\varepsilon = 0$ (5.5) takes the form

$$\inf_{q \in Q_h} \sup_{\boldsymbol{v} \in \boldsymbol{V}_h} \frac{(q, \operatorname{div} \boldsymbol{v})}{\|\boldsymbol{v}\|_0 \|\mathbf{grad}\, q\|_0} \geq \alpha_0 > 0. \tag{5.9}$$

In fact, this property, which is often referred to as the weak inf–sup condition for the Taylor–Hood element, was established in [4]. We can therefore conclude that in the two extreme cases, $\varepsilon = 0$ and $\varepsilon = 1$, the inf–sup condition is satisfied. Furthermore, these properties imply that that the weakly defined gradient, $\mathbf{grad}_h : Q_h \mapsto \boldsymbol{V}_h$ given by

$$(\boldsymbol{v}, \mathbf{grad}_h\, q) = -(\operatorname{div} \boldsymbol{v}, q) \quad \forall \boldsymbol{v} \in \boldsymbol{V}_h, q \in Q_h,$$

is one–one. Let $\boldsymbol{G}_h \subset \boldsymbol{V}_h$ be defined as $\mathbf{grad}_h(Q_h)$. As a consequence of the two estimates (4.2) and (5.9) we obtain that for all $\boldsymbol{u} \in \boldsymbol{G}_h$

$$\|\,\mathbf{grad}_h^{-1}\,\boldsymbol{u}\|_0 \leq \alpha_1^{-1}\|\boldsymbol{u}\|_{-1,h} \equiv \alpha_1^{-1} \sup_{\boldsymbol{v}\in\boldsymbol{V}_h} \frac{(\boldsymbol{u},\boldsymbol{v})}{\|\boldsymbol{v}\|_1},$$

and

$$\|\,\mathbf{grad}_h^{-1}\,\boldsymbol{u}\|_1 \leq \alpha_0^{-1}\|\boldsymbol{u}\|_0.$$

From (2.1), (2.2) and Lemma 5.1 it therefore follows that

$$|\,\mathbf{grad}_h^{-1}\,\boldsymbol{u}|_\varepsilon \leq \max(\alpha_1^{-1}, \alpha_0^{-1}) \sup_{\boldsymbol{v}\in\boldsymbol{V}_h} \frac{(\boldsymbol{u},\boldsymbol{v})}{\|\boldsymbol{v}\|_\varepsilon}, \quad \forall \boldsymbol{u} \in \boldsymbol{G}_h,$$

and by letting $\boldsymbol{u} = \mathbf{grad}_h\, q$ this implies (5.5). Hence, we have completed our theoretical explanation of the observations done in Example 4.2.

6. Discontinuous approximation of the pressure

In the analysis above we have strongly utilized the fact that we have continuous discrete pressures, i.e. the pressure space Q_h is a subspace of H^1. In fact, the bilinear form associated with the preconditioner $N_h : Q_h \mapsto Q_h$ is required to be equivalent to the H^1 inner product on Q_h. However, several common Stokes elements use discontinuous piecewise constant pressures. The construction of uniform preconditioners for the coefficient operator $\mathcal{A}_{\varepsilon,h}$ in these cases will be discussed in this section. For simplicity, we restrict the discussion to the well known $P_2 - P_0$ element, but we have also seen similar behavior as we will observe below in numerical experiments with other elements like the nonconforming Crouzeix-Raviart element [9].

Let $Q_h \subset L_0^2$ be the space of discontinuous constant functions with respect to a triangulation $\mathcal{T}_h$, where, as above, $\{\mathcal{T}_h\}$ is a shape regular and quasi–uniform family of triangulations of Ω. As for the Taylor–Hood element discussed above, the velocity space $\boldsymbol{V}_h \subset \boldsymbol{H}_0^1$ consists of continuous, piecewise quadratic vector fields. A weakly defined gradient operator $\mathbf{grad}_h : Q_h \mapsto \boldsymbol{V}_h$ is given by

$$(\mathbf{grad}_h\, q, \boldsymbol{v}) = -(q, \operatorname{div} \boldsymbol{v}) \quad \forall \boldsymbol{v} \in \boldsymbol{V}_h,\ q \in Q_h.$$

A discrete analog of the norm on $(H^1 \cap L_0^2) + \varepsilon^{-1} \cdot L_0^2$ can now be defined on Q_h as

$$|q|_{\varepsilon,h} = \inf_{\substack{q=q_1+q_2 \\ q_1,q_2\in Q_h}} (\|\,\mathbf{grad}_h\, q_1\|_0^2 + \varepsilon^{-2}\|q_2\|_0^2)^{1/2}.$$

The appropriate uniform inf–sup condition we are seeking takes the form

$$\inf_{q\in Q_h}\sup_{\boldsymbol{v}\in \boldsymbol{V}_h}\frac{(q,\operatorname{div}\boldsymbol{v})}{\|\boldsymbol{v}\|_\varepsilon |q|_{\varepsilon,h}}\geq \alpha>0, \tag{6.1}$$

for a suitable α independent of ε and h.

For the P_2-P_0 element the standard inf–sup condition (4.2) is well–known, cf. for example [7, Chapter 6.4]. In particular, this implies that $\mathbf{grad}_h : Q_h \mapsto \boldsymbol{V}_h$ is one–one. Furthermore, (4.2) implies the discrete Poincaré inequality

$$\|q\|_0 \leq \alpha_1^{-1}\|\mathbf{grad}_h\, q\|_0 \quad \forall q\in Q_h. \tag{6.2}$$

It is also straightforward to check that the norms $\|q\|_0$ and $|q|_{1,h}$ are equivalent on Q_h, uniformly in h. To see this note that $|q|_{1,h}\leq \|q\|_0$ is a direct consequence of the definition of $|q|_{1,h}$. On the other hand, if q_1 is chosen as the minimizer in the definition of $|q|_{1,h}$ we have

$$|q|_{1,h}^2 = \|\mathbf{grad}_h\, q_1\|_0^2 + \|q-q_1\|_0^2$$

and

$$(\mathbf{grad}_h\, q_1, \mathbf{grad}_h\, r) + (q_1, r) = (q,r) \quad \forall r\in Q_h.$$

From (6.2) we then obtain

$$\begin{aligned}\|q\|_0^2 &= (q, q-q_1) + (q,q_1)\\ &= (q,q-q_1) + \|q_1\|_0^2 + \|\mathbf{grad}_h\, q_1\|_0^2\\ &\leq \frac{1}{2}\|q\|_0^2 + (1+\alpha_1^{-2})|q|_{1,h}^2.\end{aligned}$$

Therefore, $\|q\|_0$ and $|q|_{1,h}$ are uniformly equivalent on Q_h, and (6.1) for $\varepsilon = 1$ follows from (4.2).

When $\varepsilon = 0$ (6.1) holds with constant $\alpha = 1$. This is a direct consequence of the definitions of $\mathbf{grad}_h$ and $|\cdot|_{\varepsilon,h}$. The uniform inf-sup condition (6.1) therefore follows for all $\varepsilon\in[0,1]$ by an argument completely analog to the one given in §5.2 above.

Having established a uniform inf–sup condition we are again in position to construct a uniform, block diagonal preconditioner for the operator $\mathcal{A}_{\varepsilon,h}$ using similar arguments as above. Consider an operator of the form

$$\mathcal{B}_{\varepsilon,h} = \begin{pmatrix} \boldsymbol{M}_{\varepsilon,h} & 0\\ 0 & \varepsilon^2 I_h + N_h\end{pmatrix} \tag{6.3}$$

mapping $\boldsymbol{V}_h\times Q_h$ into itself. In fact, in the present case, where the pressure space Q_h is discontinuous, we simply take I_h to be the identity operator. Furthermore, as in §5.2 above we have to our disposal a uniform preconditioner $\boldsymbol{M}_{\varepsilon,h}:\boldsymbol{V}_h\mapsto\boldsymbol{V}_h$ for the discrete version of the differential operator $\boldsymbol{I}-\varepsilon^2\boldsymbol{\Delta}$ on the piecewise quadratic space $\boldsymbol{V}_h$.

It only remains to specify the symmetric and positive definite preconditioner $N_h : Q_h\mapsto Q_h$. Assume that we can construct N_h such

that the norms $\|\mathbf{grad}_h\, q\|_0$ and $(N_h^{-1}q, q)^{1/2}$ are equivalent, uniformly in h, on Q_h. As in §5 above it then follows from the uniform inf–sup condition (6.1), some obvious upper bounds, and the theory of [6] that the condition number $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$ is bounded independently of ε and h.

A potential difficulty for the construction of the preconditioner N_h is that the operator $\mathbf{grad}_h : Q_h \mapsto \boldsymbol{V}_h$ is nonlocal. However, there is a local norm, $\|q\|_{1,h}$, which is equivalent to $\|\mathbf{grad}_h\, q\|_0$, and the structure of this local norm can more easily be used to construct the preconditioner N_h. Define a new norm on Q_h by

$$\|q\|_{1,h}^2 = \sum_{e\in\mathcal{E}_h} [q]_e^2, \tag{6.4}$$

where $\mathcal{E}_h$ is the set of interior edges of $\mathcal{T}_h$ and $[q]_e$ is the jump of q on the edge e.

Lemma 6.1. *The norms $\|\mathbf{grad}_h\, q\|_0$ and $\|q\|_{1,h}$ are equivalent on Q_h, uniformly in h.*

Proof. The standard degrees of freedom for the space $\boldsymbol{V}_h$ is the function values at each vertex and the zero order moments on each edge. As a consequence of equivalence of norms we therefore obtain, from a standard scaling argument, that

$$\sum_{e\in\mathcal{E}_h} (\int_e \boldsymbol{v}\cdot\boldsymbol{n}_e\, d\rho)^2 \le c\|\boldsymbol{v}\|_0^2 \qquad \forall \boldsymbol{v}\in\boldsymbol{V}_h,$$

where c is a constant independent of h. Here $\boldsymbol{n}_e$ is a unit normal vector on the edge e and ρ is the arc length along e. As a consequence, for any $\boldsymbol{v}\in\boldsymbol{V}_h$ and $q\in Q_h$ we have

$$\begin{aligned}(\mathbf{grad}_h\, q, \boldsymbol{v}) &= -\sum_{T\in\mathcal{T}_h}\int_T q\,\mathrm{div}\,\boldsymbol{v}\, dx = -\sum_{e\in\mathcal{E}_h}[q]_e\int_e \boldsymbol{v}\cdot\boldsymbol{n}_e\, d\rho \\ &\le \|q\|_{1,h}(\sum_{e\in\mathcal{E}_h}(\int_e \boldsymbol{v}\cdot\boldsymbol{n}_e\, d\rho)^2)^{1/2} \le c\|q\|_{1,h}\|\boldsymbol{v}\|_0,\end{aligned}$$

and we can therefore conclude that

$$\|\mathbf{grad}_h\, q\|_0 \le c\|q\|_{1,h} \quad \forall q\in Q_h.$$

To establish the opposite bound let $q\in Q_h$ be given and define $\hat{\boldsymbol{v}}\in\boldsymbol{V}_h$ such that $\hat{\boldsymbol{v}}$ is zero at each vertex, the tangential component of the zero order moments are zero on each edge, and

$$\int_e \hat{\boldsymbol{v}}\cdot\boldsymbol{n}_e\, d\rho = -[q]_e$$

for all $e\in\mathcal{E}_h$. Again, equivalence of norms implies that

$$\|\hat{\boldsymbol{v}}\|_0^2 \le c\sum_{e\in\mathcal{E}_h}(\int_e \hat{\boldsymbol{v}}\cdot\boldsymbol{n}_e\, d\rho)^2 = c\sum_{e\in\mathcal{E}_h}[q]_e^2,$$

where the constant c is independent of h. Hence,

$$(\mathbf{grad}_h\, q, \hat{\boldsymbol{v}}) = -\sum_{e\in\mathcal{E}_h}[q]_e \int_e \hat{\boldsymbol{v}}\cdot\boldsymbol{n}_e\, d\rho \;=\; \sum_{e\in\mathcal{E}_h}[q]_e^2 \geq c^{-1}\|q\|_{1,h}\|\hat{\boldsymbol{v}}\|_0,$$

which implies that

$$\|q\|_{1,h} \leq c\|\,\mathbf{grad}_h\, q\|_0.$$

This complets the proof. □

6.1. **Numerical experiments.** Our purpose is to repeat the experiments we did it Examples 4.1 and 4.2, but this time we use the $P_2 - P_0$ element for the discretization. In order to complete the description of the preconditioner $\mathcal{B}_{\varepsilon,h}$ given by (6.3) we have to make a proper choice for the preconditioner N_h on Q_h. However, due to Lemma 6.1 the operator N_h can be constructed as preconditioner for the "finite difference Laplacian" obtain from the bilinear form associated the norm $\|\cdot\|_{1,h}$, cf. (6.4). This can be done in many ways. Here we shall adopt the auxiliary space technique of Xu [21], where the auxiliary space consists of piecewise linear functions. The advantage with this approach is that N_h is essentially constructed from the corresponding preconditioner introduced in §4.2 above. In Examples 4.1 and 4.2, the subspace of $H^1 \cap L_0^2$ consisting of continuous piecewise linear functions with respect to the triangulation $\mathcal{T}_h$ was denoted Q_h, but here, where Q_h already denotes the space of discontinuous constants, we will refer to this space as S_h.

Let $P_h : S_h \mapsto Q_h$ be the L^2 projection, and $P_h^* : Q_h \mapsto S_h$ the adjoint operator with respect to the L^2 inner product. The preconditioner N_h we shall use will be of the form

$$N_h = \tau h^2 I + P_h N_h^S P_h^*, \tag{6.5}$$

where $N_h^S : S_h \mapsto S_h$ is the standard V–cycle multigrid preconditioner for the discrete Laplacian on S_h, described in §4.2 above, while $\tau > 0$ is a suitable scaling constant. In the experiments below $\tau = 0.15$. The preconditioner N_h is computationally feasible since the L^2 projection P_h is local.

First, we check the efficiency of the preconditioner N_h by computing the condition numbers of $N_h(-\Delta_h)$, where $\Delta_h : Q_h \mapsto Q_h$ is given by

$$(-\Delta_h p, q) = \sum_{e\in\mathcal{E}_h}[p]_e[q]_e \quad \forall p, q \in Q_h.$$

The results, which are given in Table 6.1, clearly indicate the $\kappa(N_h(-\Delta_h))$ is bounded independently of h. In fact, a theoretical verification of this will be given in §6.2 below.

h	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
$\kappa(N_h(-\Delta_h))$	3.07	3.13	3.16	3.18	3.17	3.18

TABLE 6.1. Condition numbers for the operators $N_h(-\Delta_h)$.

Having verified, at least experimentally, that N_h is a uniform preconditioner for $-\Delta_h$ we should expect that the operator $\mathcal{B}_{\varepsilon,h}$, given by (6.3), is a uniform preconditioner for the operator $\mathcal{A}_{\varepsilon,h}$.

The observed condition numbers of $\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h}$, for $\varepsilon \in [0,1]$ and decreasing values of h, are given in Table 6.2. In complete agreement with the prediction of the theory these condition numbers appears to bounded independently of ε and h.

$h\backslash\varepsilon$	0	0.001	0.01	0.1	0.5	1.0
2^{-2}	5.51	5.16	4.45	4.82	6.96	7.06
2^{-3}	4.96	4.95	4.48	5.56	7.85	7.79
2^{-4}	5.69	5.51	4.35	6.38	8.31	8.35
2^{-5}	5.22	5.07	4.46	7.12	8.72	8.74
2^{-6}	5.23	4.77	5.09	7.74	9.02	9.05
2^{-7}	5.28	4.30	5.93	8.27	9.24	9.28

TABLE 6.2. Condition numbers for $\kappa(\mathcal{B}_{\varepsilon,h}\mathcal{A}_{\varepsilon,h})$ using the $P_2 - P_0$ element.

6.2. **Analysis of the preconditioner.** In order to complete the analysis of the preconditioner (6.3) we will show that the operator N_h, given by (6.5), is a uniform preconditioner with respect to h for the discrete Laplacian Δ_h on Q_h. More precisely, we will show that the norms

$$\|q\|_{1,h} \equiv (-\Delta_h q, q)^{1/2} \quad \text{and} \quad (N_h \Delta_h q, \Delta_h q)^{1/2} \tag{6.6}$$

are uniformly equivalent on Q_h.

In order to apply the theory of [21] we need to establish that the projection P_h is stable and accurate in the sense that

$$\|P_h s\|_{1,h} \leq c \|\mathbf{grad}\, s\|_0 \quad \forall s \in S_h, \tag{6.7}$$

and

$$\|(I - P_h)s\|_0 \leq ch \|\mathbf{grad}\, s\|_0 \quad \forall s \in S_h. \tag{6.8}$$

Here, the constant c is independent of h. Furthermore, we need to construct an operator $R_h : Q_h \to S_h$ which also is stable an accurate, i.e.

$$\|\mathbf{grad}\, R_h q\|_0 \leq c\|q\|_{1,h} \quad \forall q \in Q_h, \tag{6.9}$$

and

$$\|(I - R_h)q\|_0 \leq ch\|q\|_{1,h} \quad \forall q \in Q_h, \tag{6.10}$$

where again c is independent of h. The operator R_h is only needed for the analysis. It is a straightforward consequence of these four estimates and the theory given in [21], cf. Theorems 2.1 and 2.2 of that paper, to conclude that the two norms given in (6.6) are uniformly equivalent. Hence, the remaining task is to establish the estimates (6.7)–(6.10).

In fact, since P_h is local and preserves piecewise constants, the estimate in (6.8) holds for any $s \in L_0^2 \cap H^1$, and hence on S_h. In order to show (6.7) let us first denote by Z_h the set of piecewise linear functions with respect to $\mathcal{T}_h$, where no continuity constraints are imposed. Hence, $Q_h, S_h \subset Z_h$. Recall that a linear function on a triangle is uniquely determined by the zero order moments with respect to each edge. Therefore, as a consequence of a scaling argument, it follows that the two norms

$$\|z\|_0 \quad \text{and} \quad h(\sum_e (\int_e z_- \, d\rho)^2 + (\int_e z_+ \, d\rho)^2)^{1/2}$$

are equivalent on Z_h. Here z_- and z_+ denote the two restriction of z to e obtained from the two triangles meeting e, and the sum is taken over all edges of the triangulation, with an obvious modification if e is a boundary edge. Using this norm equivalence on Z_h we have for any $s \in S_h$,

$$\begin{aligned} \|P_h s\|_{1,h}^2 = \sum_{e \in \mathcal{E}_h} [P_h s]_e^2 &= \sum_{e \in \mathcal{E}_h} (|e|^{-1} \int_e [P_h s - s]_e \, d\rho)^2 \\ &\leq c h^{-2} \|P_h s - s\|_0^2. \end{aligned}$$

Here, $|e|$ denotes the length of e. Hence, (6.7) follows from (6.8).

In order to show (6.9) and (6.10) we will introduce an alternative discrete norm on Z_h. Since a linear function on a triangle is uniquely determined by its values at each vertex it follows that the two norms

$$\|z\|_0 \quad \text{and} \quad h(\sum_{x \in \mathcal{N}_h} \sum_{T \in \mathcal{T}_h(x)} (z_T(x))^2)^{1/2}$$

are equivalent on Z_h. Here, $z_T(x)$ denotes the value of z at the vertex x on the triangle T, $\mathcal{N}_h$ denotes the set of vertices of the triangulation $\mathcal{T}_h$, and $\mathcal{T}_h(x)$ denotes the set of triangles in $\mathcal{T}_h$ meeting the vertex x. Furthermore, note that the shape–regularity of the triangulation $\mathcal{T}_h$ implies that the number of triangles in $\mathcal{T}_h(x)$ is bounded. Therefore, the equivalence above will still hold if we replace the discrete l^2 norm of the values of z at the vertex x, $\{z_T(x)\}$, by any other norm. In particular, the two norms

$$\|z\|_0 \quad \text{and} \quad h(\sum_{x \in \mathcal{N}_h} (\bar{z}(x)^2 + \sum_{e \in \mathcal{E}_h(x)} [z(x)]_e^2))^{1/2} \tag{6.11}$$

are equivalent on Z_h, where $\mathcal{E}_h(x)$ is the set of interior edges meeting x and $\bar{z}(x)$ is the arithmetic mean of z at x, i.e.

$$\bar{z}(x) = |\mathcal{T}_h(x)|^{-1} \sum_{\mathcal{T}_h(x)} z_T(x).$$

Here $|\mathcal{T}_h(x)|$ denotes the number of triangles in $\mathcal{T}_h(x)$.

Let $\hat{S}_h = S_h \oplus \mathbb{R}$, i.e. $\hat{S}_h$ is the space of continuous piecewise linear functions with no mean value constraint. Then $\hat{S}_h \subset Z_h$ and any element of $\hat{S}_h$ is uniquely determined by its values at each vertex. Define an operator $\hat{R}_h : Q_h \mapsto \hat{S}_h$ by $\hat{R}_h q(x) = \bar{q}(x)$ for all vertices x, i.e. the value of $\hat{R}_h q$ at x is the corresponding mean value of q at x. Hence, $(I - \hat{R}_h)q \in Z_h$ with mean value zero at each vertex, and therefore (6.11) implies that

$$\|(I - \hat{R}_h)q\|_0 \le ch\Big(\sum_{x\in\mathcal{N}_h} \sum_{e\in\mathcal{E}_h(x)} [q(x)]_e^2\Big)^{1/2} = \sqrt{2}ch\|q\|_{1,h} \tag{6.12}$$

for all $q \in Q_h$, where the constant c is independent of h. The desired operator $R_h : Q_h \mapsto S_h$ is defined by

$$R_h q = \hat{R}_h q - |\Omega|^{-1} \int_\Omega \hat{R}_h q \, dx.$$

The estimate (6.10) is an immediate consequence of the estimate (6.12).

Finally, (6.9) essentially follows from (6.10) and an inverse inequality. We have,

$$\begin{aligned} \|\operatorname{\mathbf{grad}} R_h q\|_0^2 = \sum_{T\in\mathcal{T}_h} \|\operatorname{\mathbf{grad}} R_h q\|_{0,T}^2 &= \sum_{T\in\mathcal{T}_h} \|\operatorname{\mathbf{grad}}(R_h q - q)\|_{0,T}^2 \\ &\le ch^{-2}\|R_h q - q\|_0^2, \end{aligned}$$

and hence (6.10) implies (6.9). This completes the analysis of the preconditioner (6.3).

References

[1] D.N. Arnold, F. Brezzi and M. Fortin, A stable finite element method for the Stokes equations, Calcolo 21 (1984), pp. 337–344.

[2] D.N. Arnold, R.S. Falk and R. Winther, Preconditioning discrete approximations of the Reissner–Mindlin plate model, M^2AN 31 (1997), pp 517–557.

[3] J.H. Bramble and J.E. Pasciak, Iterative techniques for time dependent Stokes problem, Comput. Math. Appl. 33 (1997), pp. 13–30.

[4] M. Bercovier and O. Pironneau, Error estimates for finite element method solution of the Stokes problem in primitive variables, Numer. Math. 33 (1979), pp. 211–224.

[5] J. Bergh and J. Löfström, *Interpolation spaces*, Springer Verlag, 1976.

[6] F. Brezzi, On the existence, uniqueness and approximation of saddle–point problems arising from Lagrangian multipliers, RAIRO Anal. Numér. 8 (1974), pp. 129–151.

[7] F. Brezzi and M. Fortin, *Mixed and hybrid finite element methods*, Springer Verlag, 1991.

[8] P. Clement, Approximation by finite element functions using local regularization, RAIRO Anal. Numér. 9 (1975), pp. 77–84.
[9] M. Crouzeix and P.A. Raviart, Conforming and non–conforming finite element methods for solving the stationary Stokes equations, RAIRO Anal. Numér. 7 (1973), pp. 33–76.
[10] V. Girault and P.-A. Raviart, *Finite element methods for Navier–Stokes equations*, Springer Verlag 1986.
[11] Edward J. Dean and Roland Glowinski, *On Some Finite Element Methods for the Numerical Simulation of Incompressible Viscous Flow*, In Proceedings: Incompressible computational fluid dynamics; Trends and advances, Editors: M. D. Gunzburger and R. A. Nicolaides, Cambridge University Press, 1993.
[12] W. Hackbusch, *Iterative solution of large sparse systems of equations*, Springer Verlag 1994.
[13] E. Haug and R. Winther, A domain embedding preconditioner for the Lagrange multiplier system, Math. Comp. 69 (1999), pp. 65–82.
[14] K.A. Mardal, X.-C. Tai and R. Winther, A robust finite element method for Darcy–Stokes flow, SIAM J. Numer. Anal. 40 (2002), pp. 1605–1631.
[15] J. Nečas, *Equations aux dérivée partielles*, Presses de l'Université de Montréal 1965.
[16] C.C Paige and M.A. Sauders, Solution of sparse indefinite systems of linear equations, SIAM J. Numer. Anal. 12 (1975), pp. 617–629.
[17] O. Pironneau, *The finite element method for fluids*, John Wiley & Sons, 1989.
[18] T. Rusten and R. Winther, A preconditioned iterative method for saddle point problems, SIAM J. Matrix Anal. 13 (1992), pp. 887–904.
[19] S. Turek, *Efficient Solvers for Incompressible Flow Problem*, Springer Verlag 1999.
[20] R. Verfürth, Error estimates for a mixed finite element approximation of the Stokes equation, R.A.I.R.O. Anal. Numer. 18 (1984), pp. 175–182.
[21] J. Xu, The auxiliary space method and optimal multigrid preconditioning techniques forunstructured grids, Computing 56, (1996) pp. 215–235.

Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway
E-mail address: kent-and@ifi.uio.no

Department of Informatics and Department of Mathematics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway
E-mail address: rwinther@ifi.uio.no