Bachelor's Thesis

Implementation of test scenarios for incompressible flow using a divergence–free finite–element approach

Author:	Benjamin Rüth
	Maistraße 54
	80337 München
Field of Studies:	Informatics, Scientific Computing
Reviewer:	1st Univ.–Prof. Dr. Hans–Joachim Bungartz (TUM)
	2nd Dr. rer. nat. Tobias Neckel (TUM)
Supervisor:	Dr. rer. nat. Tobias Neckel (TUM)
Date:	17.10.2014

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Wissenschaftliches Rechnen
Boltzmannstraße 3
85748 Garching bei München

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich diese Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe.

_____

Benjamin Rüth, München, den 17.10.2014

Contents

List of Figures

List of Tables

1. Introduction

1.1. Numerical simulation of flow problems

The simulation of fluids is an important and vivid field of studies in engineering and science. Complex flow scenarios can be found when designing hydraulic devices pumping fluids through pipe networks. But also the human heart pushes fluids through complicated vein systems. In general all these flow phenomena can, dependent on the model and the problem, be divided into two categories: compressible and incompressible flow. Especially the simulation of incompressible flow is a difficult field, because one has to satisfy the incompressibility condition. Violating this incompressibility condition often leads to wrong results and simulations with non–physical behaviour such as increasing energy or momentum.

One very popular way to simulate fluid is the finite volume method (FVM). Here one divides the whole simulation domain into many subdomains (finite volumes) and claims important conservation laws like mass conservation, momentum conservation and energy conservation on each subdomain. This leads to big systems of equations which have to be solved using standard techniques from linear algebra. An important advantage of FVM is that the method is directly defined over the conservation laws and thus conservation of properties, which have to be conserved, has a very high priority.

Another approach is the finite difference method (FDM). First the simulation domain is overlaid by a grid of discrete points. Then the incompressible Navier–Stokes equations, which is the partial differential equation (PDE) describing flow, is approximated by substituting occurring derivatives by difference quotients between these gridpoints. This leads — again — to a big linear equation system. The main advantage of FDM is that the implementation is very easy and straightforward. A good example for an implementation of a FDM algorithm for the solution of fluid problems is given in [4].

In this thesis a finite element method (FEM) is used. In this case one modifies the incompressible Navier–Stokes equations and uses tools from variational calculus and functional analysis in order to find an approximative solution of the flow problem. The solution of the Navier–Stokes equations becomes an optimization problem of finding the approximative solution which is closest to the real solution. The incompressibility condition can be interpreted as a constraint to the optimization problem and therefore the incompressible Navier–Stokes equations can be approximated by a restricted optimization problem,

which can be solved by the method of Lagrange multipliers. The approximative solution is represented by using certain basis function, which are often nodal basis functions, and a discretisation of the simulation domain into a finite number of subdomains or cells. The solution of the problem can be found by calculating the weights of the base functions in the approximative solution, which are leading to the optimal solution. The calculation of the weight again leads to a big linear system of equations.

1.2. Structure of the thesis

The main difference of our FEM to standard FEM approaches is the usage of divergence–free basis functions, which satisfy the incompressibility condition a priori. The focus of the thesis lies on numerical test scenarios in order to show that divergence–free basis functions on the one hand are able to prevent errors originating from a violation of incompressibility and on the other hand can guarantee correct physical behaviour of the simulation, conserving energy and momentum. These numerical experiments are done using a MATLAB–Tool which has been developed originally at the Chair of Scientific Computing at TUM by Tobias Neckel during the work on his PhD–Thesis [5] and has been extended significantly during our work on this thesis in order to be able to run different test scenarios.

In order to present the concept of the divergence–free FEM approach used for the computation of these test scenarios, in chapter 2 and chapter 3 the basic equations of fluid dynamics are developed and the finite–element–discretization for the incompressible Navier–Stokes equations is explained first. The necessary implementations for the numerical experiments performed in chapter 5 are presented in chapter 4, where the focus lies on the additions to the already existing code *Quickfluid*. In the final chapter 6 a conclusion follows and the important discoveries on FEM using divergence–free basis functions are repeated.

1.3. Acknowledgement

I want to thank my supervisor Dr. Tobias Neckel for his constant support during the work on this thesis. Especially when facing problems he often gave me the right impulses and motivations for handling the problems and finding a solution. I also thank the Chair of Scientific Computing at Technische Universität München, the head of the chair, Prof. Dr. Hans–Joachim Bungartz, and especially Prof. Dr. Michael Baader and Dr. Wolfgang Eckhardt for reading great lectures on the field of scientific computing.

I thank my family for their support during my whole life, especially during the last years at university. I would not have been able to study without their education and financial

support! Finally I also thank my friends and my girlfriend for rereading this thesis, helping me to find errors of any kind and listening to me when talking of the current problems with the thesis — this was surely often a hard task for my friends, because I actually know that mathematics is not everyone's favourite topic.

2. Governing equations

2.1. Conservation laws

When developing an algorithm for the solution of flow problems by using the finite element method, one needs to understand the conservation laws first. These are

- conservation of momentum,
- conservation of mass and
- conservation of energy.

2.1.1. The Reynolds transport theorem

For the development of the conservation laws the Reynolds transport theorem (RTT) will play an important role. If we want to calculate the total time derivative of a time–dependent integral the following equation holds:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega} f\left(\boldsymbol{x},t\right)\ \mathrm{d}x = \int\limits_{\Omega}\frac{\partial f\left(\boldsymbol{x},t\right)}{\partial t}\ \mathrm{d}x + \int\limits_{\Omega}\mathrm{div}\left(f\left(\boldsymbol{x},t\right)\boldsymbol{u}\left(\boldsymbol{x},t\right)\right)\ \mathrm{d}x. \tag{2.1.1}$$

Ω denotes the volume over which we are integrating and $\boldsymbol{u}\left(\boldsymbol{x},t\right)$ the fluid's velocity. $f\left(\boldsymbol{x},t\right)$ describes a generic property of the fluid like for example the density of mass. Thus the RTT is used for calculating the total inflow (or outflow) of mass into (or out of) the domain Ω during a certain time period.

2.1.2. The continuity equation

The conservation of mass is stated by the continuity equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\int\limits_{\Omega}\rho\left(\boldsymbol{x},t\right)\ \mathrm{d}x\right) = 0$$

with $\rho(\boldsymbol{x}, t)$ the density of the fluid. The integral over the density returns the mass in the control volume Ω, which should not change over time. Using (2.1.1) the integral can be rearranged:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\int_{\Omega} \rho(\boldsymbol{x}, t)\ \mathrm{d}x\right) = \int_{\Omega} \frac{\partial \rho(\boldsymbol{x}, t)}{\partial t}\ \mathrm{d}x + \int_{\Omega} \operatorname{div}\left(\rho(\boldsymbol{x}, t)\, \boldsymbol{u}(\boldsymbol{x}, t)\right)\ \mathrm{d}x.$$

Only incompressible fluids are considered. Therefore $\frac{\partial \rho(\boldsymbol{x},t)}{\partial t} = 0$. Furthermore mass conservation has to hold for infinitesimally small sub–volumes of Ω and the density has to be constant due to incompressibility. That leads finally to the continuity equation

$$\boldsymbol{\nabla} \cdot \boldsymbol{u}(\boldsymbol{x}, t) = 0. \tag{2.1.2}$$

(2.1.2) claims that the velocity field of an incompressible flow problem has to be divergence–free.

2.1.3. The momentum equation

The conservation of momentum can be represented by

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\int_{\Omega} \rho(\boldsymbol{x}, t)\, \boldsymbol{u}(\boldsymbol{x}, t)\ \mathrm{d}x\right) = \sum \boldsymbol{F}.$$

The density $\rho(\boldsymbol{x}, t)$ times the velocity $\boldsymbol{u}(\boldsymbol{x}, t)$ of the fluid gives the density of the momentum. By integrating over the control volume Ω the total momentum of the fluid inside the control volume is obtained. This momentum should only change over time by the amount of the applied Forces $\boldsymbol{F}$ due to Newton's second law.
Again (2.1.1) is used in order to rearrange the integral. The calculations are quite complicated and can be found in literature [5]. Here only the final differential form of the momentum equation is presented and interpreted:

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \boldsymbol{\nabla})\, \boldsymbol{u} + \frac{1}{\rho}\boldsymbol{\nabla} p - \frac{\mu}{\rho}\Delta \boldsymbol{u} = \boldsymbol{g}. \tag{2.1.3}$$

The first term $\frac{\partial \boldsymbol{u}}{\partial t}$ is the acceleration of the fluid. As commonly known acceleration is caused by the forces acting on the fluid, which are

- body forces $\boldsymbol{g}$, like for example gravity,
- convective forces $(\boldsymbol{u} \cdot \boldsymbol{\nabla})\, \boldsymbol{u}$,
- diffusive forces $-\frac{\mu}{\rho}\Delta \boldsymbol{u}$, influenced by the dynamic viscosity μ and the density ρ of the fluid, and
- pressure forces $\frac{1}{\rho}\boldsymbol{\nabla} p$.

2.1.4. The energy equation

The energy equation is not necessary for the formulation of the incompressible Navier–Stokes equations, nevertheless conserving energy is an important property in order to obtain correct results. The Lax–Milgram theorem states, that energy has to be dissipated over time for convergence of the solution(see [2]). Therefore the energy equation explains under which conditions convergence can be guaranteed.

The change of kinetic energy over time is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\int\limits_{\Omega}\frac{1}{2}\rho\left|\boldsymbol{u}\right|^2\ \mathrm{d}x\right).$$

Using (2.1.1) and (2.1.2) this gives

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\int\limits_{\Omega}\frac{1}{2}\rho\left|\boldsymbol{u}\right|^2\ \mathrm{d}x\right)=\int\limits_{\Omega}\frac{\partial}{\partial t}\left(\frac{1}{2}\rho\left|\boldsymbol{u}\right|^2\right)+\frac{1}{2}\rho\left|\boldsymbol{u}\right|^2\mathrm{div}\left(\boldsymbol{u}\right)\ \mathrm{d}x=\int\limits_{\Omega}\frac{1}{2}\rho\frac{\partial}{\partial t}\left(\boldsymbol{u}^T\cdot\boldsymbol{u}\right)\ \mathrm{d}x.$$

Calculating the derivative yields

$$\int\limits_{\Omega}\frac{1}{2}\rho\frac{\partial}{\partial t}\left(\boldsymbol{u}^T\cdot\boldsymbol{u}\right)\ \mathrm{d}x=\int\limits_{\Omega}\rho\left(\frac{\partial\boldsymbol{u}}{\partial t}\right)^T\cdot\boldsymbol{u}\ \mathrm{d}x.$$

For $\frac{\partial\boldsymbol{u}}{\partial t}$ (2.1.3) can be plugged which leads to

$$\frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega}\frac{1}{2}\rho\left|\boldsymbol{u}\right|^2\ \mathrm{d}x=-\int\limits_{\Omega}2\mu\left|\boldsymbol{\nabla}^{\boldsymbol{S}}\boldsymbol{u}\right|^2\ \mathrm{d}x,\tag{2.1.4}$$

where

$$\boldsymbol{\nabla}^{\boldsymbol{S}}\boldsymbol{u}=\frac{1}{2}\left(\boldsymbol{\nabla}\boldsymbol{u}+\left(\boldsymbol{\nabla}\boldsymbol{u}\right)^T\right).$$

Thus the energy is dissipated over time. The rate of dissipation correlates with the viscosity of the fluid μ. (2.1.2) has been used several times in the preceding calculations. Without incompressibility the energy equation would look different and this explains, why incompressibility has to be satisfied in order to obtain a convergent solution.

2.2. Incompressible Navier–Stokes equations and weak form

From the conservation laws presented in section 2.1 the fundamental incompressible Navier–Stokes equations as well as the — for FEM necessary — weak form of the incompressible Navier–Stokes equations can be developed.

2.2.1. Incompressible Navier–Stokes equations

For incompressible flow we must satisfy mass conservation (2.1.2) on the one hand and momentum conservation (2.1.3) on the other hand. Composing these two equations leads to the incompressible Navier–Stokes equations :

$$\frac{\partial \boldsymbol{u}}{\partial t} - \nu \Delta \boldsymbol{u} + (\boldsymbol{u} \cdot \boldsymbol{\nabla})\, \boldsymbol{u} + \frac{1}{\rho} \boldsymbol{\nabla} p = \boldsymbol{g} \tag{2.2.1}$$

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0. \tag{2.2.2}$$

(2.2.2) can be interpreted as a constraint on the solution $\boldsymbol{u}$. Solving the incompressible Navier–Stokes equations is therefore a constrained problem[1].

2.2.2. Weak form of the Navier–Stokes equations

For solving the incompressible Navier–Stokes equations using a FEM approach the so called weak form of the incompressible Navier–Stokes equations is needed. The weak form is just a different formulation of the 'strong' form (2.2.1) and (2.2.2). The derivation of the weak form can be found in literature for example in [3] or [5].

The weak form of the incompressible Navier–Stokes equations has the following form:

Find $(\boldsymbol{u}, p) \in U \times P$ such that for all $(\boldsymbol{s}, q) \in S \times Q$ holds:

$$(\dot{\boldsymbol{u}}, \boldsymbol{s})_0 + a(\boldsymbol{u}, \boldsymbol{s}) + b(p, \boldsymbol{s}) = l(\boldsymbol{s}) \tag{2.2.3}$$

$$c(\boldsymbol{u}, q) = 0. \tag{2.2.4}$$

[1] When using FEM this becomes a constrained optimization problem.

Where we are using the following notation:

$$(\dot{\boldsymbol{u}}, \boldsymbol{s})_0 = \int_\Omega \frac{\partial \boldsymbol{u}}{\partial t} \cdot \boldsymbol{s} \, \mathrm{d}x \tag{2.2.5}$$

$$a(\boldsymbol{u}, \boldsymbol{s}) = \int_\Omega (\boldsymbol{u} \cdot \boldsymbol{\nabla}) \boldsymbol{u} \cdot \boldsymbol{s} + \nu \boldsymbol{\nabla} \boldsymbol{u} : \boldsymbol{\nabla} \boldsymbol{s}^T \, \mathrm{d}x \tag{2.2.6}$$

$$b(p, \boldsymbol{s}) = -\frac{1}{\rho} \int_\Omega p (\boldsymbol{\nabla} \cdot \boldsymbol{s}) \, \mathrm{d}x \tag{2.2.7}$$

$$l(\boldsymbol{s}) = \int_\Omega \boldsymbol{g} \cdot \boldsymbol{s} \, \mathrm{d}x + \int_{\Gamma_N} \boldsymbol{f} \cdot \boldsymbol{s} \, \mathrm{d}a \tag{2.2.8}$$

$$c(\boldsymbol{u}, q) = \int_\Omega (\boldsymbol{\nabla} \cdot \boldsymbol{u}) q \, \mathrm{d}x. \tag{2.2.9}$$

One can recognize the terms of the incompressible Navier–Stokes equations in the weak form. The functions $\boldsymbol{u} \in U$ and $p \in P$ are the solution of the equation and the functions $\boldsymbol{s} \in S$ and $q \in Q$ are the so called test functions. U, S, P, Q are infinite–dimensional function spaces.

3. Finite element method

3.1. Discretisation

In this first section the weak form of the Navier–Stokes equations, developed in chapter 2, and basis functions are going to be used in order to formulate a discrete form of the Navier–Stokes equations. This approach of solving partial differential equation (PDE) is called the finite element method (FEM). For more details on the theory of FEM for the incompressible Navier–Stokes equations see [3].

3.1.1. Definition of basis functions and discrete Navier–Stokes equations

When discretizing the computation domain Ω by dividing it into several quadratic cells, n nodes are obtained. Node i has the position $\boldsymbol{x}_i = \begin{pmatrix} x_i & y_i \end{pmatrix}^T$ and $\mathbf{u}(\boldsymbol{x}_i) = \begin{pmatrix} u_i & v_i \end{pmatrix}^T$. The basis functions $\boldsymbol{\Phi}_i^x(\boldsymbol{x})$ and $\boldsymbol{\Phi}_i^y(\boldsymbol{x})$ are defined in the following way:

$$\boldsymbol{\Phi}_i^x(\boldsymbol{x}) = \begin{pmatrix} \phi_i^1(\boldsymbol{x}) \\ \phi_i^2(\boldsymbol{x}) \end{pmatrix}$$
$$\boldsymbol{\Phi}_i^y(\boldsymbol{x}) = \begin{pmatrix} \phi_i^2(\boldsymbol{x}) \\ \phi_i^1(\boldsymbol{x}) \end{pmatrix}$$

Only nodal basis functions are considered, leading to

$$\boldsymbol{\Phi}_i^x(\boldsymbol{x}_j) = \begin{pmatrix} \phi_i^1(\boldsymbol{x}_j) \\ \phi_i^2(\boldsymbol{x}_j) \end{pmatrix} = \begin{pmatrix} \delta_{ij} \\ 0 \end{pmatrix}$$
$$\boldsymbol{\Phi}_i^y(\boldsymbol{x}_j) = \begin{pmatrix} \phi_i^2(\boldsymbol{x}_j) \\ \phi_i^1(\boldsymbol{x}_j) \end{pmatrix} = \begin{pmatrix} 0 \\ \delta_{ij} \end{pmatrix},$$

where δ_{ij} is Kronecker's delta and $\boldsymbol{x}_j$ is the coordinate of any node.

With these basis functions the discrete solution $\hat{\boldsymbol{u}}_h$ of the PDE is defined as:

$$\hat{\boldsymbol{u}}_h(\boldsymbol{x},t) = \sum_{i=1}^{n} u_i(t)\,\boldsymbol{\Phi}_i^x + v_i(t)\,\boldsymbol{\Phi}_i^y. \tag{3.1.1}$$

Using

$$\boldsymbol{u}_h = \Big(\; u_1(t)\,,\;\; \cdots\,,\;\; u_n(t)\,,\;\; v_1(t)\,,\;\; \cdots\,,\;\; v_n(t)\;\Big)^T$$

and

$$\boldsymbol{\Phi} = \Big(\; \boldsymbol{\Phi}_1^x(t)\,,\;\; \cdots\,,\;\; \boldsymbol{\Phi}_n^x(t)\,,\;\; \boldsymbol{\Phi}_1^y(t)\,,\;\; \cdots\,,\;\; \boldsymbol{\Phi}_n^y(t)\;\Big)^T$$

we can also define $\hat{\boldsymbol{u}}_h(\boldsymbol{x},t)$ in the following way:

$$\hat{\boldsymbol{u}}_h(\boldsymbol{x},t) = \sum_{i=1}^{2n} (\boldsymbol{u}_h)_i\,\boldsymbol{\Phi}_i.$$

Therefore our continuous problem (2.2.1) under the constraint (2.2.2) reduces to the following discrete problem which is basically just a system of equations:

Find $\boldsymbol{u}_h$ such that

$$A\dot{\boldsymbol{u}}_h + D\boldsymbol{u}_h + C(\boldsymbol{u}_h)\,\boldsymbol{u}_h - M^T\boldsymbol{p}_h = \boldsymbol{f} \tag{3.1.2}$$

$$M\boldsymbol{u}_h = 0. \tag{3.1.3}$$

The matrices in this equation are the discrete form of the continuous differential operators from the weak form of the incompressible Navier–Stokes equations. Because $\boldsymbol{u}_h$ is a finite vector, this problem can be solved.

3.1.2. Ritz–Galerkin approach

The discrete form ((3.1.2) and (3.1.3)) can be derived from the weak form ((2.2.3) and (2.2.4)) by using the Ritz–Galerkin approach. We move from the infinite–dimensional function spaces used in chapter 2 to the finite–dimensional function spaces

$$U_h \subset U \qquad S_h \subset S \qquad P_h \subset P \qquad Q_h \subset Q.$$

$\{\boldsymbol{\Phi}\}_{i=1...2n}$ is the basis of U_h and S_h[1]. P_h and Q_h are also defined by some similar basis $\{\boldsymbol{\Psi}\}_{I=1...N}$[2]. Therefore it is sufficient to consider the weak form for every one of the $2n$ basis functions $\boldsymbol{\Phi}_i$ out of $\{\boldsymbol{\Phi}\}_{i=1...2n}$ and N basis functions$\boldsymbol{\Psi}_I$ out of $\{\boldsymbol{\Psi}\}_{I=1...N}$ in order to check that the equation holds for every $(\boldsymbol{s}_h, q_h) \in S_h \times Q_h$. Instead of $\boldsymbol{u}$ and p we are using our approximative solution $\hat{\boldsymbol{u}}_h$ for the velocity field and $\hat{p}_h$ for the pressure field[3]. Now the weak form reads

Find $(\hat{\boldsymbol{u}}_h, \hat{p}_h) \in U_h \times P_h$ such that for all $i = 1, ..., 2n,\ I = 1, ..., N$ holds:

$$\left(\dot{\hat{\boldsymbol{u}}}_h, \boldsymbol{\Phi}_i\right)_0 + a\left(\hat{\boldsymbol{u}}_h, \boldsymbol{\Phi}_i\right) + b\left(\hat{p}_h, \boldsymbol{\Phi}_i\right) = l\left(\boldsymbol{\Phi}_i\right) \tag{3.1.4}$$

$$c\left(\hat{\boldsymbol{u}}_h, \boldsymbol{\Psi}_I\right) = 0. \tag{3.1.5}$$

Here the same notation like in (2.2.5) through (2.2.9) is used.

3.1.3. Derivation of the matrices

Having narrowed down to these finite sets of functions[4], the integrals of the weak form can be evaluated and the matrices A, C and D can be set up. As an example the evaluation of $\left(\dot{\hat{\boldsymbol{u}}}_h, \boldsymbol{\Phi}_i\right)_0$ is shown here. The following expression is obtained by starting with (2.2.5), the approximation $\dot{\hat{\boldsymbol{u}}}_h$ of $\dot{\boldsymbol{u}}$ and the basis function $\boldsymbol{\Phi}_i$:

$$\left(\dot{\hat{\boldsymbol{u}}}_h, \boldsymbol{\Phi}_i\right)_0 = \int_\Omega \frac{\partial \hat{\boldsymbol{u}}_h}{\partial t} \cdot \boldsymbol{\Phi}_i \, \mathrm{d}x = \int_\Omega \left[\sum_{j=1}^{2n} \left(\frac{\partial \left(\boldsymbol{u}_h\left(t\right)\right)_j}{\partial t} \boldsymbol{\Phi}_j\right) \cdot \boldsymbol{\Phi}_i\right] \mathrm{d}x.$$

Putting the sum and the derivative outside of the integral, substituting $\frac{\partial (\boldsymbol{u}_h(t))_j}{\partial t}$ with $\left(\dot{\boldsymbol{u}}_h\right)_j$ and calculating the product yields

$$\sum_{j=1}^{2n} \left(\dot{\boldsymbol{u}}_h\right)_j \int_\Omega \boldsymbol{\Phi}_j \cdot \boldsymbol{\Phi}_i \, \mathrm{d}x.$$

[1] See subsection 3.1.1 for the definition of the basis functions building the basis.
[2] The derivation of the basis of the basis functions for the approximation of the pressure $\hat{p}_h$ will not be discussed here. See [5] for details on this topic.
[3] Again the construction of $\hat{p}_h$ can be found in literature and will not be discussed here.
[4] The basis functions for the velocity are described in section 3.3. The basis functions for the pressure are cellwise constant basis functions in this thesis. In general also more complicated basis functions are used.

The integral can now be substituted by

$$A_{ij} = \int_{\Omega} \boldsymbol{\Phi}_j \cdot \boldsymbol{\Phi}_i \, \mathrm{d}x.$$

Leading to

$$\sum_{j=1}^{2n} \left(\dot{\boldsymbol{u}}_h\right)_j \int_{\Omega} \boldsymbol{\Phi}_j \cdot \boldsymbol{\Phi}_i \, \mathrm{d}x = \sum_{j=1}^{2n} \left(\dot{\boldsymbol{u}}_h\right)_j A_{ij}.$$

(2.2.5) has to hold for every $i = 1, ..., 2n$ due to (3.1.4). $2n$ expressions are obtained, which can be written as a matrix–vector–product:

$$\left(\dot{\hat{\boldsymbol{u}}}_h, \boldsymbol{\Phi}_i\right)_0 = \int_{\Omega} \frac{\partial \hat{\boldsymbol{u}}_h}{\partial t} \cdot \boldsymbol{\Phi}_i \, \mathrm{d}x \Leftrightarrow A\dot{\boldsymbol{u}}_h.$$

Similar calculations[5] lead to the representation of the matrices C and D:

$$a\left(\hat{\boldsymbol{u}}_h, \boldsymbol{\Phi}_i\right) = \int_{\Omega} \left(\boldsymbol{u} \cdot \boldsymbol{\nabla}\right) \boldsymbol{u} \cdot \boldsymbol{s} \, \mathrm{d}x + \int_{\Omega} \nu \boldsymbol{\nabla} \boldsymbol{u} : \boldsymbol{\nabla} \boldsymbol{s}^T \, \mathrm{d}x \Leftrightarrow C\left(\boldsymbol{u}_h\right) \boldsymbol{u}_h + D\boldsymbol{u}_h.$$

The Matrix M is developed in an entirely different way using a finite volume discretization: M discretizes the incompressibility constraint on the simulation domain. The discrete continuity equation for one cell can be derived quickly by integrating the already known continuity equation (2.1.2) and using the divergence theorem.

$$0 \stackrel{!}{=} \int_{\Omega} \boldsymbol{\nabla} \cdot \boldsymbol{u}\left(\boldsymbol{x}, t\right) \mathrm{d}\Omega = \oint_{\partial\Omega=\Gamma} \boldsymbol{u}\left(\boldsymbol{x}, t\right) \cdot \, \mathrm{ds}.$$

Only quadratic cells are considered and the velocities on the nodes are given. The velocities are numbered in a Z–numbering (see subsection 4.3.1). One cell is illustrated in Figure 3.1.1.

Using the trapezoidal rule for evaluating the integral over Γ becomes

$$\begin{array}{llll} \displaystyle\int_{\Gamma_1} \boldsymbol{u}\left(\boldsymbol{x}, t\right) \cdot \, \mathrm{ds} & + \displaystyle\int_{\Gamma_2} \boldsymbol{u}\left(\boldsymbol{x}, t\right) \cdot \, \mathrm{ds} & + \displaystyle\int_{\Gamma_3} \boldsymbol{u}\left(\boldsymbol{x}, t\right) \cdot \, \mathrm{ds} & + \displaystyle\int_{\Gamma_4} \boldsymbol{u}\left(\boldsymbol{x}, t\right) \cdot \, \mathrm{ds} = \\ \frac{1}{2}\left(v_1 + v_2\right) & + \frac{1}{2}\left(-u_1 - u_3\right) & + \frac{1}{2}\left(-v_3 - v_4\right) & + \frac{1}{2}\left(u_2 + u_4\right). \end{array}$$

This expression can be rewritten and must be equal to 0 due to incompressibility:

$$\frac{1}{2}\left(-u_1 + u_2 - u_3 + u_4 + v_1 + v_2 - v_3 - v_4\right) = 0. \tag{3.1.6}$$

[5] For a detailed derivation see [1] or [3].

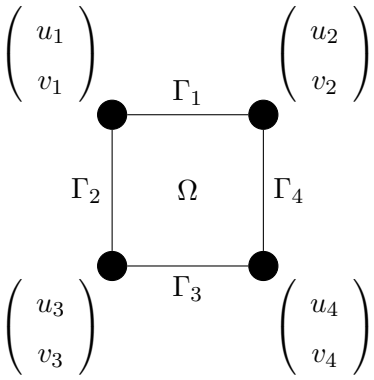

Figure 3.1.1.: one cell with velocities at nodes

Incompressibility is claimed for each cell of the N cells in the domain and this system of N equations can be rewritten in matrix–vector–notation to obtain the representation of the matrix M:

$$M\boldsymbol{u}_h = 0.$$

The correct mapping from the local Z–numbering of the velocities to the global lexicographical numbering is of course also very important and will be discussed later on in section 4.3. A detailed explanation, why $-M^T$ discretizes the gradient can be found in [1]. Having finished the derivation of the matrices A, C, D and M the discrete incompressible Navier–Stokes equations (3.1.2) can be stated. Furthermore the equivalence of (3.1.2) to (3.1.4) is shown.

3.1.4. The pressure Poisson equation

In order to guarantee that the constraint (3.1.3) is fulfilled, the pressure Poisson equation (PPE) is applied. This equation will be derived in the following:

Solving the discrete NSE (3.1.2) for $\dot{\boldsymbol{u}}_h$ gives

$$\dot{\boldsymbol{u}}_h = A^{-1}\left(\boldsymbol{f} - D\boldsymbol{u}_h - C\left(\boldsymbol{u}_h\right)\boldsymbol{u}_h + M^T\boldsymbol{p}_h\right). \tag{3.1.7}$$

Additionally the incompressibility constraint (3.1.3) is differentiated

$$M\dot{\boldsymbol{u}}_h = 0. \tag{3.1.8}$$

This yields

$$0 = MA^{-1}\left(\boldsymbol{f} - D\boldsymbol{u}_h - C\left(\boldsymbol{u}_h\right)\boldsymbol{u}_h + M^T\boldsymbol{p}_h\right). \tag{3.1.9}$$

Solving for $MA^{-1}M^T\boldsymbol{p}_h$ finally defines the PPE:

$$MA^{-1}M^T\boldsymbol{p}_h = MA^{-1}\left(D\boldsymbol{u}_h + C\left(\boldsymbol{u}_h\right)\boldsymbol{u}_h - \boldsymbol{f}\right). \tag{3.1.10}$$

Substituting $MA^{-1}M^T$ with Q and the right hand side of (3.1.10) with $\boldsymbol{b}$ yields

$$Q\boldsymbol{p}_h = \boldsymbol{b}. \tag{3.1.11}$$

That is a linear system of equations which can be solved using standard techniques from linear algebra[6].

3.2. Boundary conditions

In this section the different kinds of boundary conditions (BC) will be presented shortly. Here just the main differences of the boundary conditions and their fields of application will be explained. For a detailed explanation of the theory of boundary conditions refer to [3]. The details on the implementation of these boundary conditions will be discussed in section 4.4.

3.2.1. Free nodes

Free nodes[7] are considered as the standard case and do not demand any special treatment. They do not belong to any boundary and lie inside of the domain, therefore each free node has four neighbouring cells. The velocity $\boldsymbol{u} = \begin{pmatrix} u & v \end{pmatrix}^T$ on this node has two degrees of freedom. Figure 3.2.1 illustrates a free node.

3.2.2. Dirichlet boundary condition, inlet and wall nodes

Dirichlet boundaries Γ_D are either no–slip–walls[8] or inlets[9]. Both components of the velocity $\boldsymbol{u_D} = \begin{pmatrix} u_D & v_D \end{pmatrix}^T$ on nodes lying on a Dirichlet boundary are given. For

[6] In the MATLAB Code used for the numerical test scenarios the same notation is used.
[7] Nodes lying inside the domain, which are not constrained.
[8] walls where the fluid sticks to the wall and has therefore no velocity
[9] domains where the fluid enters the simulation domain with a fixed, defined velocity

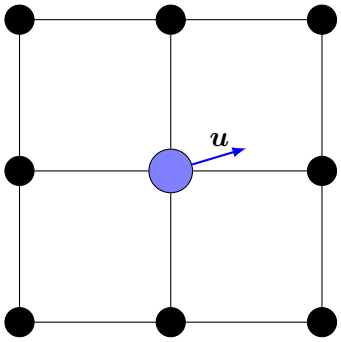

Figure 3.2.1.: a free node with its velocity $\boldsymbol{u}$

inlets at least one component (usually the normal component) is not equal to zero, for no–slip–walls both components are equal to zero due to the no–slip–condition at the wall. Each Dirichlet node has either one or two neighbouring cells. In *Quickfluid* inlet and wall nodes are stored separately for postprocessing issues like the calculation of forces acting on walls. Figure 3.2.2 illustrates both cases of Dirichlet nodes (corner nodes and nodes at one side of the domain).

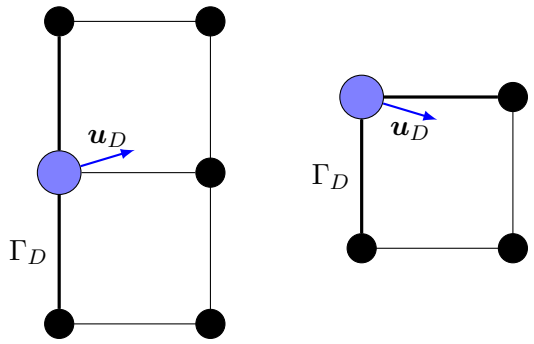

Figure 3.2.2.: a Dirichlet node with its given velocity $\boldsymbol{u}_D$, both cases, corner and side, are shown.

3.2.3. Outlet boundary condition and outlet nodes

Outlet boundaries are Neumann boundaries. Nodes lying on outlet boundaries have two neighbouring cells[10], but both velocity components of $\boldsymbol{u}_N = (\ u \quad v\ )^T$ are free. Due to this reason a special treatment of outlet nodes is needed(see section 4.4). See Figure 3.2.3 for an illustration of this type of boundary condition.

[10] Outlets on corners are not considered in this thesis, because they are not needed for our test scenarios.

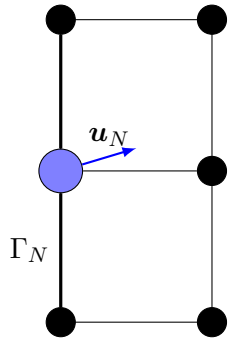

Figure 3.2.3.: an outlet node with its given velocity $\boldsymbol{u}_N$

3.2.4. Slip boundary condition and slip nodes

Slip boundaries are applied at slip–walls or at symmetry axes. Nodes lying on a slip–wall always have two neighbouring cells. At slip–walls the normal component of the velocity is always equal to zero, because the fluid must not penetrate the boundary; the tangential component is free. Corners with slip boundaries are equal to Dirichlet boundaries with both components of the velocity equal to zero. Therefore corners do not have to be considered. For the scenario given in Figure 3.2.4 the velocity at the highlighted node has the value $\boldsymbol{u}_S = (\ 0 \quad v\ )^T$. Due to the complex conditions at slip nodes these nodes have to be treated in a special way(see section 4.4).

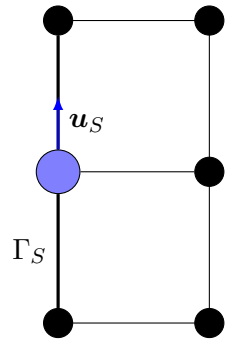

Figure 3.2.4.: a slip node with its given velocity $\boldsymbol{u}_S$

3.3. Different types of basis functions

In this thesis two different types of basis functions for the velocity are compared:

- Pagoda basis functions, which are used in many standard FEM approaches[11] and
- our particular divergence–free basis functions, which guarantee incompressibility a priori.

These types of basis functions will be presented in detail and the analytical formulation of them will be given in this section. The basis functions used for the interpolation of the pressure will not be discussed because they are identical for both approaches presented in this thesis[12]. Finally the both types of basis functions will be compared due to their ability to approximate divergence–free velocity fields.

3.3.1. Pagoda basis functions

Pagoda basis functions are commonly used for FEM algorithms. These functions are defined in the following way:

$$\boldsymbol{\Phi}_i^x(\boldsymbol{x}) = \begin{pmatrix} \phi_i^1(\boldsymbol{x}) \\ \phi_i^2(\boldsymbol{x}) \end{pmatrix} = \begin{pmatrix} \phi(\boldsymbol{x}-\boldsymbol{x}_i) \\ 0 \end{pmatrix}$$
$$\boldsymbol{\Phi}_i^y(\boldsymbol{x}) = \begin{pmatrix} \phi_i^2(\boldsymbol{x}) \\ \phi_i^1(\boldsymbol{x}) \end{pmatrix} = \begin{pmatrix} 0 \\ \phi(\boldsymbol{x}-\boldsymbol{x}_i) \end{pmatrix}.$$

The index i defines the node which is affected by the basis function. $\boldsymbol{x}_i$ holds the coordinates of this node. ϕ is a bilinear, piecewise defined function:

$$\phi(\boldsymbol{x}) = \begin{cases} (h-x)(h-y) & \text{, if } \quad 0 \le x \le h \quad \wedge \quad 0 \le y \le h \\ (h+x)(h-y) & \text{, if } \quad -h \le x \le 0 \quad \wedge \quad 0 \le y \le h \\ (h+x)(h+y) & \text{, if } \quad -h \le x \le 0 \quad \wedge \quad -h \le y \le 0 \\ (h-x)(h+y) & \text{, if } \quad 0 \le x \le h \quad \wedge \quad -h \le y \le 0 \\ 0 & \text{, otherwise} \end{cases}.$$

Therefore the function ϕ only has compact support on a area $[-h,h]^2$, which covers the

[11] In literature this kind of basis function is often used for a Q_1Q_0–element. The subscript denotes the degree of the basis function where the basis function for the velocity is bilinear (degree one) and the basis function for the pressure is constant (degree zero).

[12] For the pressure cell–wise constant basis functions, which are often called Q_0 in literature, are used.

adjacent cells to node i if h is the distance from one node to another. The four different cases, where $\phi \neq 0$, are corresponding to the four quadrants of the coordinate system. A plot of the pagoda basis function for $\boldsymbol{x}_i = \begin{pmatrix} 0 & 0 \end{pmatrix}^T$ is given in Figure 3.3.1. The implementation of the function can be found in subsection A.2.1.

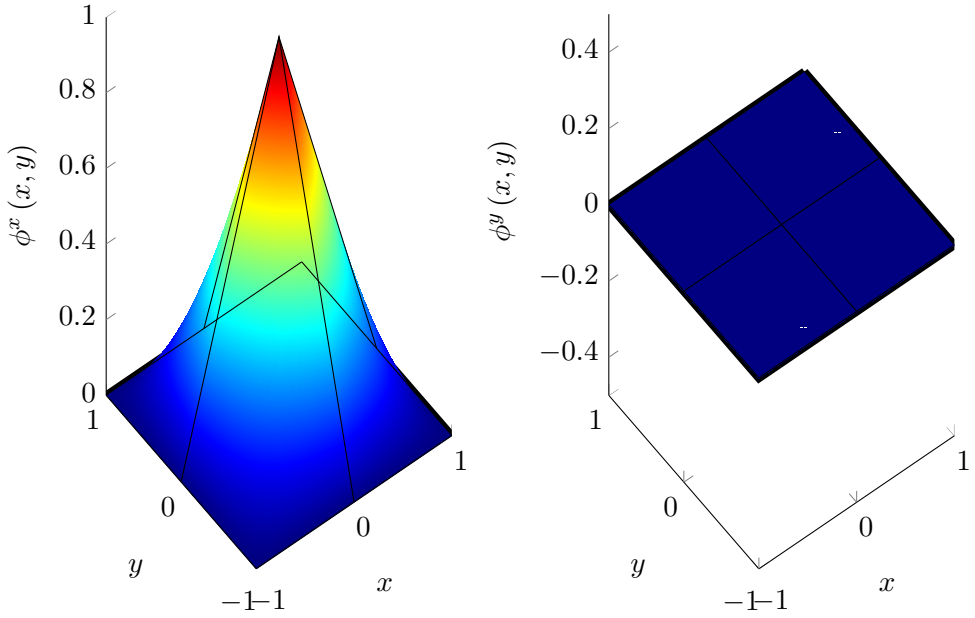

Figure 3.3.1.: pagoda basis function

3.3.2. Divergence–free basis functions

In this thesis basis functions designed for a divergence–free FEM are going to be surveyed. The detailed derivation of this kind of basis functions can be found in [6] or [1]. In the following only the most important aspects of divergence–free basis functions will be considered.

The main idea behind divergence–free basis functions is to satisfy the incompressibility constraint through a special construction of the basis functions. Divergence–free basis functions have the property that as long as the velocity field on a cell fulfils the discrete continuity equation (3.1.6), incompressibility can be satisfied inside the cell as well.

By using the discrete continuity equation and introducing one more node[13] in the center of each cell which is used for satisfying incompressibility, the divergence–free basis function can be as follows

$$\boldsymbol{\Phi}_i^x(\boldsymbol{x}) = \begin{pmatrix} \phi_i^1(\boldsymbol{x}) \\ \phi_i^2(\boldsymbol{x}) \end{pmatrix} = \begin{pmatrix} \tilde{\phi}(\boldsymbol{x}-\boldsymbol{x}_i) \\ \tilde{\varphi}(\boldsymbol{x}-\boldsymbol{x}_i) \end{pmatrix}$$
$$\boldsymbol{\Phi}_i^y(\boldsymbol{x}) = \begin{pmatrix} \phi_i^2(\boldsymbol{x}) \\ \phi_i^1(\boldsymbol{x}) \end{pmatrix} = \begin{pmatrix} \tilde{\varphi}(\boldsymbol{x}-\boldsymbol{x}_i) \\ \tilde{\phi}(\boldsymbol{x}-\boldsymbol{x}_i) \end{pmatrix}.$$

Comparing this function to the pagoda basis function one will see immediately that the divergence–free basis function has two components $\neq 0$, while the second component of the pagoda basis function is equal to zero. The second component of the divergence–free basis function is used for realizing traversing flow, which is important for satisfying incompressibility. This difference will be visualized in subsection 3.3.3. For $(x, y) \in [0, 1]^2$ and $h \neq 0$, $\tilde{\phi}$ and $\tilde{\varphi}$ are defined in the following way:

$$\tilde{\phi}(\boldsymbol{x}) = \begin{cases} 1 - \frac{x}{h} - \frac{y}{2h} & \text{, if Case I} \\ \frac{1}{2} - \frac{x}{2h} & \text{, if Case II} \\ \frac{1}{2} - \frac{y}{2h} & \text{, if Case III} \\ 1 - \frac{x}{2h} - \frac{y}{h} & \text{, if Case IV} \\ 0 & \text{, otherwise outside of support} \end{cases}$$
$$\tilde{\varphi}(\boldsymbol{x}) = \begin{cases} \frac{y}{2h} & \text{, if Case I} \\ \frac{1}{2} - \frac{x}{2h} & \text{, if Case II} \\ \frac{1}{2} - \frac{y}{2h} & \text{, if Case III} \\ \frac{x}{2h} & \text{, if Case IV} \\ 0 & \text{, otherwise outside of support.} \end{cases}.$$

The different cases are defined in the following way:

$$\begin{aligned} \text{Case I: } & x \geq y \wedge x \leq h - y \\ \text{Case II: } & x \geq y \wedge x \geq h - y \\ \text{Case III: } & x \leq y \wedge x \geq h - y \\ \text{Case IV: } & x \leq y \wedge x \leq h - y. \end{aligned}$$

In Figure 3.3.2 the four different cases are illustrated.

This definition of $\tilde{\phi}$ and $\tilde{\varphi}$ can only be applied for the first quadrant. The other rep-

[13] This node is not a additional degree of freedom. It is just introduced for the derivation of the basis functions and the total number of degrees of freedom does not differ for the two types of basis functions.

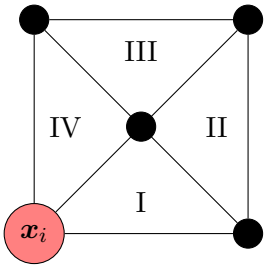

Figure 3.3.2.: The four different cases for the first quadrant. The red node has the coordinates $\boldsymbol{x}_i$.

resentations of $\tilde{\phi}$ and $\tilde{\varphi}$ are obtained using symmetry conditions. A plot of the full divergence–free basis function on the four neighbouring cells of node i is shown in Figure 3.3.3. The implementation of the function can be found in subsection A.2.2.

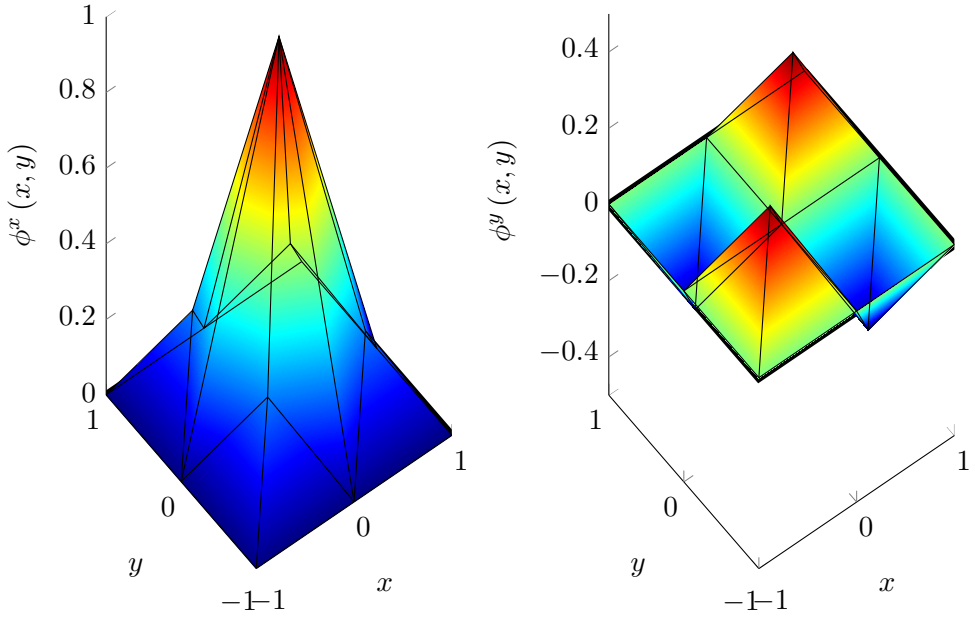

Figure 3.3.3.: divergence–free basis functions

3.3.3. Comparison

In order to understand the benefit of divergence–free basis functions, in the following section a very simple flow field will be considered using both pagoda basis functions and divergence–free basis functions. The discrete field shown in Figure 3.3.4 is divergence–free due to (3.1.6)[14].

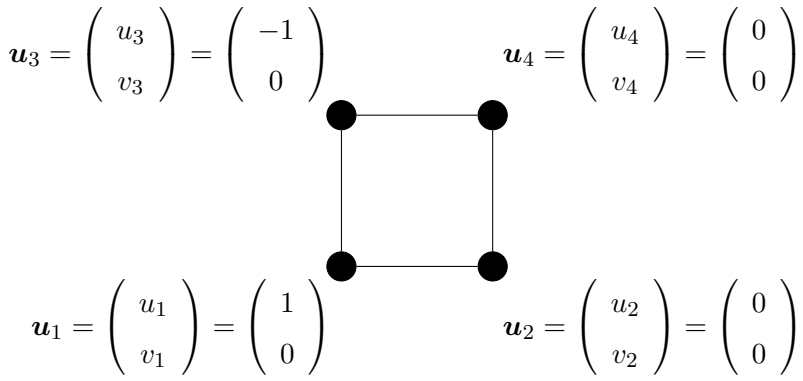

Figure 3.3.4.: one very simple, divergence–free velocity field

First pagoda basis functions will be used for the interpolation of the velocity field on the cell. Due to (3.1.1) the field $\hat{\boldsymbol{u}}_h(\boldsymbol{x}, t)$ is given by the following sum:

$$\begin{aligned}
\hat{\boldsymbol{u}}_h(\boldsymbol{x}, t) &= \sum_{i=1}^{4} u_i(t)\, \boldsymbol{\Phi}_i^x + v_i(t)\, \boldsymbol{\Phi}_i^y = u_1 \boldsymbol{\Phi}_1^x + u_3 \boldsymbol{\Phi}_3^x \\
&= u_1 \begin{pmatrix} \phi(x - x_1, y - y_1) \\ 0 \end{pmatrix} + u_3 \begin{pmatrix} \phi(x - x_3, y - y_3) \\ 0 \end{pmatrix} \\
&= (+1) \begin{pmatrix} \phi(x - 0, y - 0) \\ 0 \end{pmatrix} + (-1) \begin{pmatrix} \phi(x - 0, y - 1) \\ 0 \end{pmatrix}.
\end{aligned}$$

[14] The velocities for this scenario are numbered in a lexicographic way (global numbering), but the discrete incompressibility equation is given in Z–numbering (local numbering). Therefore just putting the velocities into the equation will not work, but a sufficient mapping has to be applied from the local numbering to the global numbering. See chapter 4 for details on this topic.

Only the area $\Omega = [0,1]^2$ is considered. Therefore the right piecewise definitions of $\phi(x,y)$ leads to

$$\hat{\boldsymbol{u}}_h(\boldsymbol{x},t) = 1\begin{pmatrix}(1-x)(1-y)\\0\end{pmatrix} + (-1)\begin{pmatrix}(1-x)(1+(y-1))\\0\end{pmatrix} = \begin{pmatrix}2xy-2y-x+1\\0\end{pmatrix}.$$

Calculating the divergence yields

$$\boldsymbol{\nabla}\cdot\hat{\boldsymbol{u}}_h(\boldsymbol{x},t) = \frac{\partial}{\partial x}(2xy-2y-x+1) + \frac{\partial 0}{\partial y} = 2y-1 \neq 0,$$

which shows that the resulting velocity field is not divergence–free, even if the initially given velocities on the nodes fulfil (3.1.6). A plot of the continuous field using a pagoda basis function and its divergence is shown in Figure 3.3.5.

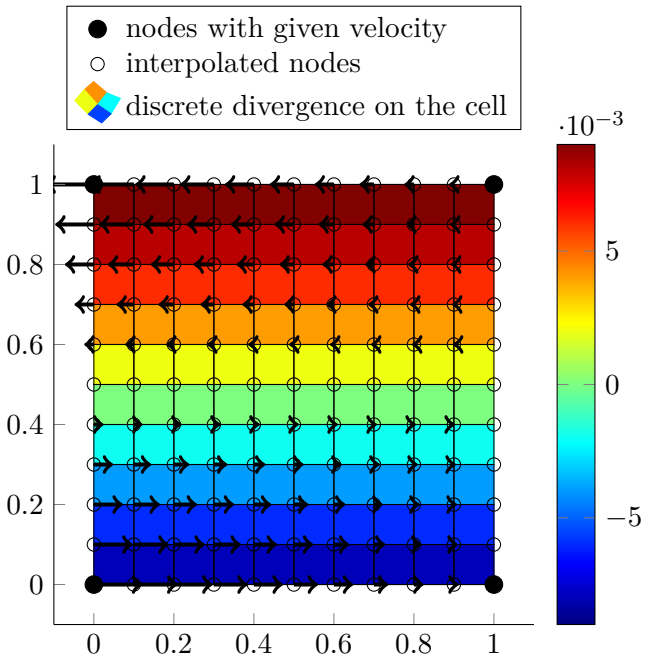

Figure 3.3.5.: interpolating field using pagoda basis functions

The divergence is calculated using (3.1.6) for the cells of the mesh of interpolated nodes. See also subsection 4.6.2 for more information about the calculation of the divergence.

Even if the initial 1×1 grid is refined[15] such that the discrete divergence of the 2×2 grid is equal to zero, the continuous divergence is still not equal to zero — even if the turning can now be realized properly. See Figure 3.3.6.

[15] Another five nodes are added.

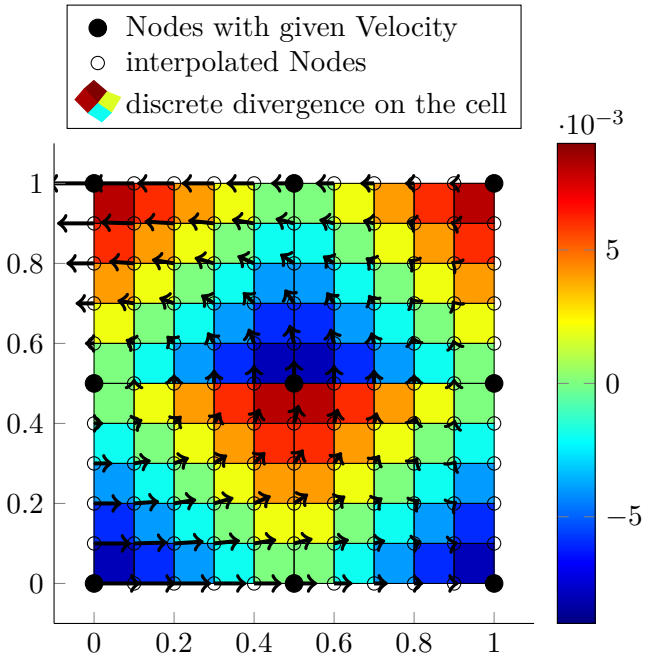

Figure 3.3.6.: interpolating field using pagoda basis functionson a refined 2×2 grid

Now divergence–free basis functions are used for the same initial setup: Again (3.1.1) is used for the calculation of $\hat{\boldsymbol{u}}_h(\boldsymbol{x}, t)$ and a similar expression with divergence–free basis functions is obtained.

$$\begin{aligned}
\hat{\boldsymbol{u}}_h(\boldsymbol{x}, t) &= \sum_{i=1}^{4} u_i(t)\, \boldsymbol{\Phi}_i^x + v_i(t)\, \boldsymbol{\Phi}_i^y = u_1 \boldsymbol{\Phi}_1^x + u_3 \boldsymbol{\Phi}_3^x \\
&= u_1 \begin{pmatrix} \tilde{\phi}(x - x_1, y - y_1) \\ \tilde{\varphi}(x - x_1, y - y_1) \end{pmatrix} + u_3 \begin{pmatrix} \tilde{\phi}(x - x_3, y - y_3) \\ \tilde{\varphi}(x - x_3, y - y_3) \end{pmatrix} \\
&= (+1) \begin{pmatrix} \tilde{\phi}(x - 0, y - 0) \\ \tilde{\varphi}(x - 0, y - 0) \end{pmatrix} + (-1) \begin{pmatrix} \tilde{\phi}(x - 0, y - 1) \\ \tilde{\varphi}(x - 0, y - 0) \end{pmatrix}.
\end{aligned}$$

In order to avoid piecewise definitions only $\Omega_{IV} = \{(x, y) \mid x \leq y \wedge y \leq h - x\}$ will be considered as an example. Ω_{IV} is identical to the area IV from Figure 3.3.2. Inserting the definitions of $\tilde{\phi}$ and $\tilde{\varphi}$ from subsection 3.3.2 leads to the following expression for $\hat{\boldsymbol{u}}_h(\boldsymbol{x}, t)$ on Ω_{IV}:

$$\hat{\boldsymbol{u}}_h(\boldsymbol{x}, t) = (+1) \begin{pmatrix} 1 - \frac{x}{2} - y \\ \frac{x}{2} \end{pmatrix} + (-1) \begin{pmatrix} 1 - \frac{x}{2} + (y - 1) \\ -\frac{x}{2} \end{pmatrix} = \begin{pmatrix} 1 - 2y \\ x \end{pmatrix}.$$

The divergence of $\hat{\boldsymbol{u}}_h(\boldsymbol{x}, t)$ is equal to

$$\boldsymbol{\nabla} \cdot \hat{\boldsymbol{u}}_h(\boldsymbol{x}, t) = \frac{\partial}{\partial x}(1 - 2y) + \frac{\partial x}{\partial y} = 0.$$

For the other subsections of Ω the divergence is naturally also equal to zero. Unlike using pagoda basis functions, the continuous field using divergence–free basis functions is divergence–free . A plot of the field using divergence–free basis functions is given in Figure 3.3.7.

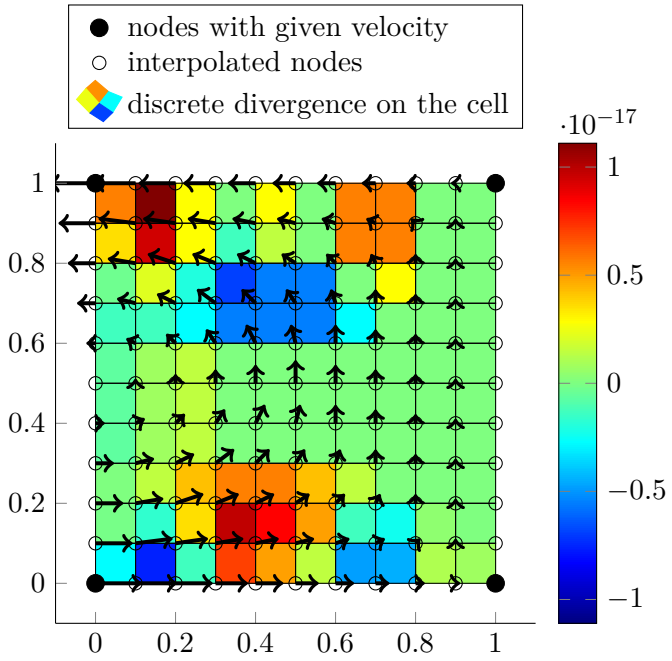

Figure 3.3.7.: interpolating field using divergence–free basis functions; the divergence of the field is on the scale of numerical errors

Using divergence–free basis functions one can realize the turning of the fluid. The fluid is neither compressed nor expanded, because of the divergence–free velocity field. The calculated divergence using (3.1.6) is only in the range of numerical errors.

This simple example already shows, that by enforcing discrete incompressibility divergence–free basis functions can generate continuously divergence–free velocity fields while fields generated using pagoda basis functions do have a divergence not equal to zero. From (2.1.4) follows that violating incompressibility may cause errors in the energy conservation. Therefore the use of divergence–free basis functions prevents errors originating from a violation of the energy conservation by guaranteeing pointwise incompressibility.

4. Implementation

4.1. Structure of the program

The FEM approach described in the previous sections has been implemented in the MATLAB–code *Quickfluid*. The necessary local matrices for the discrete Navier–Stokes equations have been obtained using MAPLE–worksheets developed in [5] and [7] and are listed in subsection A.1.1 and subsection A.1.2. By choosing a set of matrices corresponding with the type of basis functions, one can switch between standard FEM and divergence–free FEM.

Using these local matrices one can calculate pressure, velocity and acting forces using a matrix–free approach which is explained later in this chapter. One has to keep in mind that *Quickfluid* is not using the exact mass matrix but a lumped mass matrix in order to avoid solving another linear equation system in each timestep. The concept of mass lumping will also be explained in subsection 4.4.1.

Initial and boundary conditions for different scenarios (see chapter 5) are hard–coded in *Quickfluid*. Using a configuration file one can choose the type of scenario with the corresponding boundary and initial conditions. Furthermore important parameters like the dimension of the simulation domain, the Reynolds number, the number of cells of the discretization and the size of one timestep can be determined using the configuration file.

In each time step the PPE is set up — again using a matrix–free approach — and solved exactly using the backslash operator in MATLAB. As a last step inside the timeloop time integration is performed using an explicit Euler method. The approach of calculating the pressure field via the pressure Poisson equation in the first place and then updating the velocity field by applying the acting forces and the pressure gradient is called the 'Chorin projection method'.

For scenarios with given initial conditions an important and quite complex preprocessing step is the calculation of the discrete initial velocity field via an L^2–projection. This step is also implemented in *Quickfluid* for scenarios where it is necessary. See section 4.5 for a detailed description of this method.

For postprocessing the tool *Quickvis* is available, which is used for the visualization

of the velocity and pressure field. Additional tools for the visualization of the applied boundary conditions as well as the visualization of the divergence of a given velocity field have been implemented (see section 4.6).

The rough structure of the program with the single steps necessary in each timestep is visualized in Figure 4.1.1.

4.2. Implementation of element matrices

In order to be able to compute the test scenarios using divergence–free basis functions one needs to compute and implement the local element matrices. In [1] these matrices have been developed and calculated for pagoda basis functions as well as divergence–free basis functions for 2D computations. In [7] these matrices have also been computed for the three–dimensional case. For the computation of the matrices MAPLE has been used in order to evaluate the integrals analytically.

The big advantage of the implementation of the divergence–free FEM approach used in *Quickfluid* can be discovered when implementing divergence–free basis functions into an existing FEM–framework: Key features like the compact support of the basis functions and the number of degrees of freedom per cell are the same and therefore the program does not have to be changed except for the element matrices in use.

4.3. Different concepts of numbering

In *Quickfluid* different concepts of numbering are used. In the following these concepts as well as their fields of application will be explained. Particularly the differences in global numbering (on the whole domain) and local numbering (on a single cell or around a single node) will be explained in order to avoid confusion in the following chapters.

4.3.1. Z–numbering

When looking at single cells one often needs to gather information from the neighbouring nodes. These nodes are traversed in the order illustrated in Figure 4.3.1. This type of numbering is a local numbering, because for each cell the adjacent nodes could be ordered in that way, but the numbering does not uniquely identify certain nodes like a global numbering would do.

The Z–numbering is applied in all kinds of local operations on cells, like for example the calculation of the divergence. We remember, that the divergence of the velocity $\boldsymbol{\nabla} \cdot \boldsymbol{u}$ is

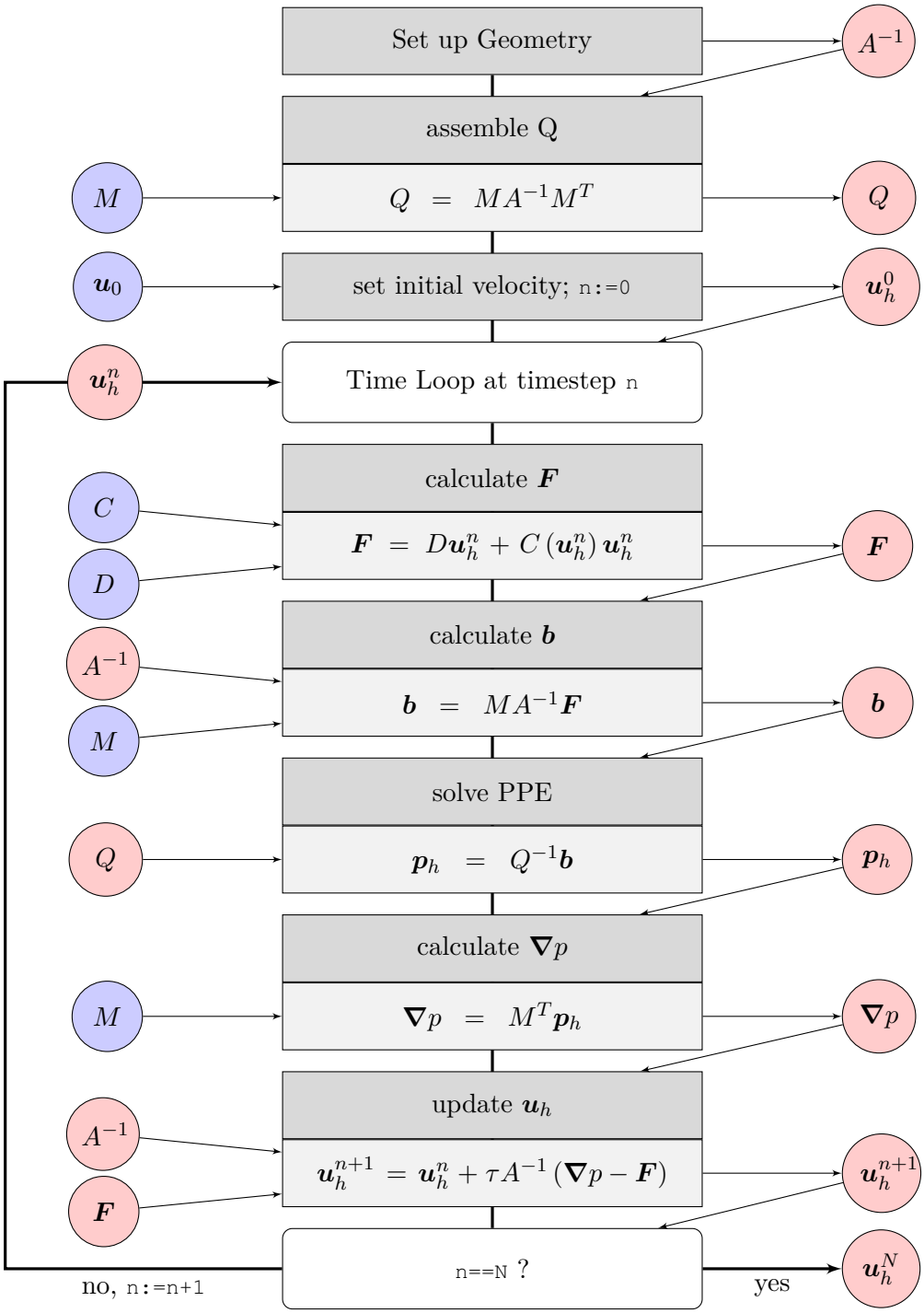

Figure 4.1.1.: structure of the program *Quickfluid*, red circles denoting already given data (element matrices M, C, D and continuous initial condition $\boldsymbol{u}_0$), blue circles denoting data generated within *Quickfluid*

discretized through the matrix M. The derivation of the local matrix M_{loc} has already been done in subsection 3.1.3 by using a finite volume approach. The local matrix M_{loc} for one cell has the following form:

$$M_{\text{loc}} = \begin{pmatrix} -0.5 & 0.5 & -0.5 & 0.5 & 0.5 & 0.5 & -0.5 & -0.5 \end{pmatrix}.$$

If the velocities on the corners of the cell are put into one vector $\boldsymbol{u}_h$ in the order given by Z–numbering[1] one can easily calculate the discrete divergence of the cell by calculating the following matrix–vector–product:

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = M\boldsymbol{u}_h = \begin{pmatrix} -0.5 & 0.5 & -0.5 & 0.5 & 0.5 & 0.5 & -0.5 & -0.5 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}.$$

One may notice that calculating this product gives exactly the expression (3.1.6), which is the discrete continuity equation. Of course this only gives the divergence for one cell[2].

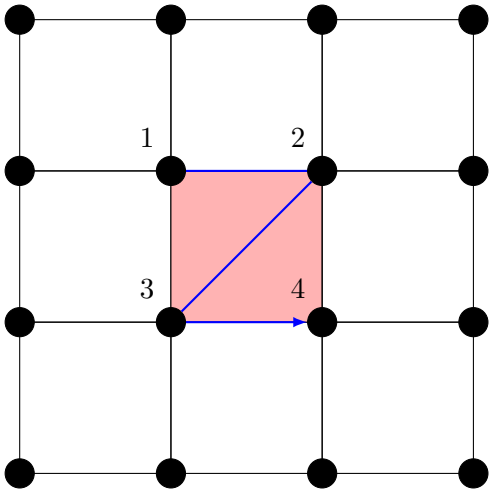

Figure 4.3.1.: Z–numbering pattern of the nodes around one cell

[1] The x-component of the velocity first, then the y-component

[2] The global matrix M can be used for the calculation of the divergence of each cell. The assembly of M will be explained later in this section.

The same numbering is also applied for the calculation of the pressure gradient $\boldsymbol{\nabla} p$ through $-M^T \boldsymbol{p}_h$. Here the resulting gradient $(\boldsymbol{\nabla} p)_i = (\ (\boldsymbol{\nabla} p_x)_i \quad (\boldsymbol{\nabla} p_y)_i\ )^T$ on node i is returned in the order defined by Z–numbering around the cell with given pressure p.

$$-M_{\text{loc}}^T p = - \begin{pmatrix} -0.5 \\ 0.5 \\ -0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ -0.5 \\ -0.5 \end{pmatrix} p = \begin{pmatrix} 0.5p \\ -0.5p \\ 0.5p \\ -0.5p \\ -0.5p \\ -0.5p \\ 0.5p \\ 0.5p \end{pmatrix} = \begin{pmatrix} (\boldsymbol{\nabla} p_x)_1 \\ (\boldsymbol{\nabla} p_x)_2 \\ (\boldsymbol{\nabla} p_x)_3 \\ (\boldsymbol{\nabla} p_x)_4 \\ (\boldsymbol{\nabla} p_y)_1 \\ (\boldsymbol{\nabla} p_y)_2 \\ (\boldsymbol{\nabla} p_y)_3 \\ (\boldsymbol{\nabla} p_y)_4 \end{pmatrix}.$$

Again this will only lead to the pressure gradient for the nodes surrounding one cell[3].

4.3.2. Counter–clockwise–numbering

Sometimes one wants to traverse the cells lying along one node. Here a different numbering, a counter–clockwise–numbering (CCW–numbering) shown in Figure 4.3.2, is applied. The function which assembles the matrix Q, which is important for solving the

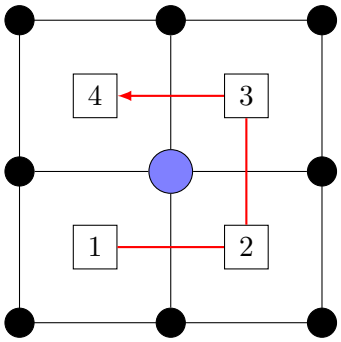

Figure 4.3.2.: CCW–numbering pattern of the cells around one node

PPE, uses this kind of numbering. This kind of numbering is also a local numbering.

[3]The x–component of the gradient comes — again — first, then the y–component

4.3.3. Lexicographical numbering

In order to be able to uniquely identify cells as well as nodes, a lexicographical numbering is used. The ordering is illustrated in Figure 4.3.3 for nodes and in Figure 4.3.4 for cells. The figures show a simulation domain Ω which has the dimension of 4×2 cells, respectively 5×2 nodes. In *Quickfluid* the dimensionality of the simulation domain is saved via the number of cells in x–direction N_x —- respectively y–direction N_y. The number of nodes in each direction can be calculated easily, since it is just $n_x = N_x + 1$ (respectively $n_y = N_y + 1$) on a regular Cartesian grid. The total number of cells $N = N_x N_y$ and $n = n_x n_y$ is also easily calculated.

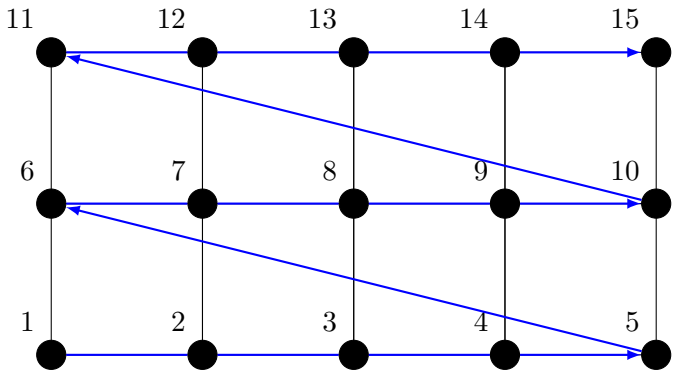

Figure 4.3.3.: lexicographical numbering of the nodes on the whole domain

Many different properties of the fluid are saved on the nodes — for example the velocity. The vector $\boldsymbol{u}_h$ has $2n = 2n_x n_y = 2(N_x{+}1)(N_y{+}1) = 20$ entries[4]. In $\boldsymbol{u}_h$ all x–components of the velocity are saved first in a order given by the lexicographical numbering; then the y–components follow. The velocity on node i is composed of the x–component $(\boldsymbol{u}_h)_i$ and the y–component $(\boldsymbol{u}_h)_i$. The pressure is saved for each cell and therefore the vector $\boldsymbol{p}_h$ has N entries, since the pressure is a scalar quantity.

4.3.4. Matrix–free approach

In *Quickfluid* mainly a matrix–free approach is used. That means that no big global matrices are assembled, but that all cells or nodes are traversed and the contributions of each cell or node are summed up to the requested quantity. One example is the calculation of the pressure gradient:

- Initialize the global vector $\boldsymbol{\nabla} p$ with the dimension $2n$.
- Loop over all N cells of the domain in a lexicographical manner.

[4]The two is due to the two components of the velocity on each node.

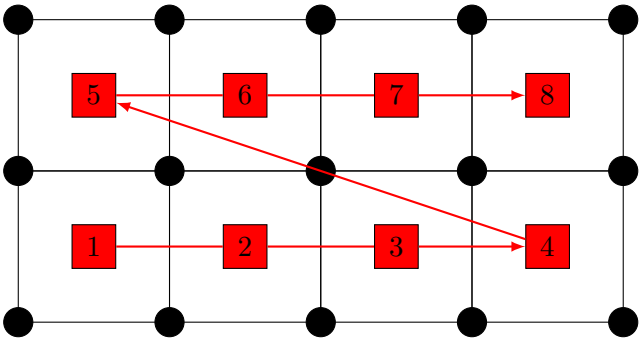

Figure 4.3.4.: lexicographical numbering of the cells on the whole domain

- Calculate the local contribution of the pressure $(\boldsymbol{p}_h)_I$ on cell I to the pressure gradient on each node neighbouring cell I by $-M_{\text{loc}}^T (\boldsymbol{p}_h)_I$ (Result is returned in Z–numbering!).
- Distribute the local contributions to the global vector $\boldsymbol{\nabla} p$ by applying a proper mapping from the current cell I and the local numbering to the global numbering[5].

Additionally to the pressure gradient also the right hand side of the PPE $\boldsymbol{b}$ as well as the acting force $\boldsymbol{F}$ is calculated via a matrix–free approach.

4.3.5. Assembly routine

An alternative to a matrix–free approach is the assembly of global matrices. One will face some disadvantages, like the huge sparse matrices generated by the assembly routine[6], but sometimes the global representation of the matrix is inevitable. The assembly routine always follows the same recipe. The algorithm will be demonstrated for the assembly of the matrix M:

- Initialize the global matrix M with the dimension $N \times 2n$.
- Loop over all N cells of the domain in a lexicographical manner.
- Calculate the global indices of the nodes by applying a proper mapping from the

[5]The domain given in Figure 4.3.4 and Figure 4.3.3 can be considered as an example: Let $I = 3$, then the global indices of the neighbouring nodes are $i = 3, 4, 8, 9$. In Z–numbering these nodes will be returned in the following order: x–component of nodes $8, 9, 3, 4$ and then y–component of nodes $8, 9, 3, 2$. Knowing the global indices of the returned results it is easy to save the results to the right positions in $\boldsymbol{\nabla} p$.

[6]For the very small scenario (only 8 cells) shown in Figure 4.3.3 and Figure 4.3.4The matrix M has the dimension 8×30; the matrix A has the dimension 30×30. One might imagine how big these matrices can become for scenarios with bigger dimensions.

current cell I and the local numbering to the global numbering.

- Distribute the entries of the local matrix M_{loc} to the global matrix M at the global indices corresponding to the currently processed nodes.

One might recognize the similarities between this algorithm and the one given in subsection 4.3.4. The global Matrices M and A are needed for example when calculating the L^2–projection and for the calculation of the discrete divergence of several cells. Also the important matrix Q is calculated in a similar way, but with slight modifications due to the geometry information of each cell which will be explained in subsection 4.4.2.

4.4. Implementation of new boundary conditions

Quickfluid initially only supported simple scenarios like channel flow or lid–driven cavity with simple boundary conditions. For the test scenarios computed in chapter 5, inflow and outflow boundaries have to be added in every direction[7] as well as slip–wall boundaries — a new kind of boundary condition. These boundary conditions have already been presented in section 3.2. In order to add these boundary conditions to *Quickfluid* the calculation routines of the following matrices and vectors have to be modified:

- PPE matrix $Q = MA^{-1}M^T$
- the force vector $\boldsymbol{F} = D\boldsymbol{u}_h + C(\boldsymbol{u}_h)\,\boldsymbol{u}_h$
- PPE load vector $\boldsymbol{b} = MA^{-1}\boldsymbol{F}$

In this section the necessary modifications will be presented and the important mass lumping of A will be explained.

4.4.1. Mass lumping

For the calculation of the matrix Q and the vector $\boldsymbol{b}$ one is in need of the inverse of the mass matrix A. The calculation of the inverse A^{-1} implies the solution of an additional linear equation system. In order to avoid this computational cost, the mass matrix A will be lumped to the approximation $\tilde{A}$, with an easily accessible inverse.

For quadratic cells with the edge length h the following approximation $\tilde{A}_{\text{loc}}$ of A_{loc} is used:

$$A_{\text{loc}} \approx \tilde{A}_{\text{loc}} = \frac{1}{4}h^2\text{Id}.$$

[7] In channel flow only inflow from the left and outflow on the right side of the simulation domain is necessary.

$\mathrm{Id} \in \mathbb{R}^{4\times 4}$ is the identity matrix. The global lumped mass matrix $\tilde{A}$ can be constructed using the already known assembly routine. The inverse $\tilde{A}^{-1}$ of the diagonal matrix $\tilde{A}$ can be calculated easily. For cells with index i the following values for the inverse are obtained, if

- the cell is inside the domain: $\left(\tilde{A}^{-1}\right)_{i,i} = \frac{1}{h^2}$
- the cell is at the side of the domain: $\left(\tilde{A}^{-1}\right)_{i,i} = \frac{2}{h^2}$
- the cell is on a corner of the domain: $\left(\tilde{A}^{-1}\right)_{i,i} = \frac{4}{h^2}$

For a detailed derivation of mass lumping see [3] or section 5 in [1]. Also see chapter 6 in this thesis for a short discussion about mass lumping.

4.4.2. PPE matrix Q

If one uses the global lumped mass matrix $\tilde{A}$, the PPE matrix $Q = M\tilde{A}^{-1}M^T$ can be assembled using a matrix–free approach in the following way:

- Initialize the global matrix Q with the dimension $N \times N$, where N is the number of cells.
- Loop over all n nodes in the domain in a lexicographical manner.
- Decide weather the current node i is an inner node, an outlet–boundary node, a slip–boundary node, an outlet–corner node, a slip–corner node or a Dirichlet node.
- Find the cells neighbouring the current node i and add the correct weights — depending on the kind of node — to the matrix Q at the positions corresponding to the neighbouring cells.

The correct weights are calculated by determining the influence of one node on the global matrix $Q = M\tilde{A}^{-1}M^T$ for the different cases mentioned above. On the one hand these cases differ due to the value of $\left(\tilde{A}^{-1}\right)_{i,i}$ on the other hand because of the number of degrees of freedom at the current node due to the kind of applied boundary condition. Note also that the global lumped mass matrices $\tilde{A}$ for pagoda basis functions and divergence–free basis functions are identical for all nodes **but** the corner nodes, where the weights differ: For pagoda basis functions the value of $\tilde{A}$ is equal to $\frac{h^2}{4}$ while for divergence–free basis functions the value of $\tilde{A}$ is equal to $\frac{h^2}{6}$ in the bottom left and top right corner respectively $\frac{h^2}{3}$ in the top left and bottom right corner. In this thesis no corner outlet boundary conditions occur and therefore this special case does not have to be treated.

Matrices $\left(M\tilde{A}^{-1}M^T\right)_{\text{loc}} \in \mathbb{R}^{4\times 4}$ showing the influence of one node on the surrounding cells numbered in counter–clockwise order (see subsection 4.3.2) are calculated[8] in order to assemble the global matrix $Q = M\tilde{A}^{-1}M^T$. In Table 4.1 some examples for the weights for the different cases are presented.

node type	node position	value of $\left(\tilde{A}^{-1}\right)_{i,i}$	$\left(M\tilde{A}^{-1}M^T\right)_{\text{loc}}$
free	inside	$\frac{1}{h^2}$	$\frac{1}{4h^2}\begin{pmatrix} 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \end{pmatrix}$
outflow	left wall	$\frac{2}{h^2}$	$\frac{1}{4h^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
outflow [9]	bottom–left corner	$\frac{4}{h^2}$	$\frac{1}{4h^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
outflow [10]	bottom–left corner	$\frac{6}{h^2}$	$\frac{1}{4h^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
slip	left wall	$\frac{2}{h^2}$	$\frac{1}{4h^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & -2 & 0 \\ 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
slip [11]	bottom–left corner	$\frac{4}{h^2}$	$\frac{1}{4h^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
slip [12]	bottom–left corner	$\frac{6}{h^2}$	$\frac{1}{4h^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
dirichlet	any	unnecessary	$\frac{1}{4h^2}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Table 4.1.: different weights used in the assembly routine of Q, weights given in the style of element matrices

[8] For boundary nodes also the neighbouring cells lying outside of the domain are presented in the matrix due to a consistent numbering. Of course these weights are equal to zero.
[9] for pagoda basis functions
[10] for divergence–free basis functions
[11] for pagoda basis functions
[12] for divergence–free basis functions

For Dirichlet nodes as well as for slip–corner nodes all weights vanish and therefore these nodes are not considered at all in the assembly loop.

4.4.3. Force vector F

The force vector $\boldsymbol{F} = D\boldsymbol{u}_h + C(\boldsymbol{u}_h)\,\boldsymbol{u}_h$ is also composed using a matrix–free approach:

- Initialize the global vector $\boldsymbol{F}$ with the dimension $2n \times 1$.
- Iterate over all N cells of the domain in a lexicographical manner.
- Calculate the local contribution of the velocities of the nodes surrounding cell I to $\boldsymbol{F}$[13].
- Distribute the local contributions to the global vector $\boldsymbol{F}$ by applying a proper mapping from the current cell I and the local numbering to the global numbering.

Only dirichlet nodes have to be treated in a special way: Dirichlet nodes have a fixed velocity and no acceleration is applied at these nodes; therefore the force $\boldsymbol{F}_i$ and $\boldsymbol{F}_{i+n}$ is equal to zero. For slip–boundary nodes the force in normal direction is equal to zero due to the same reason.

4.4.4. PPE load vector b

The PPE load vector $\boldsymbol{b} = MA^{-1}\boldsymbol{F}$ is also calculated using the lumped mass matrix $\tilde{A}$. The values of $\tilde{A}^{-1}$ follow the same rules like when calculating Q. But this time the vector is composed by iterating over the cells and summing up the contributions of the neighbouring nodes to each cell. The diagonal matrix $\tilde{A}^{-1}$ is just scaling the lines in the vector $\boldsymbol{F}$ and then M_{loc} is multiplied from the left to the entries in $\left(\tilde{A}^{-1}\boldsymbol{F}\right)_{\text{loc}}$ in order to get the entry in $\boldsymbol{b}$ corresponding to the current cell.

[13] We calculate the contributions to $\boldsymbol{F}$ by using the local matrices D_{loc} and C_{loc} and picking the velocities of the nodes surrounding the curring cell out of $\boldsymbol{u}_h$.

4.5. Implementation of initial conditions

For the test scenario Taylor–Green vortex flow (see section 5.5) initial conditions have to be defined in a proper way. Therefore a given continuous initial condition (IC) of the following form

$$\boldsymbol{u}\left(\boldsymbol{x}, t=0\right) = \boldsymbol{u}_0\left(\boldsymbol{x}\right) = \begin{pmatrix} u_0\left(\boldsymbol{x}\right) \\ v_0\left(\boldsymbol{x}\right) \end{pmatrix}$$

has to be discretized.

The discretization is needed in order to get a valid initial setup for the FEM algorithm given through the vector $\boldsymbol{u}_h^0$. The resulting approximation $\hat{\boldsymbol{u}}_h^0\left(\boldsymbol{x}\right)$ of the continuous IC $\boldsymbol{u}_0\left(\boldsymbol{x}\right)$ can be formulated using the basis functions $\boldsymbol{\Phi}_i$ and the corresponding weights $\left(\boldsymbol{u}_h^0\right)_i$:

$$\hat{\boldsymbol{u}}_h\left(\boldsymbol{x}, t=0\right) = \hat{\boldsymbol{u}}_h^0\left(\boldsymbol{x}\right) = \sum_{i=1}^{2n} \left(\boldsymbol{u}_h^0\right)_i \boldsymbol{\Phi}_i.$$

4.5.1. The interpolation of initial conditions

The naive way for generating discrete ICs would be just taking the interpolation of $\boldsymbol{u}_0\left(\boldsymbol{x}\right)$:

$$\left(\boldsymbol{u}_h^0\right)_i = u_0\left(\boldsymbol{x}_i\right)$$

and

$$\left(\boldsymbol{u}_h^0\right)_{i+n} = v_0\left(\boldsymbol{x}_i\right).$$

The interpolation of $\boldsymbol{u}_0\left(\boldsymbol{x}\right)$ just takes the exact values of $\boldsymbol{u}_0\left(\boldsymbol{x}\right)$ at the n nodes $\boldsymbol{x}_i$. This means the resulting discrete IC is exact at the nodes and interpolates the continuous IC via the given basis functions in–between the nodes. But this very easy way of generating ICs does not lead to correct results, because the interpolation is not the best approximation[14] of $\boldsymbol{u}_0\left(\boldsymbol{x}\right)$. Even though it is exact in the nodes, a more elaborated technique, the L^2–projection, has to be used for the generation of valid ICs for the FEM Algorithm.

[14] A good approximation minimizes the L^2–error. An exact definition of the L^2–error will follow in subsection 4.5.3.

4.5.2. Theoretical background of the L^2–projection

The L^2–projection leads to the discrete IC which is closest to the continuous IC considering the L^2–norm. Furthermore the constraint of incompressibility of the resulting velocity field will be applied. For the used discretization this L^2–projected IC is the best approximation of the continuous IC, because the projection moves the IC from a general function space U to the function space of the discretization $U_h \subset U$.

The following sections strongly bases on the appendix on projections given in [3], which can be referred for more details on function spaces and the mathematical derivation of the projections presented on the following pages.

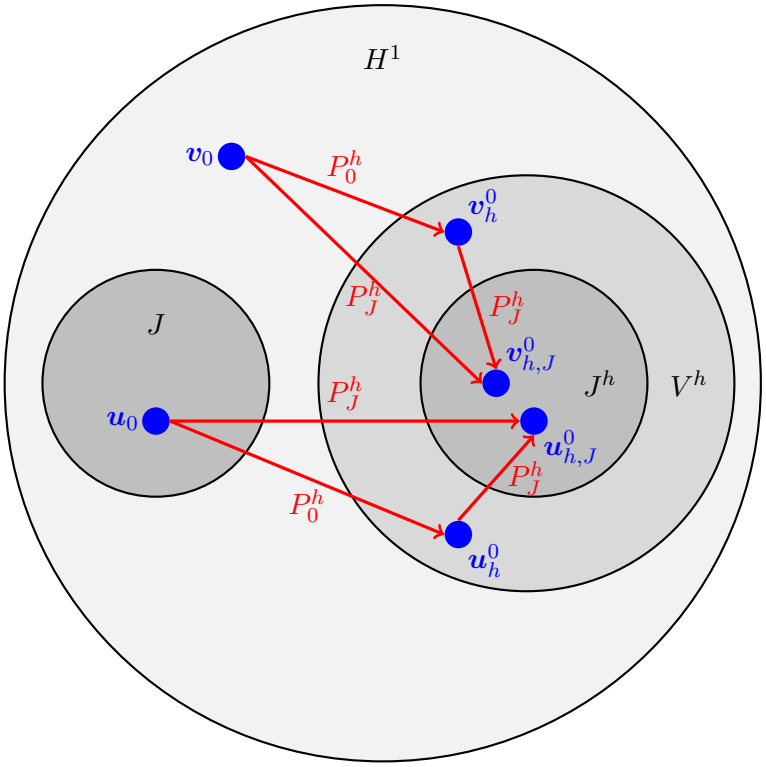

Figure 4.5.1.: some L^2–projection (red arrows) of velocity fields (blue dots) from one function space (circles) to another; Figure **A.3.3-1** from [3]

An illustration of the different L^2–projections of velocity fields can be found in Figure 4.5.1. This figure is motivated by Figure **A.3.3-1** in [3]. This figure finally gives a good overview over the function spaces of interest and the different projections of vector–valued functions:

- The space H^1 is the largest considered function space. It is assumed that the initial velocity field $\boldsymbol{u}_0$ lies inside H^1.

- The space $J \subset H^1$ is the space of continuously divergence–free functions. The initial velocity field $\boldsymbol{u}_0$ is divergence–free (if $\boldsymbol{\nabla} \cdot \boldsymbol{u}_0 = 0$ then $\boldsymbol{u}_0 \in J \subset H^1$). But the initial velocity field does not necessarily have to be divergence–free: The field $\boldsymbol{v}_0 \in H^1$ for example is not divergence–free and therefore $\boldsymbol{v}_0 \notin J$.

- The space $V^h \subset H^1$ contains velocity fields generated through a certain discretization and basis functions (like for example $\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})$).

- The space $J^h \subset V^h \subset H^1$ contains discretely divergence–free velocity fields inside of V^h. One may notice that the spaces J^h and J do not necessarily overlap. Therefore a proper projection from J to J^h is needed when generating initial conditions.

In the following three sections different L^2–projections will be presented:

- The first section deals with projections of scalar–valued functions in 1D and 2D. This kind of projection is not relevant for the problem of projecting initial conditions for flow problems, but the derivation of it will be useful for the explanation of the theoretical background and the validation of the results by comparing projected functions to examples given in [3].

- The second projection is a projection of a vector–valued function in 2D (like for example a velocity field). This projection is also shown in Figure 4.5.1(P_0^h). It is simply a L^2–projection of any velocity field ($\boldsymbol{u}_0, \boldsymbol{v}_0 \in H^1$) onto the function space of a certain discretization ($P_0^h(\boldsymbol{u}_0) = u_h^0 \in V^h$ and $P_0^h(\boldsymbol{v}_0) = v_h^0 \in V^h$).

- The last projection is also a projection of a vector–valued function in 2D and is shown in Figure 4.5.1 (P_J^h). Here the constraint of incompressibility is added and this projection maps any velocity field ($\boldsymbol{u}_0, \boldsymbol{v}_0 \in H^1$) onto the discretely divergence–free velocity field via an L^2–projection ($P_J^h(\boldsymbol{u}_0) = u_{h,J}^0 \in J^h$ and $P_J^h(\boldsymbol{v}_0) = v_{h,J}^0 \in J^h$). This projection is exactly the projection of interest when looking for valid initial conditions for flow problems. One may notice that also not necessarily continuously divergence–free functions (like $\boldsymbol{v}_0 \notin J$) can be mapped to discretely divergence–free functions via the projection P_J^h.

This theoretical explanation of the L^2–projection is finished by the following two excerpts from [3] showing that an L^2–projection with the constraint of incompressibility, like explained in subsection 4.5.5, is inevitable when computing flow problems with given ICs and should point out the relevance of the following section:

'Even if $\boldsymbol{u}$ *is 'perfectly' divergence–free (*$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0$*), its projection (*$\boldsymbol{u}_h^0$*) will not generally be'...* and additionally to that *'a (seemingly 'superior') perfectly divergence–free vector field* ***must*** *be projected to its ('inferior') discretely divergence–free counterpart in order to provide a legitimate IC for time–integration of the GFEM NSE equations. This is because* $\boldsymbol{J}^h$ *is* ***not*** *a subset of* $\boldsymbol{J}$ *— even though* $\boldsymbol{J}^h \subset \boldsymbol{V}^h$ *and* $\boldsymbol{V}^h \subset \boldsymbol{H}^1$ *and* $J \subset \boldsymbol{H}^1$*'*

4.5.3. L^2–projection of scalar–valued functions in 1D and 2D

4.5.3.1. Derivation of the projection

The L^2–norm ($\|\cdot\|_0$)of an arbitrary function $u(x) \in Ł^2$ is defined in the following way:

$$\|u(x)\|_0^2 = \int u(x)^2 \, \mathrm{d}x.$$

The L^2–projection is used to find the closest function $v(x)$ to a given function $u(x) \in L^2$ considering the L^2–norm. The choice of $v(x)$ is restricted to the subspace $S \subset L^2$. In general $u(x) \notin S$ and therefore $u(x) \neq v(x)$. The best result would be an approximation $v(x)$ of $u(x)$ which minimizes the L^2–error $\|v(x) - u(x)\|_0$[15].The L^2–projection can therefore be formulated as the following optimization problem:

$$\text{Find } \inf_{v \in S} \|v - u\|_0 .$$

Because of the positive definiteness of the norm the following optimization problem is aquivalent:

$$\text{Find } \inf_{v \in S} \|v - u\|_0^2 .$$

In other words the functional

$$F_0(v) = \|v - u\|_0^2 = \int (v - u)^2 \, \mathrm{d}x$$

has to be minimized. Varying v leads to

$$\delta F_0(v) = 2 \|v - u\|_0^2 = 2 \int (v - u) \, \delta v \, \mathrm{d}x.$$

Due to variational calculus a minimum is obtained if this equation is equal to zero. Additionally, v and δv can be substituted by linear combinations of the basis functions v_i of S in the following way:

$$v = \sum_{i=1}^{\infty} a_i v_i$$
$$\delta v = \sum_{i=1}^{\infty} b_i v_i.$$

[15] One is facing the same problem when trying to discretize a given IC $\boldsymbol{u}_0$: The given IC cannot — in general — be represented by the chosen set of basis functions and therefore one has to find the closest approximation of $\boldsymbol{u}_0$ by the given basis functions, which is $\hat{\boldsymbol{u}}_h^0 \in V^h$.

The condition for a minimum reads

$$\sum_{j=1}^{\infty} b_j \int \sum_{i=1}^{\infty} (a_i v_i - u)\, v_j \, \mathrm{d}x = 0.$$

Due to variational calculus this expression can only be equal to zero if

$$\int \sum_{i=1}^{\infty} (a_i v_i - u)\, v_j \, \mathrm{d}x = 0 \qquad \text{for } j = 1, 2, ...\infty.$$

For the scalar L^2–projection not infinitely many, but n basis functions $\mathbf{\Phi}_i$(one for each gridpoint) are availaible. The weights a_i will be the values of the approximative function $\hat{\boldsymbol{u}}_h$ on the grid points. Thus a_i is substituted by $(\boldsymbol{u}_h)_i$.

$$\int \sum_{i=1}^{n} ((\boldsymbol{u}_h)_i\, \mathbf{\Phi}_i - u)\, \mathbf{\Phi}_j = 0 \qquad \text{for } j = 1, 2, ...n.$$

Rearranging the equation leads to

$$\sum_{i=1}^{n} (\boldsymbol{u}_h)_i \int \mathbf{\Phi}_i \mathbf{\Phi}_j = \int u \mathbf{\Phi}_j \qquad \text{for } j = 1, 2, ...n.$$

By using the global mass matrix A[16], the vector $\boldsymbol{u}_h$ holding the weights and the load vector $(\boldsymbol{b})_i = \int u \mathbf{\Phi}_i \, \mathrm{d}x$[17], this expression can be also written in matrix–vector–notation:

$$A\boldsymbol{u}_h = \boldsymbol{b}.$$

This system of equations can now be solved using standard techniques from linear algebra.

4.5.3.2. Numerical Examples

This method has implemented been implemented by first calculating the local mass matrix A_{loc} and assembling the global mass matrix A via an assembly routine. The load vector $\boldsymbol{b}$ is calculated by numerical integration. The speed of the calculation of $\boldsymbol{b}$ can be further improved by considering the compact support of the basis functions when defining the integration intervals. Especially in 2D this causes a massive speedup.

In order to test the implementation, hat functions have been used as basis functions in

[16] The local mass matrix is already known from the FEM algorithm and therefore this matrix only has to be assembled through an easy assembly routine using the local mass matrix

[17] The calculation of $\boldsymbol{b}$ is often the bottleneck of the L^2–projection and very costly.

1D and pagoda basis functions in 2D. Two examples shown in [3] in **A.3.2.7** have been reproduced.

The first example is shown in Figure 4.5.2. One clearly sees that the resulting function is not the interpolation of the given function, because the values on the nodes strongly differ towards the discontinuity. Still this approximation is the best approximation of the original function considering the L^2–norm.

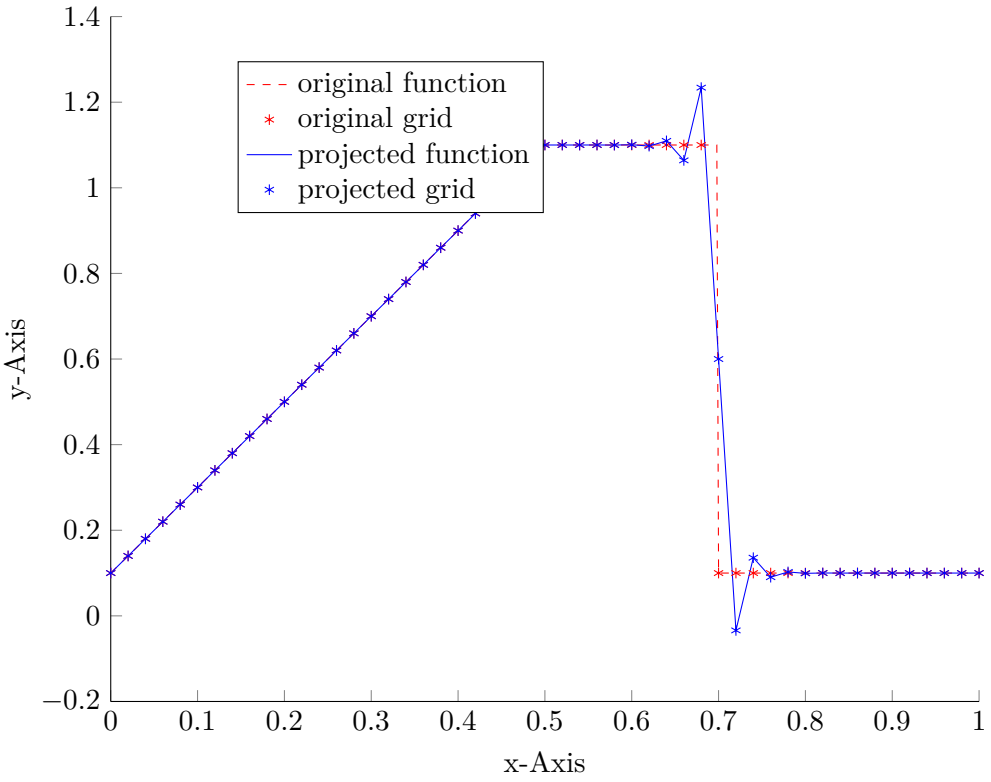

Figure 4.5.2.: plot of a discontinuous function and its L^2–projection ; problem inspired by Figure **A.3.2-23** in [3]

The second example is shown in Figure 4.5.3. The original function is the following 2D standard distribution

$$u(x,y) = e^{-\frac{1}{2}\left[\left(\frac{x-x_0}{\sigma_x}\right)^2+\left(\frac{y-y_0}{\sigma_y}\right)^2\right]}$$

with the following parameters:

$$\sigma_x = \frac{1}{6}, \sigma_y = 0.3\sigma_x, x_0 = y_0 = 1.$$

The L^2–projection has been calculated on the domain $[0,2]^2$ with 13×13 gridpoints. The maximum and minimum of the approximative function shown in Figure 4.5.3 are 0.9808 respectively -0.1065 and match the reference value given in [3]. Again the differences to the interpolant like the pits on the both sides of the standard distribution can be clearly seen.

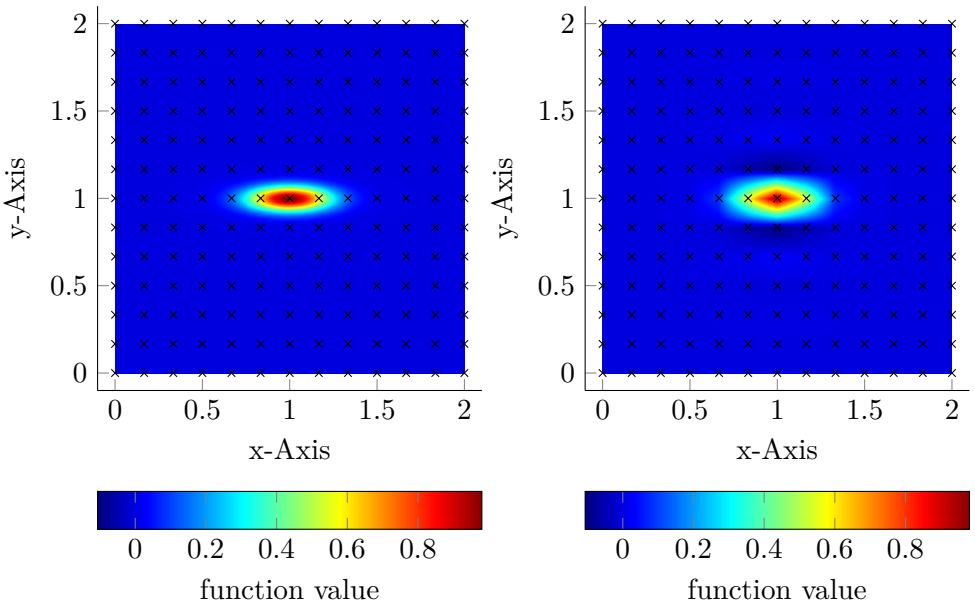

Figure 4.5.3.: a colorplot of a 2D function and its L^2–projection with crosses showing the grid of the projection; problem inspired by Figure **A.3.2-24** in [3]

4.5.4. L^2–projection of vector–valued functions in 2D

The L^2–projection for vector–valued functions is mainly the same like for scalar–valued functions, but with some tiny differences. A slightly different notation, which is more consistent with the notation in *Quickfluid*, will be used, because the 2D L^2–projection will be mainly used for the generation of valid ICs.

4.5.4.1. Derivation of the projection

First of all, of course, the L^2–norm is now acting on vector–valued functions and therefore one has to use a scalar product. The L^2–norm $\|\cdot\|_0$ of a vector–valued function $\boldsymbol{f}(\boldsymbol{x}) \in L^2$ is defined by

$$\|\boldsymbol{f}(\boldsymbol{x})\|_0^2 = \int \boldsymbol{f}(\boldsymbol{x}) \cdot \boldsymbol{f}(\boldsymbol{x}) \ \mathrm{d}\boldsymbol{x}.$$

For a given function $\boldsymbol{u}_0(\boldsymbol{x}) \in H^1 \subset L^2$ the closest function $\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) \in V^h \subset H^1$ is looked for. In other words the L^2–error $\left\|\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) - \boldsymbol{u}_0(\boldsymbol{x})\right\|_0$ has to be minimized. The following optimization problem can be formulated:

$$\text{Find} \inf_{\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) \in S} \left\|\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) - \boldsymbol{u}_0(\boldsymbol{x})\right\|_0^2.$$

This optimization problem is equivalent to the minimization of the functional

$$F\left(\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})\right) = \int \left(\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) - \boldsymbol{u}_0(\boldsymbol{x})\right) \cdot \left(\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) - \boldsymbol{u}_0(\boldsymbol{x})\right) \, \mathrm{d}\boldsymbol{x}.$$

Variation of $\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})$ gives the following condition for a minimum

$$\delta F\left(\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})\right) = 2 \int \left(\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) - \boldsymbol{u}_0(\boldsymbol{x})\right) \cdot \delta\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0.$$

Substituting $\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})$ and $\delta\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})$ by the representation using the $2n$ basis functions $\boldsymbol{\Phi}_i$ on the n nodes (2 basis functions for each node, 1 basis functions for each dimension) gives

$$\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) = \sum_{i=1}^{2n} a_i \boldsymbol{\Phi}_i$$
$$\delta\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) = \sum_{i=1}^{2n} b_i \boldsymbol{\Phi}_i.$$

This leads to the following condition for a minimum:

$$\sum_{j=1}^{2n} b_j \int \sum_{i=1}^{2n} \left(a_i \boldsymbol{\Phi}_i - \boldsymbol{u}_0(\boldsymbol{x})\right) \cdot \boldsymbol{\Phi}_j \, \mathrm{d}\boldsymbol{x} = 0.$$

Or after some modifications equivalent to the scalar–valued case and substituting a_i by $(\boldsymbol{u}_h)_i$ (the weight vector from *Quickfluid*):

$$\sum_{i=1}^{2n} (\boldsymbol{u}_h)_i \int \boldsymbol{\Phi}_i \cdot \boldsymbol{\Phi}_j \, \mathrm{d}\boldsymbol{x} = \int \boldsymbol{u}_0(\boldsymbol{x}) \cdot \boldsymbol{\Phi}_j \, \mathrm{d}\boldsymbol{x} \qquad \text{for } j = 1, 2, ...2n$$

Using the global mass matrix A, the vector $\boldsymbol{u}_h$ holding the weights and the load vector $(\boldsymbol{b})_i = \int \boldsymbol{u}_0(\boldsymbol{x}) \boldsymbol{\Phi}_i \, \mathrm{d}\boldsymbol{x}$, the expression can be rewritten in matrix–vector–notation[18].

$$A\boldsymbol{u}_h = \boldsymbol{b}.$$

This system of equations can again be solved using standard techniques.

[18] This time the dimension of the matrix A will be $2n \times 2n$ because of the two components of the basis functions. The dimension of $\boldsymbol{u}_h$ and $\boldsymbol{b}$ will be $2n \times 1$. The assembly routine has to be modified correspondingly.

4.5.4.2. Numerical examples

The L^2–projection of vector–valued functions has been implemented in the same way like the L^2–projection of scalar–valued functions. The only differences are

- the use of the local mass matrix for the vector–valued basis functions and the corresponding assembly routine,
- a slightly different calculation of $\boldsymbol{b}$ due to the scalar product inside the integral and
- some optimizations in the numerical integration when using piecewise defined basis functions[19].

In order to test the implementation and demonstrate the effect of the L^2–projection the L^2–projection of the field

$$\boldsymbol{u}_0(\boldsymbol{x}) = \begin{pmatrix} \sin(\pi x) \\ \cos(\pi y) \end{pmatrix}$$

on a 3×3 grid on the domain $[0,1]^2$ has been calculated. Figure 4.5.4 shows the values of original field $\boldsymbol{u}_0(\boldsymbol{x})$ and of the projected field $\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})$ on the nodes.

One will see the differences due to the projection:

- The vectors on the left and right edges are shifted in positive x–direction, because the x–value on these gridpoints are $\sin(0) = 0$ and $\sin(\pi) = 0$, but the x–value inside the whole domain is ≥ 0.
- The vectors on the middle of the upper and bottom edge are shifted in negative x–direction, because the x–value on these gridpoints is $\sin(0.5\pi) = 1$, but the x–value inside the whole domain is ≤ 1.

This shows the averaging behaviour of the L^2–projection. The projection takes the whole domain into consideration and fits the discrete values of the approximative function that the results matches best in a global way (L^2–norm). In subsection 4.6.2 the discrete divergence of the approximative function $\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})$ is shown[20], which is clearly not equal to zero (in Figure 4.5.1 the corresponding function is $\boldsymbol{v}_h^0 \notin J^h$). This is what one would expect since the divergence of the original field $\boldsymbol{u}_0(\boldsymbol{x})$ is also not equal to zero.

[19] This is of course very important when using divergence–free FEM . Here each cell has been divided into eight triangular integration domains which are summed up in the end in order to avoid discontinuities inside of one integration domain. In MATLAB this approach is recommended when using the function `integral2`, which is the numerical integration function in use for the L^2–projection presented here. Without this optimization the vector $\boldsymbol{b}$ could not be calculated at all.

[20] The divergence of this and following fields is calculated using the postprocessing tool presented in subsection 4.6.2.

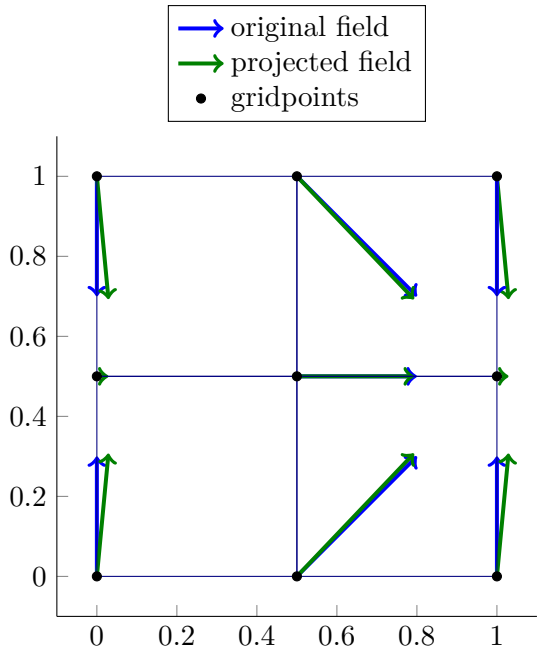

Figure 4.5.4.: a quiver plot of both the original and the projected field

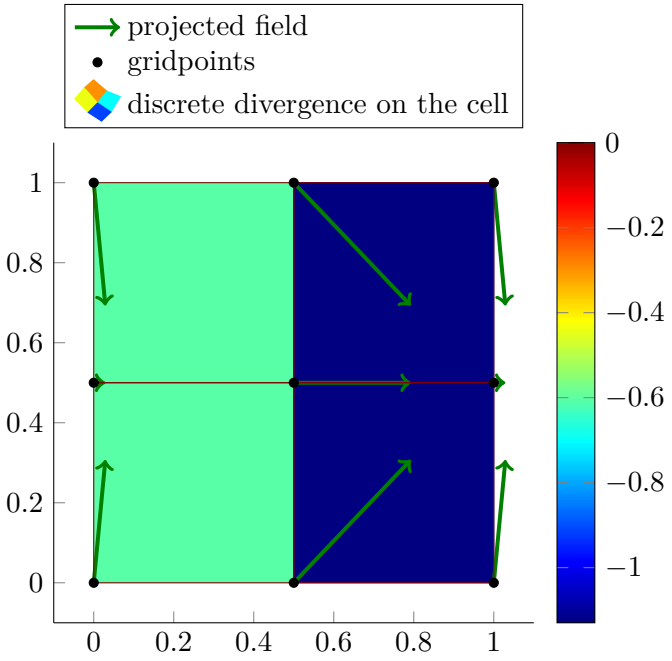

Figure 4.5.5.: the discrete divergence of the projected field

4.5.5. L^2–projection of vector–valued constrained functions in 2D

For the generation of proper IC in *Quickfluid* via an L^2–projection two important constraints have to be added:

- The velocity field has to be divergence–free in order to be able represent a proper IC for incompressible flow and
- applied BCs — especially Dirichlet boundary condition — have to be applied in the right way.

4.5.5.1. Introducing the constraint of incompressibility

First the constraint $\boldsymbol{\nabla} \cdot \hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) = 0$ has to be added to the projection. This leads to the following constrained optimization problem:

$$\begin{aligned} &\text{Find} \inf_{\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) \in S} \left\| \hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) - \boldsymbol{u}_0(\boldsymbol{x}) \right\|_0^2 \\ &\text{while } \boldsymbol{\nabla} \cdot \hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) = 0. \end{aligned}$$

This optimization problem is equivalent to the minimization of the following functional, where the Lagrange multiplier λ is introduced in order to insert the constraint:

$$F\left(\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}), \lambda\right) = \int \left(\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) - \boldsymbol{u}_0(\boldsymbol{x})\right) \cdot \left(\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) - \boldsymbol{u}_0(\boldsymbol{x})\right) \, \mathrm{d}\boldsymbol{x} - \int \lambda \boldsymbol{\nabla} \cdot \hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}.$$

Variation of $\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})$ and λ leads to a pair of variational equations[21]

$$\int \left(\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) - \boldsymbol{u}_0(\boldsymbol{x})\right) \cdot \delta\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} - \int \lambda \boldsymbol{\nabla} \cdot \delta\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0$$

and

$$\int \delta\lambda \boldsymbol{\nabla} \cdot \hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0.$$

[21] This expression is only valid for the special case with homogeneous Dirichlet boundary condition and without any Neumann boundary condition! For details on the general case and the derivation of this expression see [3].

Like in subsection 4.5.4 substituting $\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})$ and $\delta\hat{\boldsymbol{u}}_h^0(\boldsymbol{x})$ with its representation using the basis functions $\boldsymbol{\Phi}_i$ leads to

$$\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) = \sum_{i=1}^{2n} a_i \boldsymbol{\Phi}_i$$
$$\delta\hat{\boldsymbol{u}}_h^0(\boldsymbol{x}) = \sum_{i=1}^{2n} b_i \boldsymbol{\Phi}_i.$$

Additionally substituting λ and $\delta\lambda$ with the following basis functions representations,

$$\lambda = \sum_{I=1}^{N} \alpha_I \boldsymbol{\Psi}_I$$

and

$$\delta\lambda = \sum_{J=1}^{N} \beta_J \boldsymbol{\Psi}_J,$$

where N is the number of cells and $\boldsymbol{\Psi}_I$ are constant basis functionsOne may notice that the same representation is used for the Lagrange multiplier λ, which has already been used for the pressure in the previous chapters. This explains, why the pressure in incompressible flow is often referred as a Lagrange multiplier., lead to the two modified variational equations

$$\int \left[\left(\sum_{i=1}^{2n} a_i \boldsymbol{\Phi}_i - \boldsymbol{u}_0(\boldsymbol{x}) \right) \cdot \left(\sum_{j=1}^{2n} b_j \boldsymbol{\Phi}_j \right) \right] \mathrm{d}\boldsymbol{x} - \int \left[\left(\sum_{I=1}^{N} \alpha_I \boldsymbol{\Psi}_I \right) \boldsymbol{\nabla} \cdot \left(\sum_{j=1}^{2n} b_j \boldsymbol{\Phi}_j \right) \right] \mathrm{d}\boldsymbol{x} = 0$$

and

$$\int \left[\left(\sum_{J=1}^{N} \beta_J \boldsymbol{\Psi}_J \right) \boldsymbol{\nabla} \cdot \left(\sum_{i=1}^{2n} a_i \boldsymbol{\Phi}_i \right) \right] \mathrm{d}\boldsymbol{x} = 0.$$

Again the problem is modified by substituting a_i with $(\boldsymbol{u}_h)_i$ and α_I with λ_I. Additionally taking into consideration, that the variational equation has to be true for every b_j, β_J leads to the following system:

$$\sum_{i=1}^{2n} (\boldsymbol{u}_h)_i \int \boldsymbol{\Phi}_i \boldsymbol{\Phi}_j \, \mathrm{d}\boldsymbol{x} - \sum_{I=1}^{N} \lambda_I \int \boldsymbol{\Psi}_I \boldsymbol{\nabla} \cdot \boldsymbol{\Phi}_j \, \mathrm{d}\boldsymbol{x} = \int \boldsymbol{u}_0(\boldsymbol{x}) \boldsymbol{\Phi}_j \, \mathrm{d}\boldsymbol{x} \quad \text{for } j = 1, 2, ...2n$$

and

$$\sum_{i=1}^{2n} (\boldsymbol{u}_h)_i \int \boldsymbol{\Psi}_J \boldsymbol{\nabla} \cdot \boldsymbol{\Phi}_i \, \mathrm{d}\boldsymbol{x} = 0 \qquad \text{for } J = 1, 2, ...N.$$

By using the mass matrix $(A)_{ij} = \int \boldsymbol{\Phi}_i \boldsymbol{\Phi}_j \, \mathrm{d}\boldsymbol{x}, A \in \mathbb{R}^{2n \times 2n}$, the matrix discretizing the divergence operator $M \in \mathbb{R}^{N \times 2n}$ and the load vector $(\boldsymbol{b})_i = \int \boldsymbol{u}_0 (\boldsymbol{x}) \boldsymbol{\Phi}_i \, \mathrm{d}\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^{2n}$ the following system of equations in matrix–vector–notation can be set up:

$$\begin{aligned} A\boldsymbol{u}_h + M^T \boldsymbol{\lambda}_h &= \boldsymbol{b} \\ M\boldsymbol{u}_h &= \boldsymbol{0}. \end{aligned}$$

$\boldsymbol{u}_h \in \mathbb{R}^{2n}$ is the already known vector holding the weights for the basis functions $\boldsymbol{\Phi}_i$ and $\lambda_h \in \mathbb{R}^N$ is the vector holding the Lagrange multipliers.

For the implementation the composed system has been set up in order to solve the whole problem in one step. Therefore

- the new matrix S has been assembled,
- the solution vector $\boldsymbol{w}$, which is holding the weights and Lagrange multipliers, has been composed and
- the right–hand side $\boldsymbol{c}$ which holds the load vector and many zeroes due to incompressibility has been introduces.

$$S\boldsymbol{w} = \left(\begin{array}{c|c} A & M^T \\ \hline M & 0 \end{array}\right) \left(\begin{array}{c} \boldsymbol{u}_h \\ \hline \boldsymbol{\lambda}_h \end{array}\right) = \left(\begin{array}{c} \boldsymbol{b} \\ \hline \boldsymbol{0} \end{array}\right) = \hat{\boldsymbol{c}}$$

4.5.5.2. Adding applied boundary conditions

For this thesis only the support of homogeneous Dirichlet boundary condition, Neumann boundary condition and slip boundary condition is needed. For Neumann boundary condition nothing has to be done at all and a slip boundary condition is just a composition of Neumann boundary condition in the one and Dirichlet boundary condition in the other direction. Therefore only the implementation of Dirichlet boundary condition is necessary and for homogeneous Dirichlet boundary condition this is very easy and straightforward: After setting up the composed system matrix S, a zero as the solution for the weights corresponding with homogeneous Dirichlet nodes has to be enforced. This can be achieved by setting the corresponding diagonal entry in S to 1 and all other entries in the same line to 0. The corresponding entry in the right–hand side $\boldsymbol{c}$ — of

course — has to be set to zero as well.

$$\left(\begin{array}{ccccc|ccc} A_{1,1} & \dots & A_{1,i} & \dots & A_{1,2n} & \left(M^T\right)_{1,1} & \dots & \left(M^T\right)_{1,N} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ A_{i,1} & \dots & A_{i,i} & \dots & A_{i,2n} & \left(M^T\right)_{i,1} & \dots & \left(M^T\right)_{i,N} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ A_{2n,1} & \dots & A_{2n,i} & \dots & A_{2n,2n} & \left(M^T\right)_{2n,1} & \dots & \left(M^T\right)_{2n,N} \\ \hline M_{1,1} & \dots & M_{1,i} & \dots & M_{1,2n} & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ M_{N,1} & \dots & M_{N,i} & \dots & M_{N,2n} & 0 & \dots & 0 \end{array}\right) \left(\begin{array}{c} (\boldsymbol{u}_h)_1 \\ \vdots \\ (\boldsymbol{u}_h)_i \\ \vdots \\ (\boldsymbol{u}_h)_{2n} \\ \hline (\boldsymbol{\lambda}_h)_1 \\ \vdots \\ (\boldsymbol{\lambda}_h)_N \end{array}\right) = \left(\begin{array}{c} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_{2n} \\ \hline 0 \\ \vdots \\ 0 \end{array}\right).$$

When applying Dirichlet boundary condition in x–direction on the node with index i[22] this system of equations is transformed into the following one:

$$\left(\begin{array}{ccccc|ccc} A_{1,1} & \dots & 0 & \dots & A_{1,2n} & \left(M^T\right)_{1,1} & \dots & \left(M^T\right)_{1,N} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ A_{2n,1} & \dots & 0 & \dots & A_{2n,2n} & \left(M^T\right)_{2n,1} & \dots & \left(M^T\right)_{2n,N} \\ \hline M_{1,1} & \dots & M_{1,i} & \dots & M_{1,2n} & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ M_{N,1} & \dots & M_{N,i} & \dots & M_{N,2n} & 0 & \dots & 0 \end{array}\right) \left(\begin{array}{c} (\boldsymbol{u}_h)_1 \\ \vdots \\ (\boldsymbol{u}_h)_i \\ \vdots \\ (\boldsymbol{u}_h)_{2n} \\ \hline (\boldsymbol{\lambda}_h)_1 \\ \vdots \\ (\boldsymbol{\lambda}_h)_N \end{array}\right) = \left(\begin{array}{c} b_1 \\ \vdots \\ 0 \\ \vdots \\ b_{2n} \\ \hline 0 \\ \vdots \\ 0 \end{array}\right).$$

4.5.5.3. Numerical Examples

First the effect of enforcing the incompressibility constraint is demonstrated by considering the field already known from the previous chapter:

$$\boldsymbol{u}_0(\boldsymbol{x}) = \begin{pmatrix} \sin(\pi x) \\ \cos(\pi y) \end{pmatrix}$$

This field will be projected on a 3×3 grid on the domain $[0,1]^2$. This time incompressibility will be claimed, but none Dirichlet boundary condition will be applied.

[22] For y–direction use $i+n$ instead of i.

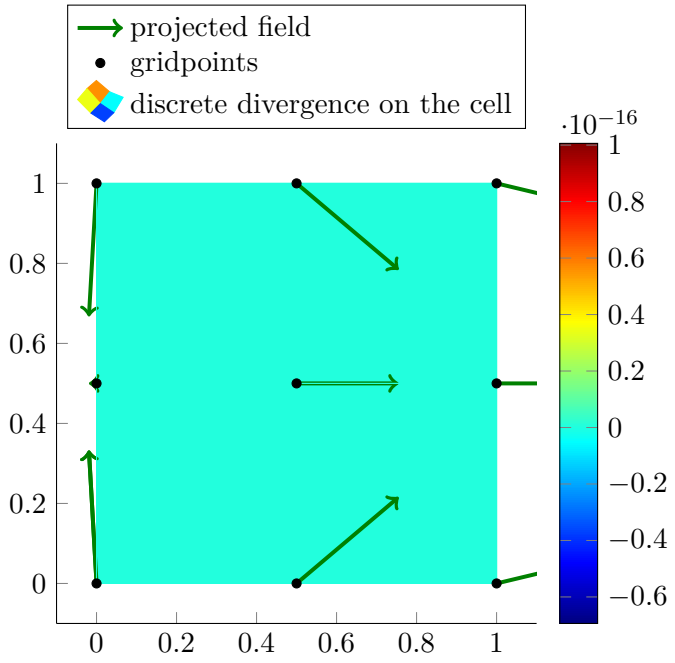

Figure 4.5.6.: the discrete divergence of the projected field with incompressibility constraint

The resulting field is shown in Figure 4.5.6. One will see that the resulting field (in Figure 4.5.1 the corresponding function is $\boldsymbol{v}_{h,J}^0 \in J^h$)is indeed discretely divergence–free, even if the original field $\boldsymbol{u}_0(\boldsymbol{x})$ is not divergence–free:

$$\boldsymbol{\nabla} \cdot (\boldsymbol{u}_0(\boldsymbol{x})) = \frac{\partial \sin(\pi x)}{\partial x} + \frac{\partial \cos(\pi y)}{\partial y} = \cos(\pi x)\,\pi - \sin(\pi y)\,\pi \neq 0$$

In order to demonstrate the effect of enforcing different boundary conditions the L^2–projection of the velocity field

$$\boldsymbol{u}_0(\boldsymbol{x}) = \begin{pmatrix} \sin(\pi x)\sin(\pi y) \\ \sin(\pi x)\sin(\pi y) \end{pmatrix}$$

will be calculated for three different types of BC:

- No BC see Figure 4.5.7
- Slip boundary condition see Figure 4.5.8
- Dirichlet boundary condition see Figure 4.5.9

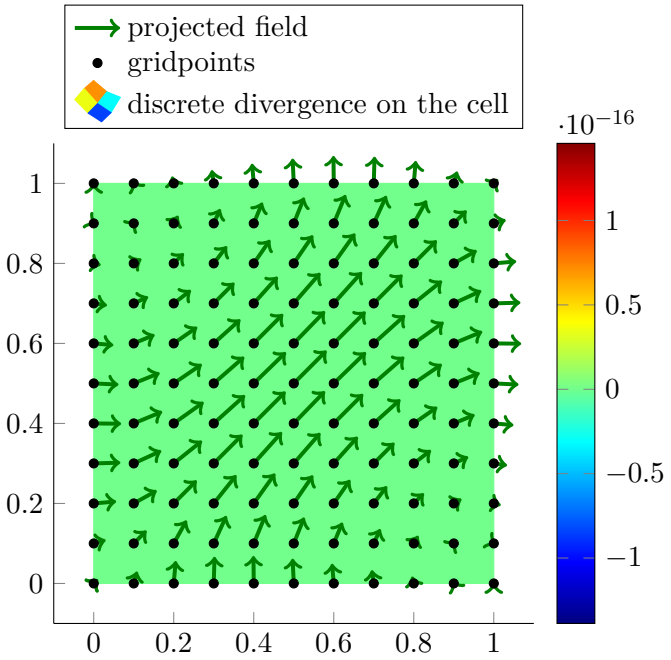

Figure 4.5.7.: the discrete divergence of the projected field with incompressibility constraint and no BC applied

The projections are calculated on a 11×11 grid on the domain $[0, 1]^2$.

All three projections have in common that the divergence is equal to a numerical zero. The differences in the resulting projections are due to the different applied BCs and the characteristic behaviour:

- In Figure 4.5.7 one sees that no special conditions are applied on the wall.
- In Figure 4.5.8 one sees that on the boundary only tangential and no normal velocities occur due to the slip boundary condition.
- In Figure 4.5.9 one sees that on the boundary neither tangential nor normal velocities occur due to the Dirichlet boundary condition[23].

Finally one should note that all L^2–projection in this section have been calculated using pagoda basis functions. L^2–projections using divergence–free basis functions have also been calculated and the implementation is simple, because one only has to modify the local matrices used for the L^2–projection.

[23] Dirichlet boundary condition are often also called a no–slip–BC.

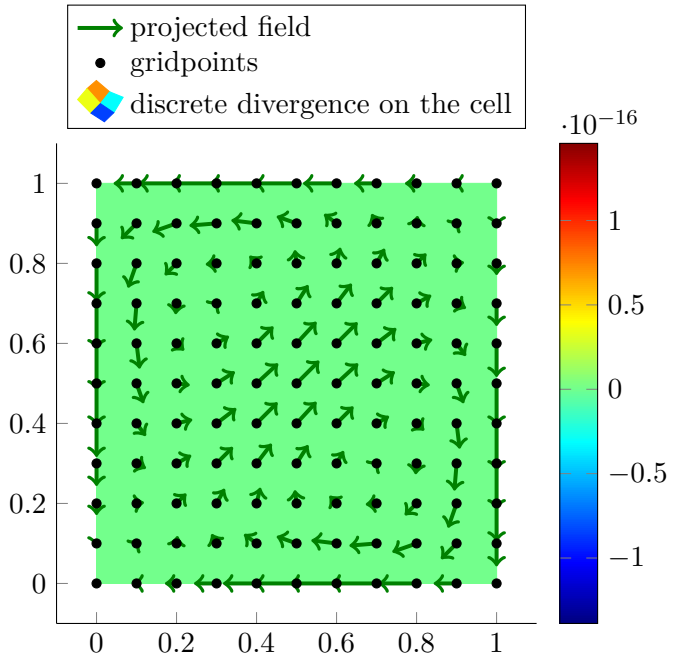

Figure 4.5.8.: the discrete divergence of the projected field with incompressibility constraint and slip boundary condition applied

4.6. Postprocessing tools

4.6.1. Visualisation of boundary conditions

In chapter 5 many different boundary conditions will be faced. For visualizing the boundary conditions, which have been presented in section 3.2 and have been implemented in section 4.4, in a compact way and for debugging of the code, a visualisation tool for boundary conditions has been implemented.

Quickfluid supports the following boundary conditions:

- Outlet boundary condition
- Inlet boundary condition
- Slip boundary condition
- Dirichlet boundary condition

For the different BCs, nodes as well as cells have to be treated in the proper way. *Quick-*

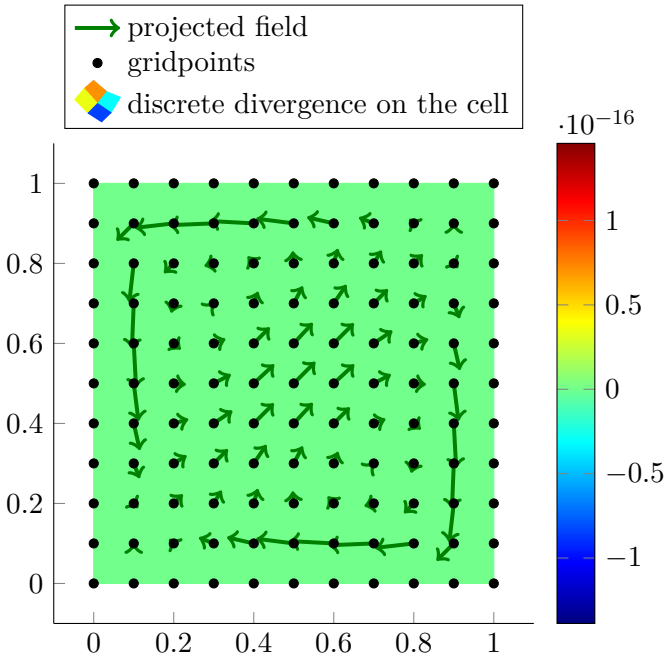

Figure 4.5.9.: the discrete divergence of the projected field with incompressibility constraint and Dirichlet boundary condition applied

fluid saves the indices of nodes and cells which are constrained into different lists shown in Table 4.2. The visualisation tool automatically generates a legend and marks constrained nodes and cells as well as free nodes. Furthermore for outlet and slip cells their position is indicated by text markers[24] because corner outlet cells do behave differently then outlet cells at the right hand side of the domain. For example Figure 5.4.1 has been generated using the visualisation tool.

4.6.2. Visualisation of discrete divergence

A visualisation tool for the discrete divergence of each cell in the simulation domain has been implemented. This tool has been particularly useful when developing the algorithm of the L^2–projection and for analysis of computational results.

[24] RHS (LHS, TOP, DWN) indicates a cell at the right–hand side (left–hand side, top, bottom) boundary of the domain and RU (RD, LU, LD) a cell at the upper–right (bottom–right, upper–left, bottom–left) corner of the domain.

[25] This information is important for the calculation of forces acting on a wall.

dirichletNode	both velocity components of the node are given
slipNode	only the normal velocity component of the node is given
freeNode	no velocity component of the node is given
wallNode	marks a node belonging to a wall[25]
inletNode	both velocity components of the node are given and fluid is entering the domain at this node
outletNode	velocity is free on this node and fluid leaves the domain at this node
slipCell	cell is part of a slip–wall boundary
outletCell	cell is part of an outlet boundary

Table 4.2.: BCs which are shown by the visualization tool for BC

The discrete divergence of one cell can be calculated by using the divergence theorem like it has been explained in subsection 3.1.3. By using the assembly routine described in subsection 4.3.5 one can generate the global matrix M from the given local matrix M_{loc}. Now the divergence of a discrete velocity field $\boldsymbol{\nabla} \cdot \boldsymbol{u}_h$ can be calculated easily by evaluating the following matrix vector product:

$$\boldsymbol{\nabla} \cdot \boldsymbol{u}_h = M \boldsymbol{u}_h.$$

The resulting vector holds the divergence of each cell in lexicographical order. The visualisation of the divergence of each cell is realized through a color plot. The velocities on each node are also shown. See for an example.

5. Numerical Test Scenarios

5.1. General remarks

5.1.1. Choice of the test scenarios

The majority of the collection of the following test scenarios has been borrowed from the PhD–thesis by J.Evans, who is working on divergence–conforming B–spline discretizations for the Navier–Stokes equations. In [2] these B–spline discretizations are developed and tested. In order to gain insight into divergence–free basis functions, *Quickfluid* is applied to the same scenarios. The main objective is to find similar advantages with respect to stability and convergence comparing divergence–free basis functions to pagoda basis functions like J.Evans found when comparing divergence–conforming B–splines to a standard FEM approach.

5.1.2. Used timestep–size

In order to guarantee stability due to the time discretization, the criterion for setting the size of one timestep τ has been taken from [5].

$$\tau \leq \sigma \min\left(\delta t_{\text{visc}}, \delta t_u, \delta t_v\right)$$

where

$$\sigma = 0.8$$

and

$$\delta t_{\text{visc}} = \frac{A_1 h^2}{4\nu}, \delta t_u = \frac{A_2 h}{2u_{\max}}, \delta t_v = \frac{A_2 h}{2v_{\max}}$$

with

$$A_1 = A_2 = 0.5.$$

In [5] the criterion is applied to a time–adaptive approach. This adaptivity is not supported in *Quickfluid*; therefore the maximum velocities $u_{\max}$ and $v_{\max}$ are determined in a preprocessing step, which is a very short simulation of the scenario with a very small

(not ideal!) timestep, which guarantees stability for sure.

5.1.3. Calculation of the error

In this thesis velocity errors are calculated using the discrete L^2–norm:

$$\|\boldsymbol{u}_h\|_{L^2} = \sqrt{\frac{1}{N}\sum_{i}^{N} u_i^2 + v_i^2}$$

Where N is the number of gridpoints and u_i and v_i are the x–component and the y–component of the velocity on the i–th gridpoint.

When calculating the L^2–error e one just has to evaluate the L^2–norm of the error:

$$e = \|\boldsymbol{u}_h - \boldsymbol{u}\|_{L^2}$$

where $\boldsymbol{u}$ is the reference velocity field, which is often an analytical solution evaluated on the gridpoints or a L^2–projected analytical field.

5.2. Curve

The first test scenario is just a simple example where fluid is streaming into the domain over the upper boundary with the velocity $\boldsymbol{v}_D = (\ 0 \quad -1\ )^T$ and exiting the domain over the left boundary with the velocity $\boldsymbol{v}_D = (\ -1 \quad 0\ )^T$. The remaining boundaries are set to homogeneous Dirichlet boundary condition with $\boldsymbol{v}_D = (\ 0 \quad 0\ )^T$.

5.2.1. Implementation of the scenario

This scenario was just the first try in order to get in touch with the program *Quickfluid* and the implementation of Dirichlet boundary condition. See figure Figure 5.2.1 for the applied boundary conditions. The calculations have been done using divergence–free basis functions. The following parameters are used:

- length in x–direction: $L = 1$
- length in y–direction: $H = 1$
- inlet velocity: $U = 1$
- Reynolds number: $\mathrm{Re} = \frac{UH}{\nu} = 1$

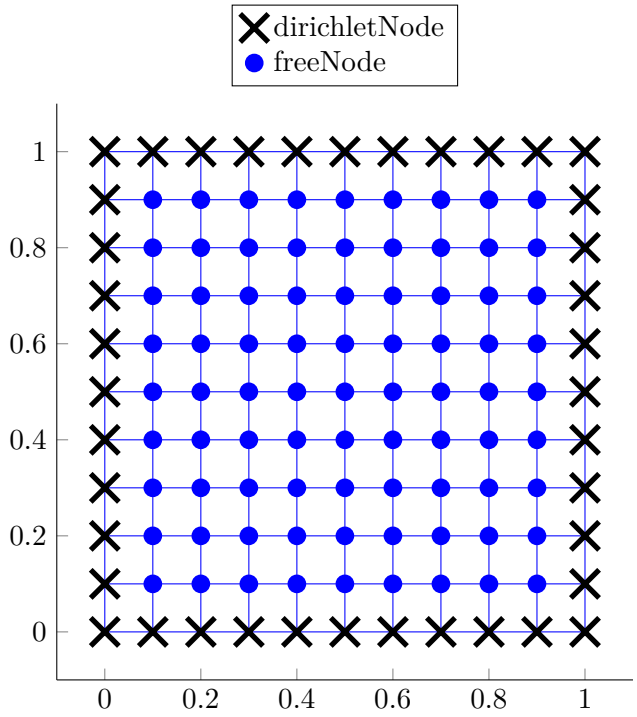

Figure 5.2.1.: applied BC for the test scenario curve

- grid size: 10×10 cells
- timestep size: $\tau = \sigma \min\left(\delta t_{\text{visc}}, \delta t_u, \delta t_v\right)$
- simulation time: $T = 0.04$

5.2.2. Interpretation of the results

The scenario produces a steady velocity field shown in Figure 5.2.2 and the pressure field shown in Figure 5.2.3. The divergence of the field is on the scale of the numerical error on the whole domain and can therefore be considered equal to zero (also shown in Figure 5.2.2).

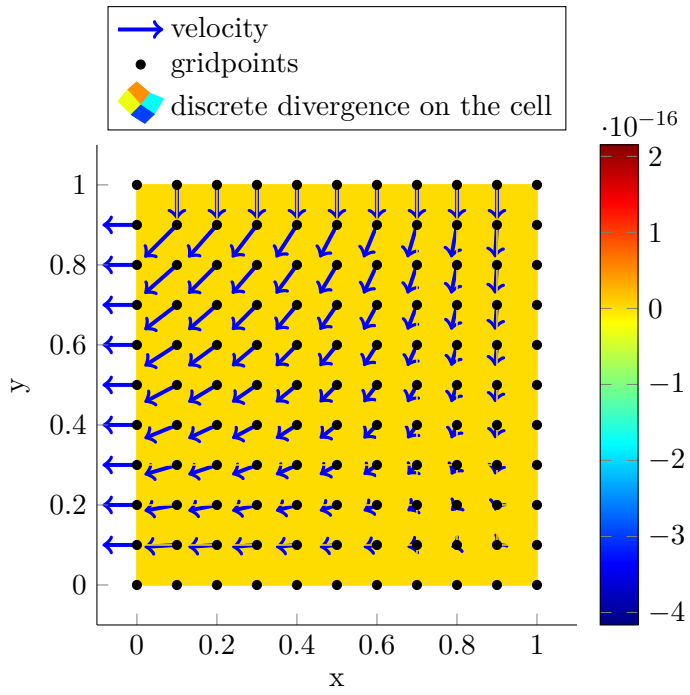

Figure 5.2.2.: resulting velocity field (with divergence) for the test scenario curve

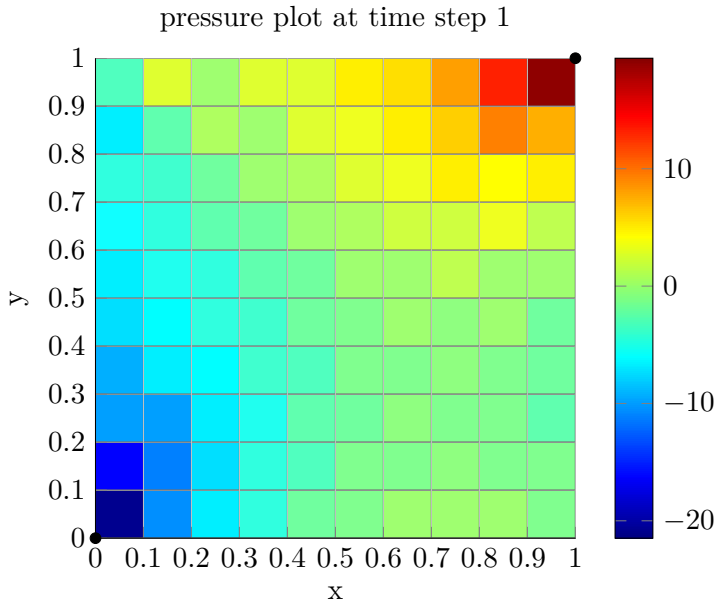

Figure 5.2.3.: resulting pressure field for the test scenario curve

5.3. Kovasznay flow

This test scenario is borrowed from [2] and is used to simulate the flow behind a infinite two–dimensional grid.

5.3.1. Implementation of the scenario

For this flow problem there exists an analytical solution[1]

$$\boldsymbol{u}(x,y) = \begin{pmatrix} 1 - e^{\lambda x}\cos(2\pi\left(y - \frac{1}{2}\right)) \\ \frac{\lambda}{2\pi}e^{\lambda x}\sin(2\pi\left(y - \frac{1}{2}\right)) \end{pmatrix},$$
$$p(x,y) = \frac{1 - e^{2\lambda x}}{2}$$

with

$$\lambda = \frac{\mathrm{Re}}{2} - \sqrt{\frac{\mathrm{Re}^2}{4} + 4\pi^2}.$$

Of course $\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0$ holds.

In *Quickfluid* Kovasznay flow is simulated using the following parameters[2] for both types of basis functions:

- length in x–direction: $L = 1$
- length in y–direction: $H = 1$
- inlet velocity: $U = 1$
- Reynolds number: $\mathrm{Re} = \frac{UH}{\nu} = 40$
- grid size: 10×10 cells
- timestep size: $\tau = \sigma \min(\delta t_{\mathrm{visc}}, \delta t_u, \delta t_v)$
- simulation time: $T = 2$

In *Quickfluid* the Reynolds number can be set by specifying the dynamic viscosity $\eta = \nu\rho$.

[1] Different to [2] Kovasznay flow is simulated on the domain $\Omega = [0,1]^2$ and not on $\Omega = [0,1] \times \left[-\frac{1}{2}, \frac{1}{2}\right]$. Therefore the analytical solution is also shifted by $-\frac{1}{2}$ in y–direction.

[2] Again these parameters are borrowed from [2].

Due to the given parameter the dynamic viscosity has to be equal to

$$\eta = \nu\rho = \frac{UH\rho}{\mathrm{Re}}.$$

Furthermore ρ is set to 1 which gives

$$\eta = \frac{1}{40}.$$

The necessary BCs are shown in Figure 5.3.1 for a 5×5 grid. Note that the BCs can be implemented easily, because they are very similar to the boundary conditions from channel flow[3].

5.3.2. Interpretation of the results

Kovasznay flow is often referred as a convergence test for FEM discretizations and therefore the convergence towards the analytical solution with ongoing time using pagoda basis functions and divergence–free basis functions has been compared. The difference between the velocity field from the numerical simulation (using both pagoda basis functions and divergence–free basis functions) and the velocity field from analytical solution is shown Figure 5.3.6 for the simulation time $t \in [0, 2]$. Both numerical simulations show the same convergence towards the analytical solution. The convergence of both basis functions is visually indistinguishable.

The constant offset of approximately 0.04 can be explained that the numerical solution cannot converge towards the analytical solution because it 'lives' in another function space and therefore this constant offset will never vanish. An L^2–projection of the analytical solution onto the discrete 10×10 grid with the applied boundary conditions could be used to eliminate this error, but the implementation of this special L^2–projection with the needed BCs has not been realized.

In Figure 5.3.2 and Figure 5.3.3 the velocity and pressure field at the beginning of the simulation[4] are shown. One will notice that these fields — especially the pressure field — are far from the analytical solution, which would be a linearly decreasing pressure field. In Figure 5.3.4 and Figure 5.3.5 the both fields are shown at the end of the simulation. Especially the linearly decreasing pressure field is indistinguishable from the analytical solution.

[3] The channel flow szenario has been already implemented in [5]. The implementation of Kovasznay flow mainly consisted in recycling existing code.

[4] using divergence–free basis functions

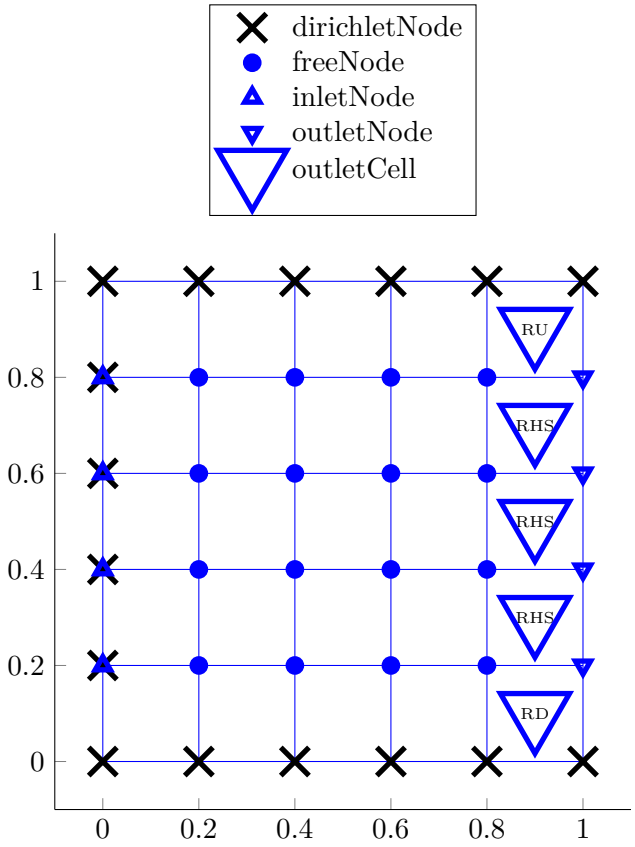

Figure 5.3.1.: applied BC for the test scenario Kovasznay flow

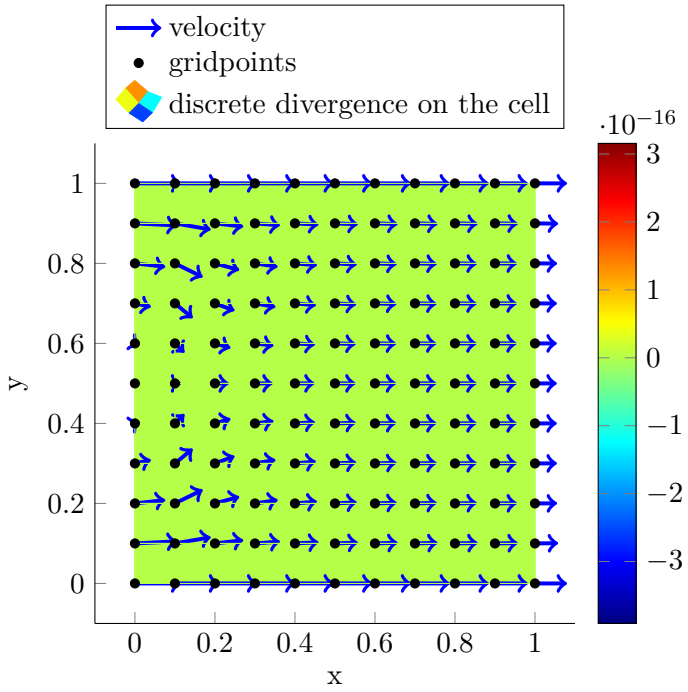

Figure 5.3.2.: resulting velocity field (with divergence) for the test scenario Kovasznay flow at the beginning of the simulation ($T = 0$)

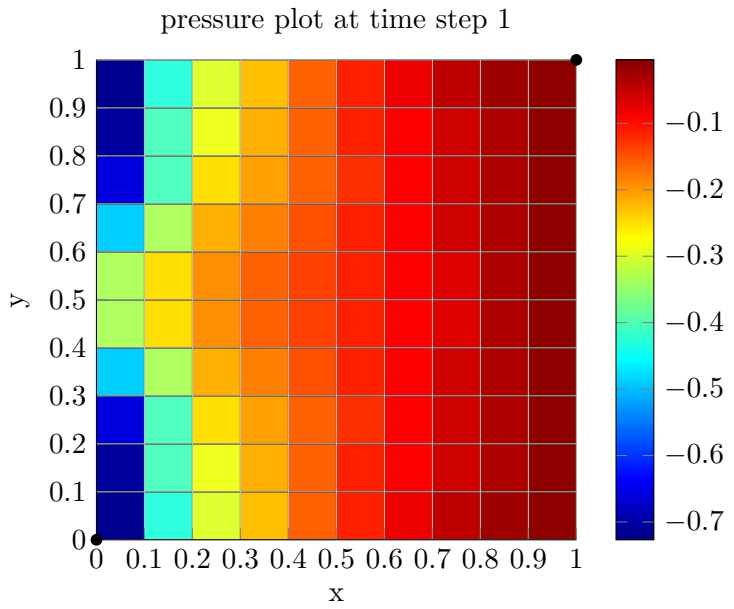

Figure 5.3.3.: resulting pressure field for the test scenario Kovasznay flow at the beginning of the simulation ($T = 0$)

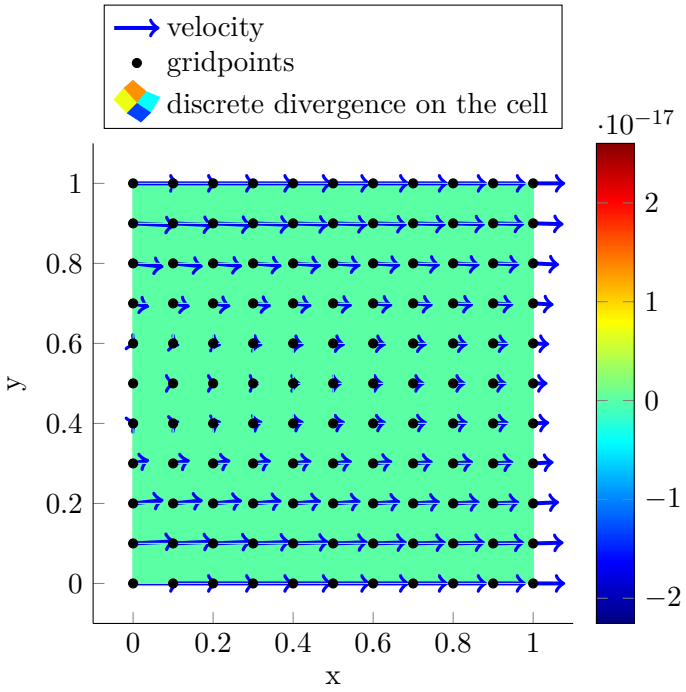

Figure 5.3.4.: resulting velocity field (with divergence) for the test scenario Kovasznay flow at the end of the simulation ($T = 2$)

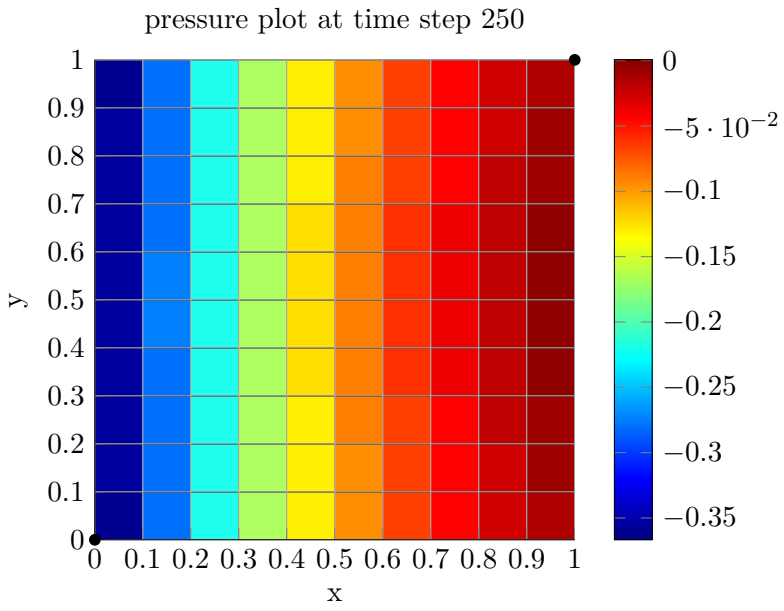

Figure 5.3.5.: resulting pressure field for the test scenario Kovasznay flow at the end of the simulation ($T = 2$)

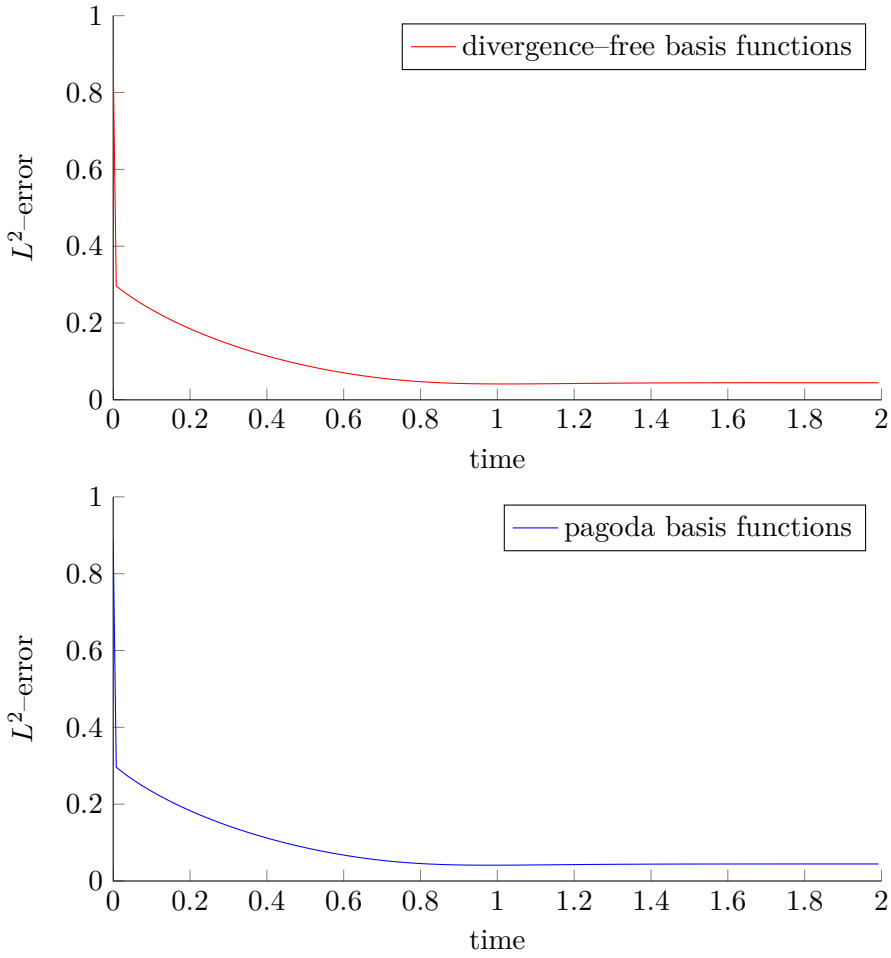

Figure 5.3.6.: convergence of the velocity field from the numerical simulation of the test scenario Kovasznay flow for two different basis functions

5.4. Confined jet impingment

Jet impingment is according to [2] a 'difficult numerical benchmark problem', because 'spurious nonzero divergence' may appear near the stagnation region and therefore the incompressibility constraint is violated at this place. In [2] it has been shown that a discretisation, which is satisfying incompressibility in an exact way, does not show this behaviour and therefore better results can be achieved. Therefore jet impingment is considered to be a test scenario which will lead to a deeper insight into the behaviour of divergence–free basis functions.

5.4.1. Implementation of the scenario

An impinging jet can be simulated easily using the boundary conditions shown in Figure 5.4.1 for a 10×5 grid. In order to make use of the symmetry of the problem — and only simulate half of the domain — a slip boundary condition is introduced at the left hand side of the domain as a symmetry BC. The top and bottom walls are non–slip walls and therefore homogeneous a Dirichlet boundary condition is applied. Only at a certain part of the top boundary there exists an inlet for the fluid with specified inlet velocity. The right–hand side of the domain is an outlet boundary condition, where the fluid exits the domain.

The used parameters are given in [2]:

- length in x–direction: $L = 8$
- length in y–direction: $H = 1$
- size of Inlet: about 10%of L[5]
- inlet velocity: $U = 1$
- Reynolds number: $\mathrm{Re} = \frac{UH}{\nu} = 50$
- grid size: 24×3 cells
- timestep size: $\tau = \sigma \min\left(\delta t_{\mathrm{visc}}, \delta t_u, \delta t_v\right)$
- simulation time: $T = 500$

The computations have been done using pagoda basis functions as well as divergence–free basis functions.

[5] Exact value depends on grid size.

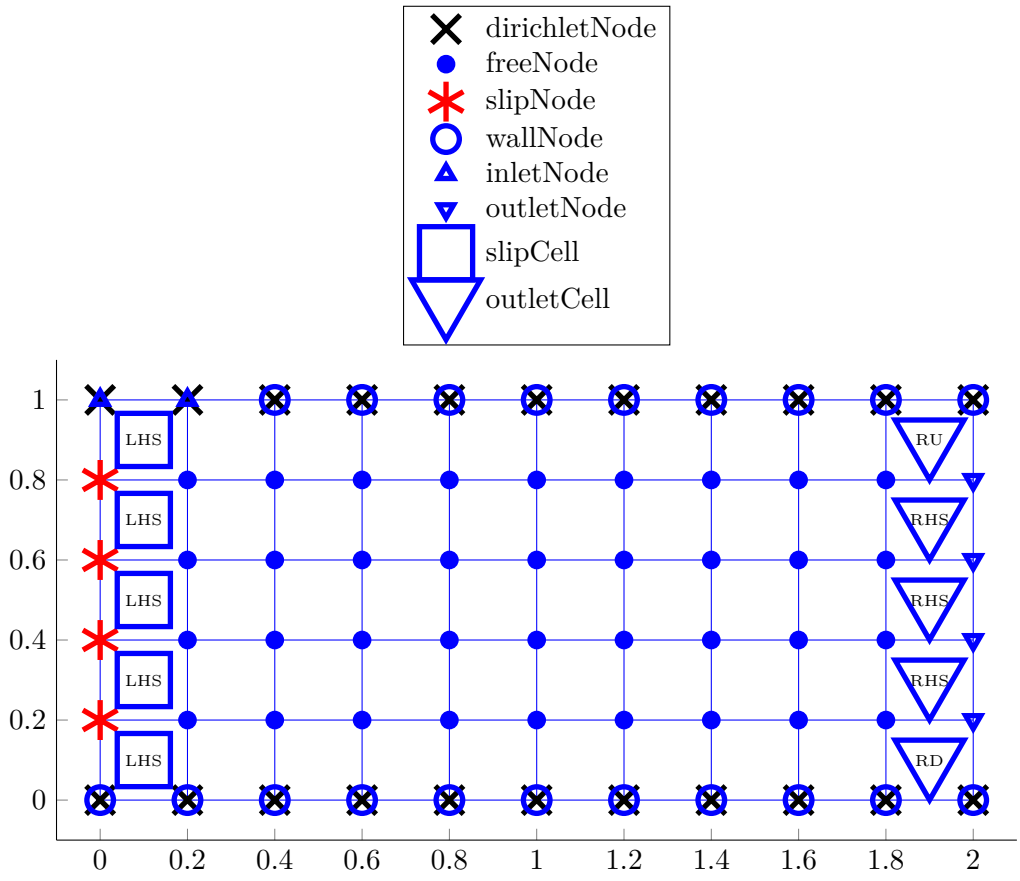

Figure 5.4.1.: boundary conditions of jet impingment

5.4.2. Interpretation of the results

The implementation of slip boundary conditions has been validated by once simulating jet impingment without using slip boundary conditions and once simulating jet impingment with slip boundary conditions. For an implementation without using slip boundary condition, jet impingment is simulated on a 48×3 grid. The whole simulation domain has to be simulated and a symmetric pressure field shown in Figure 5.4.2 develops. By using slip boundary conditions one can realize a symmetry BC and only half of the domain has to be simulated which means only a 24×3 grid is needed. The pressure field of jet impingment using slip boundary condition is shown in Figure 5.4.2. This pressure field cannot be distinguished from the right half of the pressure field in Figure 5.4.3[6]. Thus the implementation of slip boundary condition can be considered to be correct.

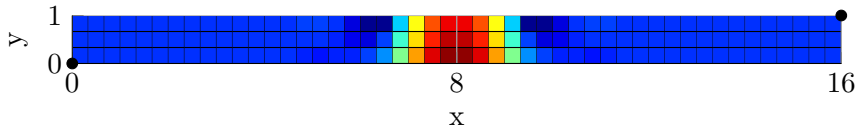

Figure 5.4.2.: resulting pressure field for the test scenario jet impingment without using slip boundary condition

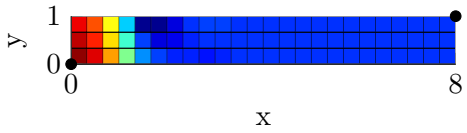

Figure 5.4.3.: resulting pressure field for the test scenario jet impingment with slip boundary condition

A simulation of jet impingment using divergence–free basis functions leads to the velocity field shown in Figure 5.4.4 and the pressure field shown in Figure 5.4.5. The presented field, obtained after a pretty long simulation time of about 500[7], is equal to the stationary field presented in [2]. The corresponding streamlines are shown in Figure 5.4.6.

[6] One should note that the presented pressure field is from a simulation of jet impingment with Re = 10000. It was not possible to stably compute simulations with high Reynolds numbers using *Quickfluid* and therefore (symmetric) oscillations develop which are nice for demonstration of the symmetry BC.

[7] which is equal to 20240 timesteps

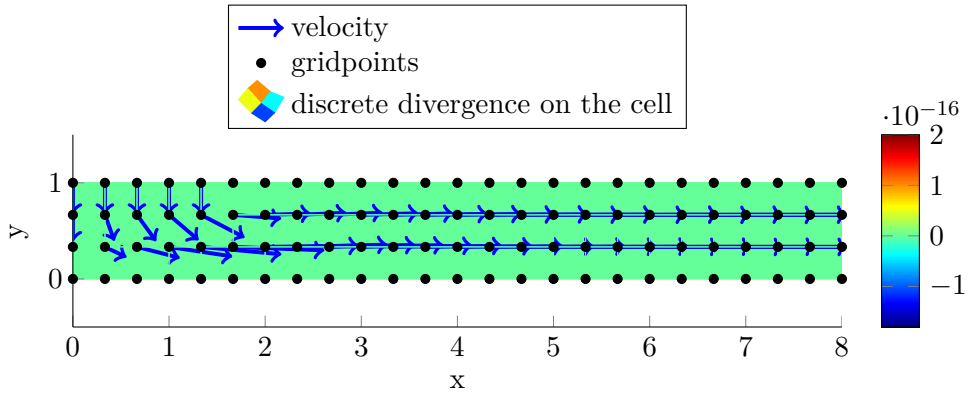

Figure 5.4.4.: resulting velocity field (with divergence) for the test scenario jet impingment

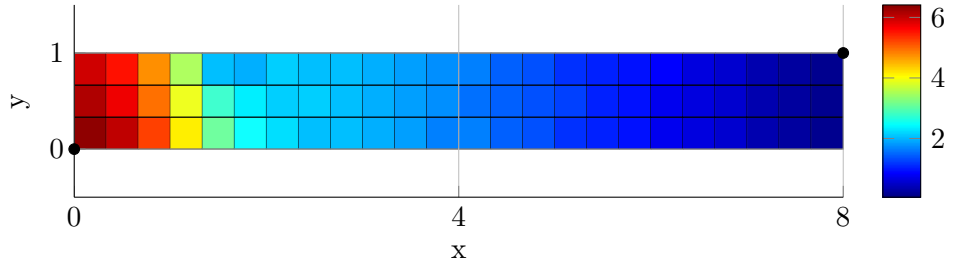

Figure 5.4.5.: resulting pressure field for the test scenario jet impingment

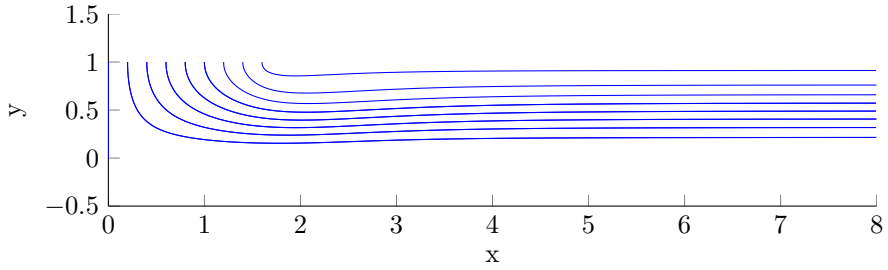

Figure 5.4.6.: resulting streamlines for the test scenario jet impingment

The simulation has also been done for higher Reynolds numbers, but a stable computation for Re $>$ 1000 has not been possible and after some steps oscillations have been noticed. The reason for this erroneous behaviour when trying to deal with high Reynolds numbers will be shortly discussed in chapter 6.

5.5. Taylor–Green vortex flow

Taylor–Green vortex flow is a periodic[8] non–stationary field of vortices. The initial velocity field for Taylor–Green vortex flow is shown in Figure 5.5.1 on the domain $[0, 2\pi]^2$. The following IC is applied:

$$\boldsymbol{u}_0(\boldsymbol{x}) = \begin{pmatrix} \sin(x)\cos(y) \\ -\cos(x)\sin(y) \end{pmatrix}$$

No external forcing is applied to the fluid and therefore the vortices exponentially decay in time only due to viscosity. The analytical solution for Taylor–Green vortex flow reads

$$\boldsymbol{u}(\boldsymbol{x}, t) = \begin{pmatrix} \sin(x)\cos(y) \\ -\cos(x)\sin(y) \end{pmatrix} e^{-2\nu t}$$
$$p(\boldsymbol{x}, t) = \frac{1}{4}\left(\cos(2x) + \cos(2y)\right) e^{-4\nu t}.$$

In [2] it has been shown that conservative B–Splines lead to a stable and physically correct solution while Q_2/Q_1–elements lead to non–physical behaviour and an exponential blow up of the error. This indicates that the use of conservative elements helps to avoid errors originating from an non–physical energy growth which is caused by violating the incompressibility constraint.

One of the main goals in this thesis is, on the one hand, to reproduce this stable behaviour using divergence–free basis functions. On the other hand it should be possible to reproduce the unstable behaviour when using pagoda basis functions. This would show that divergence–free basis functions have the same advantages like conservative B–Splines concerning energy conservation.

5.5.1. Implementation of the scenario

Taylor–Green vortex flow can be implemented using slip boundary condition. These slip boundary conditions can be interpreted as symmetry boundaries and therefore it is sufficient to simulate one vortex in order to represent the infinitely large periodic vortex field. In Figure 5.5.2 the used boundary conditions on a 8×8 grid which discretizes the simulation domain $\Omega = [0, \pi]^2$ are shown. Additionally to a proper implementation of the boundary conditions also the declaration of the initial conditions is a crucial point. In [2] the initial conditions have been implemented using an L^2–projection. This technique has been explained in section 4.5. Therefore the continuous initial condition $\boldsymbol{u}_0(\boldsymbol{x})$ for Taylor–Green vortex flow are projected on the discrete grid using L^2–projection. The

[8] in space!

resulting projected initial condition $\boldsymbol{u}_h^0$ is then just used for the discrete velocity field on the discrete grid.

The remaining parameters are the following:

- length in x–direction: $L = \pi$
- length in y–direction: $H = \pi$
- Reynolds number: $\mathrm{Re} = 100, 1000, 10000$
- grid size: $8 \times 8, 16 \times 16, 32 \times 32$ cells
- timestep size: $\tau = \sigma \min(\delta t_{\mathrm{visc}}, \delta t_u, \delta t_v)$
- simulation time: $T = 5, 50, 500, 5000$

5.5.2. Interpretation of the results

The test scenario Taylor–Green vortex flow has been simulated for a large variety of different Reynolds numbers, grid resolutions and simulation times. The error of the numerical simulation has been estimated by comparing one sample of the numerical solution to the L^2–projected analytical solution at the time corresponding to this sample.

In order to demonstrate the importance of the L^2–projection for the proper calculation of the error, the error for a simulation on a 8×8 grid with $\mathrm{Re} = 100$ and simulation time $T = 5$ and $T = 500$ has once been calculated with a projection of the analytical solution and once without a projection of the analytical solution. The results are shown in Figure 5.5.3 and Figure 5.5.4. It is obvious that the error calculated with an L^2–projection and the error without an L^2–projection differ strongly — especially at the beginning. Therefore an L^2–projection is necessary if one wants to estimate the error in the right way. Of course this procedure is — in general — very time consuming and therefore the (wrong) error calculation without L^2–projection could be used as an estimate, as both error estimates[9] are moving towards the same value with ongoing time.

For the special case of Taylor–Green vortex flow the time dependency of the analytical solution does not lead to the problem that for each step a L^2–projection has to be calculated. The time–dependent part of the solution can be separated from the space–dependent part of the solution and therefore it is sufficient to project the IC and then just add the time–dependent factor. This saves a large amount of time–consuming computations.

[9] with and without L^2–projection

$$\boldsymbol{u}(x,y,t) = e^{-2\nu t}\boldsymbol{u}(x,y,t=0) = e^{-2\nu t}\boldsymbol{u}_0(x,y) \xrightarrow{P_J^h} e^{-2\nu t}\boldsymbol{u}_h^0$$

Secondly the effect of not projecting the initial condition via an L^2–projection will be demonstrated by once plotting the error of a simulation with L^2–projected initial conditions and once without L^2–projected initial conditions (see Figure 5.5.5 and Figure 5.5.6). The simulation is again calculated on a 8×8 grid with $\mathrm{Re} = 100$ and $T = 5$ or $T = 500$. At the beginning the error of the simulation with projected initial conditions stays below the error of the simulation without projected initial conditions. But later on the errors of the both simulations — with and without projected ICs — converge towards each other and finally they are on the same level. In this thesis still a projection is applied because it is on the one hand strongly recommended by [3] and on the other hand the computational effort is low.

Taylor–Green vortex flow has been simulated for different Reynolds numbers. It has been found out that for Reynolds numbers up to 10^3 a stable computation of Taylor–Green vortex flow is possible. For Reynolds numbers bigger than 10^3 it has not been possible to calculate correct results. Probably it is necessary to implement an upwind scheme in order to be able to compute highly convective scenarios with high Reynolds numbers. This additional implementation was not possible in the scope of this thesis.

Finally Taylor–Green vortex flow has also been simulated for different resolutions. For Reynolds numbers where a stable computation is possible a higher resolution does not lead to a better or worse stability. Having a closer look at the error one will see that a higher resolution reduces the error slightly[10] and that divergence–free basis functions generally show a slightly larger error then pagoda basis functions (see Figure 5.5.8). For Reynolds numbers where it has not been possible to obtain a correct result, a massive blow–up of velocity — and thus completely incorrect behaviour — happens even earlier for higher resolutions (see Figure 5.5.9). One should also notice that, if a blow–up happens, the blow–up happens earlier when using divergence–free basis functions than for pagoda basis functions .

Comparing these results to the results in [2] one should always keep in mind that the simulations in this thesis have been calculated for different Reynolds numbers. On the one hand a simulation with $\mathrm{Re} = \infty$ is not possible with the code at hand and computations for high Reynolds numbers are not possible as well. On the other hand [2] does not supply reference solutions for simulations with lower Reynolds numbers, which have been computed in this thesis. Therefore the test scenario Taylor–Green vortex flow leads to an interesting insight into the FEM–code *Quickfluid* and reveals weak points of the code. But one should be aware that only under limitations the results from this thesis can be compared to the results given in [2] and even sometimes no statement is possible.

[10] as one expects when reducing the mesh size and calculating the solution for a finer mesh

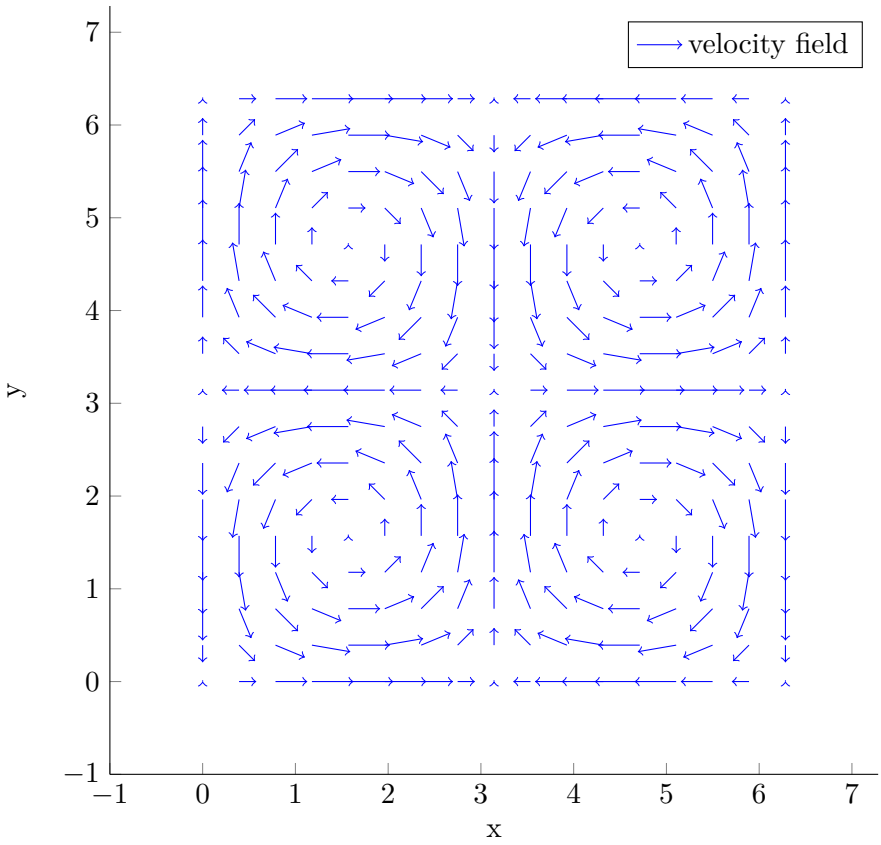

Figure 5.5.1.: analytical initial condition Taylor–Green vortex flow

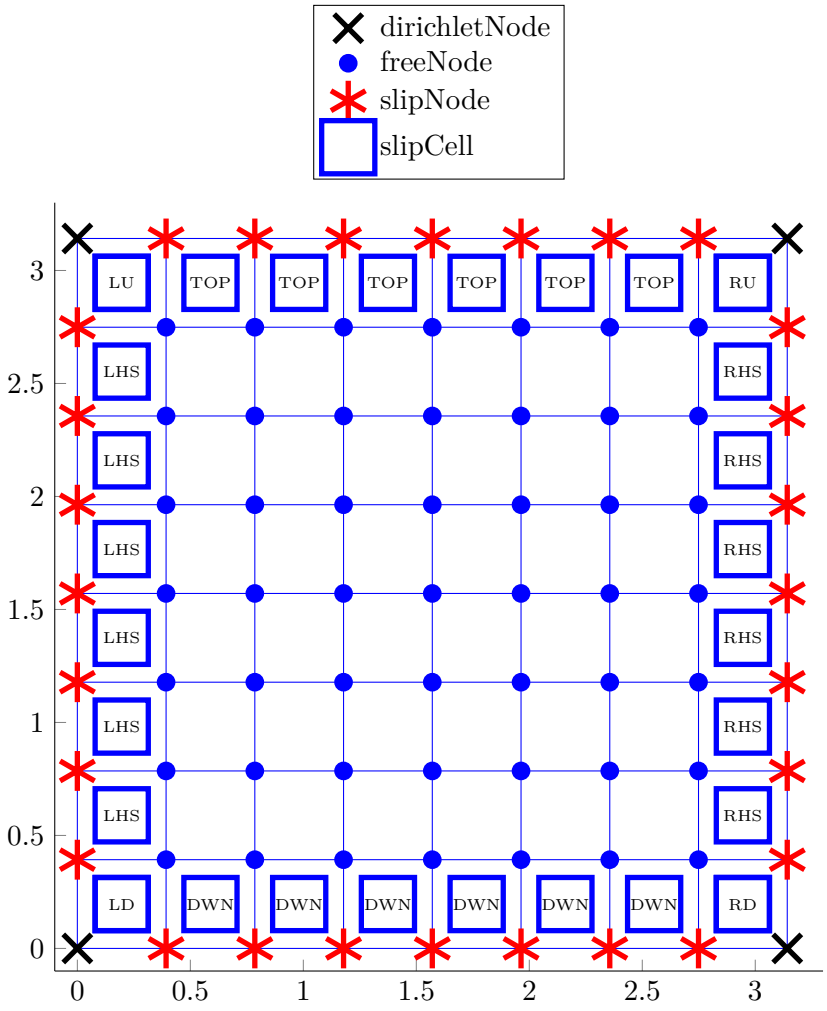

Figure 5.5.2.: boundary conditions of Taylor–Green vortex flow

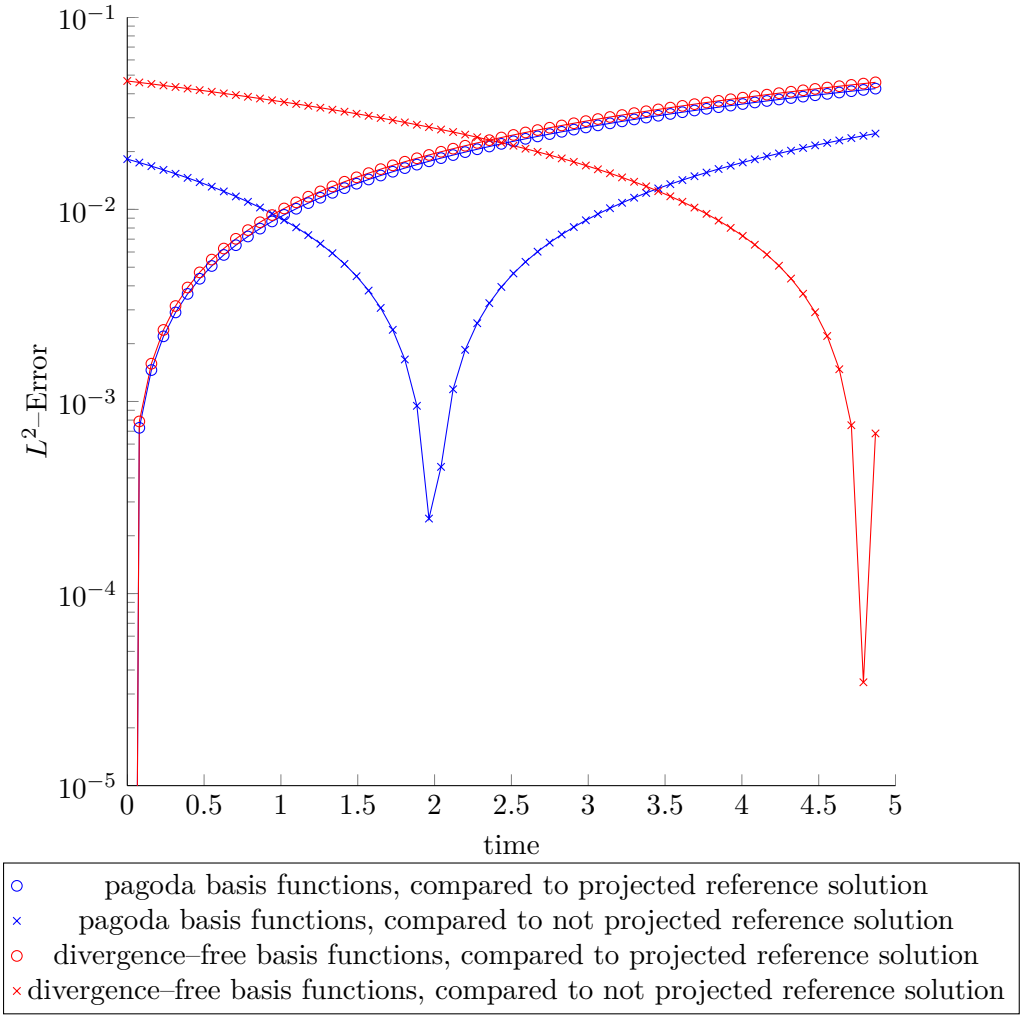

Figure 5.5.3.: error plot of Taylor–Green vortex flow on an 8×8 grid for $T = 5$ and $\mathrm{Re} = 100$, calculating the error with and without L^2–projection of the reference solution

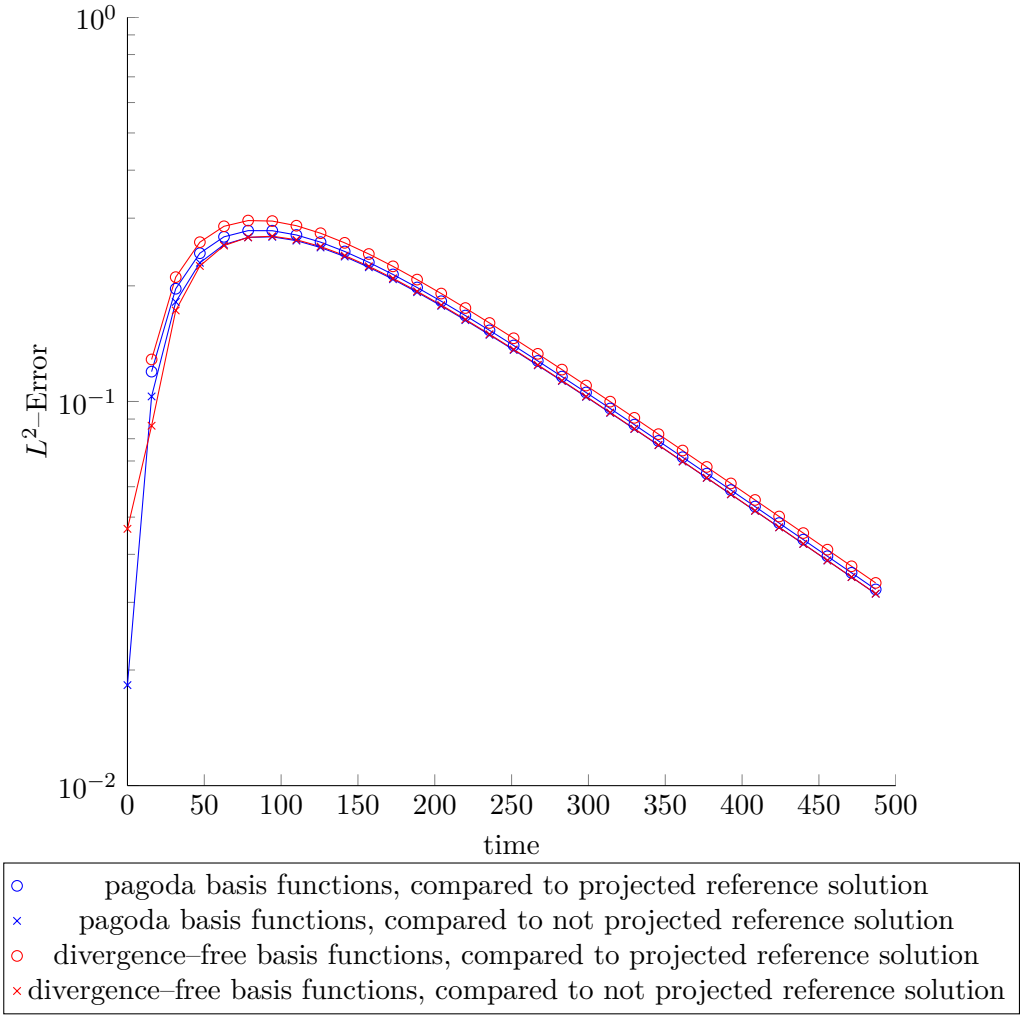

Figure 5.5.4.: error plot of Taylor–Green vortex flow on an 8×8 grid for $T = 500$ and $\mathrm{Re} = 100$, calculating the error with and without L^2–projection of the reference solution

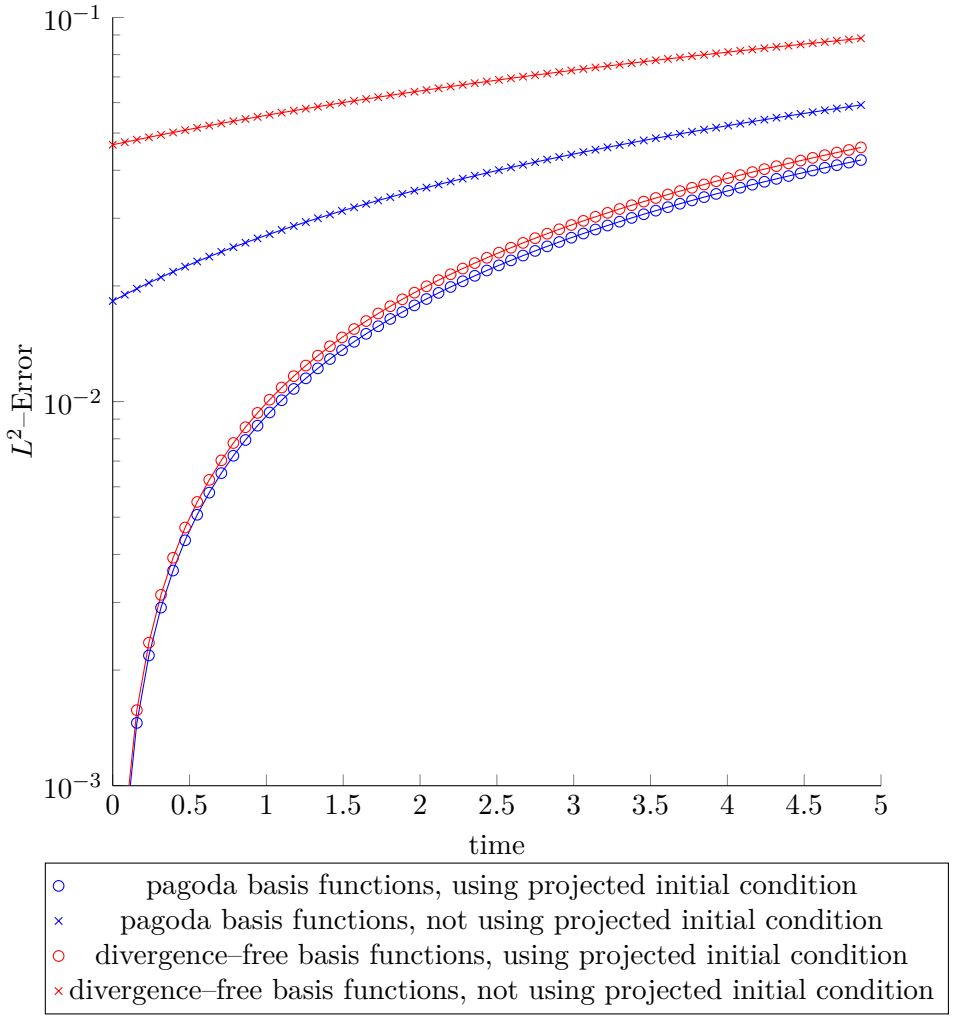

Figure 5.5.5.: error of simulations of Taylor–Green vortex flow on an 8×8 grid with and without L^2–projected initial conditions for $T = 5$ and $\mathrm{Re} = 100$

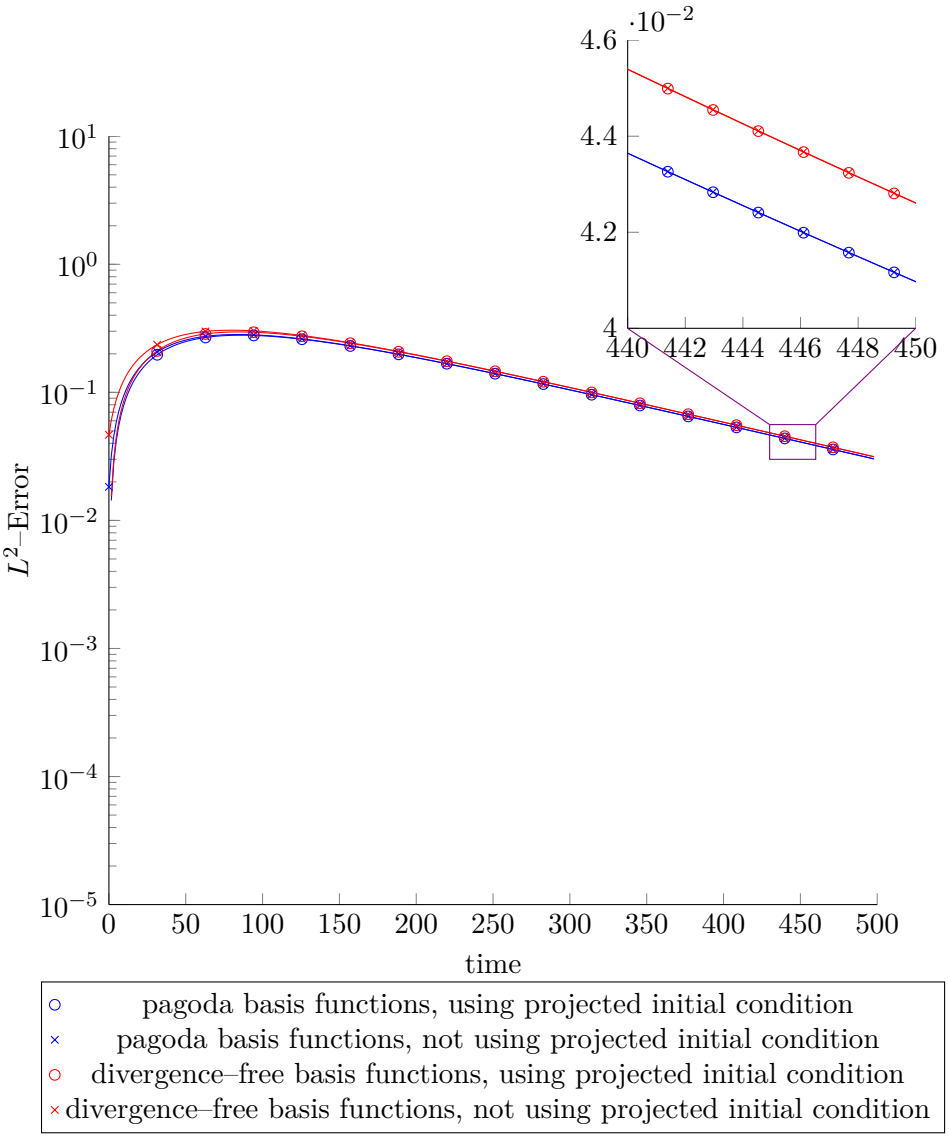

Figure 5.5.6.: error of simulations of Taylor–Green vortex flow on an 8×8 grid with and without L^2–projected initial conditions for $T = 500$ and $\mathrm{Re} = 100$

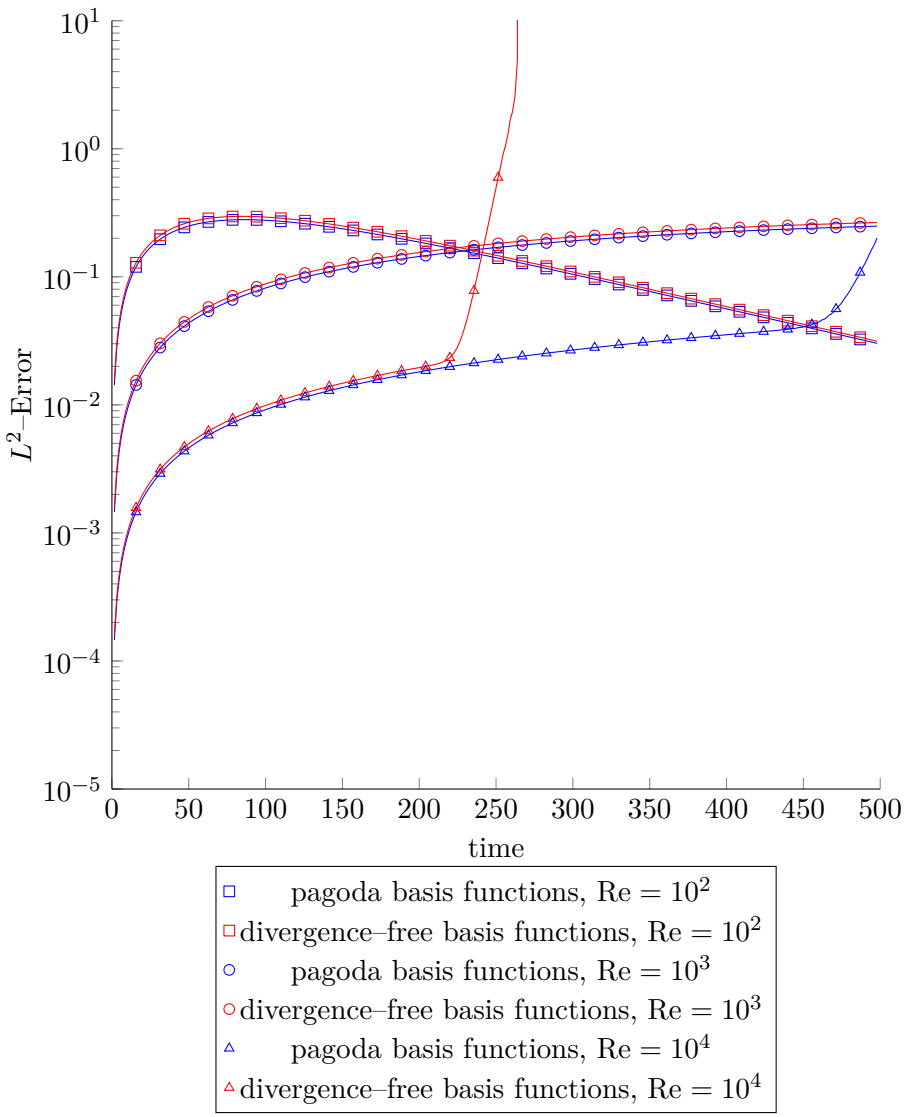

Figure 5.5.7.: error of simulations of Taylor–Green vortex flow on an 8×8 grid for different Reynolds numbers

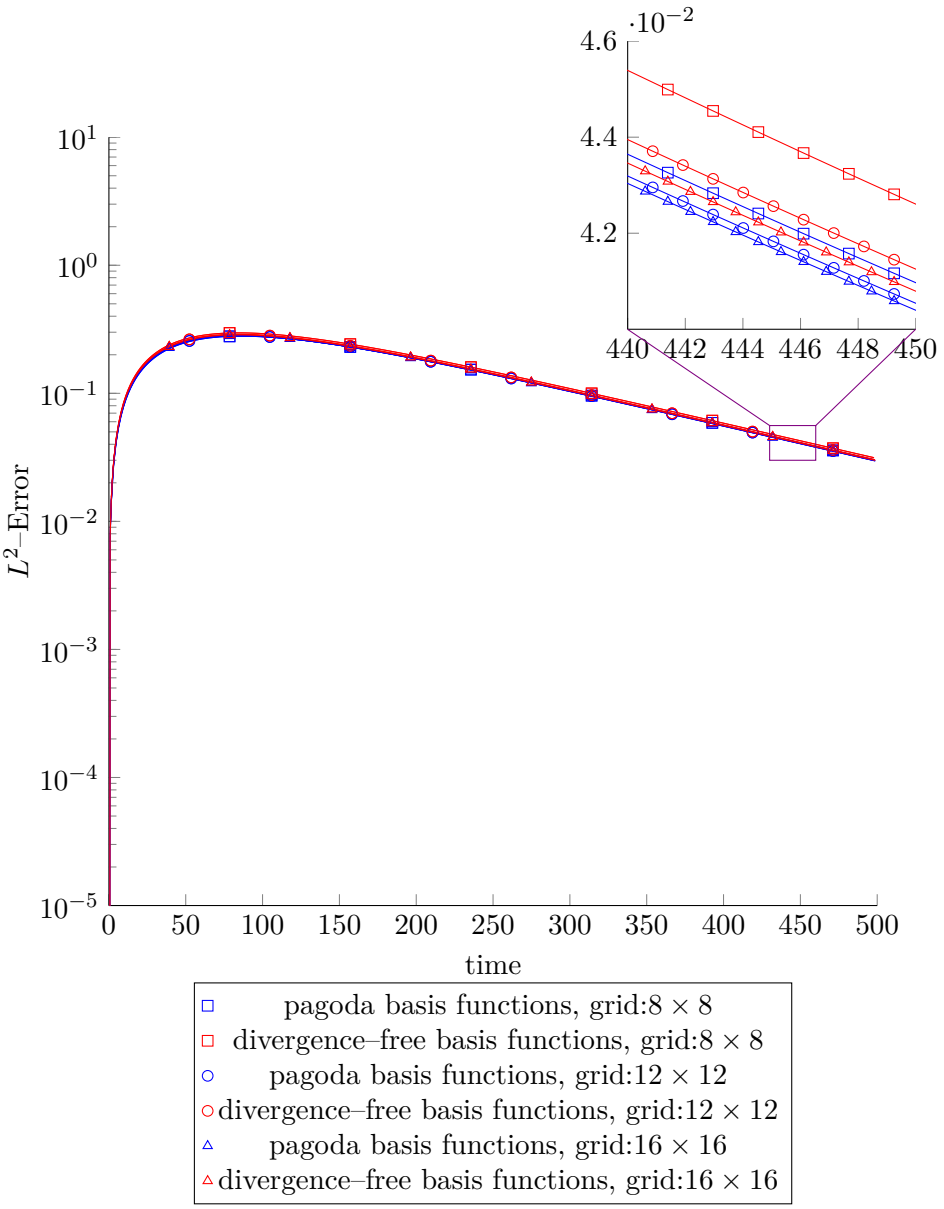

Figure 5.5.8.: error of simulations of Taylor–Green vortex flow for different grid resolutions at Re = 100

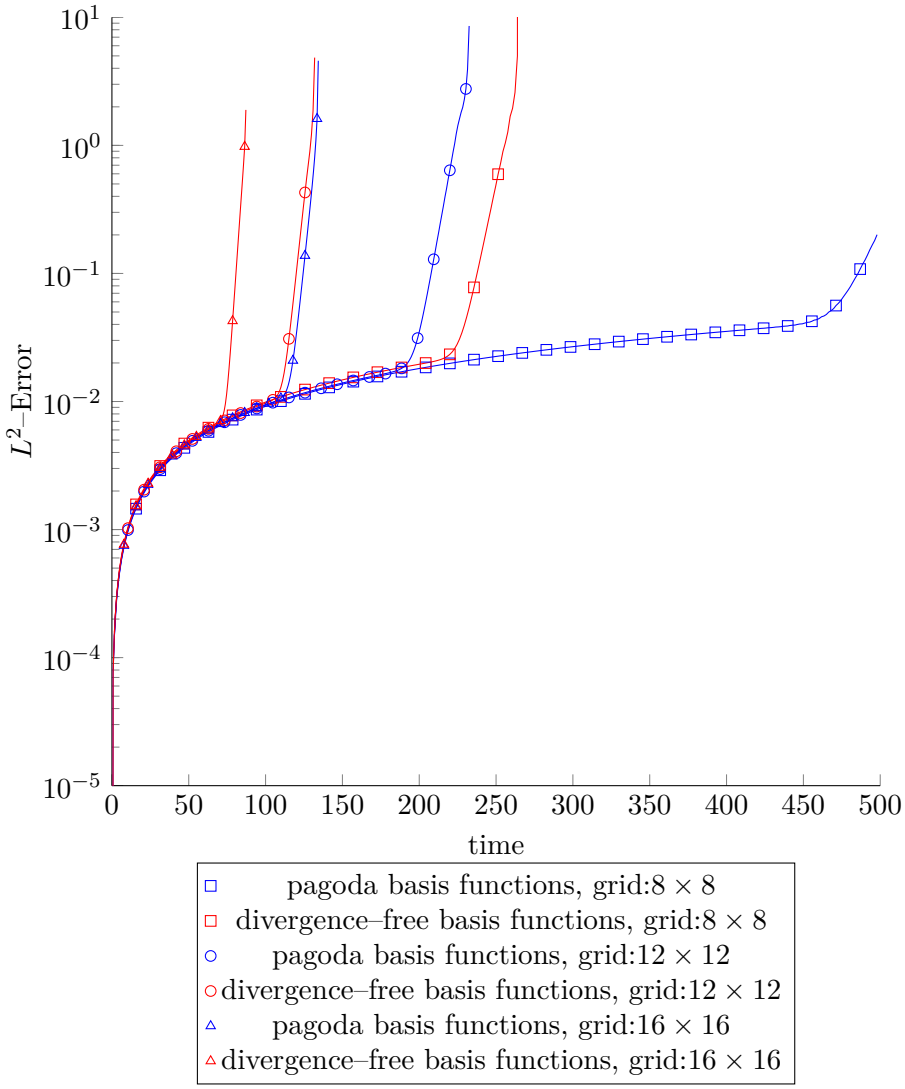

Figure 5.5.9.: error of simulations of Taylor–Green vortex flow for different grid resolutions at Re = 10000

6. Discussion of results

6.1. Comparison of standard FEM and divergence–free FEM

Two different types of basis functions for solving the incompressible Navier–Stokes equations developed in chapter chapter 2 have been presented in chapter chapter 3. The modifications to the MATLAB code *Quickfluid* have been shown in chapter 4 in order to perform various numerical Tests in chapter 5 using these two types of basis functions. In the first part of the current chapter divergence–free FEM and standard FEM are going to be compared on the basis of the results of the tests in chapter 5.

It has been shown that a stable computation of the presented test scenarios is possible for both kinds of basis functions for moderate Reynolds numbers smaller than 10000. For higher Reynolds numbers it was not possible to produce correct results for any of the both types of basis functions[1], and thus the only possible statement concerning stability is that at least divergence–free basis functions do not produce worse result than pagoda basis functions.

The test scenario Kovasznay flow is considered to test the convergence of a numerical simulation code (see section 5.3). For this test scenario both types of basis functions produce exactly the same errors over time and therefore the convergence of the simulation is identical for both types of basis functions.

6.2. Conclusion

During the work on this thesis the necessary boundary conditions and especially the L^2–projection, which is necessary for a generation of valid initial conditions, have been inserted into the existing FEM–code *Quickfluid*. In order to emphasize the easy handling of divergence–free basis functions it should again be mentioned that the main part of the work has **not** been the implementation of divergence–free basis functions, but the implementation of boundary conditions and initial conditions necessary for the considered test scenarios.

initial conditions have been implemented in a proper way by using an L^2–projection,

[1] See section 6.2 and subsection 5.5.2 for possible reasons for this behaviour.

which is according to [3] the only way of implementing initial conditions when using a GFEM[2] NSE discretisation. Additionally to the application of the L^2–projection for the generation of initial conditions, the L^2–projection could also be a useful tool for the generation of boundary conditions in the context of fluid–structure–interaction where new boundary conditions have to be calculated in every timestep due to the interaction between fluid and structure. Here the existing L^2–projection has to be extended to non–homogeneous Dirichlet boundary condition. Finally the L^2–projection has also been used for postprocessing purposes when calculating L^2–projections of an analytical reference solution for comparison with the actual numerical solution.

Even though, the question whether the used FEM–approach does or does not satisfy energy conservation and thus lead to correct results in test scenarios like jet impingment could not be answered completely. Even if it has been shown that the current version of *Quickfluid* does not lead to correct results for high Reynolds numbers it is still not entirely clear where these errors originate and how they can be prevented. It is obvious that a large scale redesign of elementary parts of the code is necessary for the implementation of concepts like upwinding[3], substitute lumped mass matrices $\tilde{A}$ with full mass matrices A[4] or trying different time integration schemes[5]. All of these points are possible topics for further investigation and the implemented test scenarios can be used for judging the effect of changes in the program.

One will notice that the approach used in this thesis compared to the approach presented in [2] show a less stable behaviour, especially for high Reynolds numbers[6]. But one should also notice that for several reasons (see subsection 5.5.2) the results in [2] could not be compared directly to the results presented in this thesis.

But even if the use of divergence–free basis functions does not show a clear advantage compared to using pagoda basis functions, the two test scenarios jet impingment and Taylor–Green vortex flow have been identified as test scenarios displaying spurious non–divergence–free behaviour and these two test scenarios — including the necessary additions like slip boundary conditions and the projection of initial conditions— have been implemented properly and thus further investigation of these two scenarios is possible — especially using the implemented post–processing tools.

[2] Galerkin finite element method

[3] This will probably lead to a stable computation of high Reynolds numbers.

[4] This is a theory due to [3] which may help reducing errors in the simulation. Still using A instead of $\tilde{A}$ leads to high computational cost in the calculation of Q and $\boldsymbol{b}$, because an additionally system of linear equations has to be solved in every step. Here a more efficient implementation of the assembly of Q and $\boldsymbol{b}$, as well as the calculation of $\boldsymbol{p}$ by solving the PPE would be necessary.

[5] This aspect has not been considered in this thesis.

[6] Evans did not meet the problems of velocity–blow up and non–physical behaviour when computing the same test scenarios with his FEM–code

A. Appendix

A.1. Element matrices

A.1.1. Matrices for standard FEM

Mass matrix A:

$$\frac{1}{36}\begin{pmatrix} 4 & 2 & 2 & 1 & 0 & 0 & 0 & 0 \\ 2 & 4 & 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 1 & 4 & 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 2 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 & 4 & 1 & 2 \\ 0 & 0 & 0 & 0 & 2 & 1 & 4 & 2 \\ 0 & 0 & 0 & 0 & 1 & 2 & 2 & 4 \end{pmatrix}$$

Discretization of ∇ via M:

$$\frac{1}{2}\begin{pmatrix} -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 \end{pmatrix}$$

Diffusion matrix D

$$\frac{\eta}{6\rho}\begin{pmatrix} 4 & -1 & -1 & -2 & 0 & 0 & 0 & 0 \\ -1 & 4 & -2 & -1 & 0 & 0 & 0 & 0 \\ -1 & -2 & 4 & -1 & 0 & 0 & 0 & 0 \\ -2 & -1 & -1 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & -1 & -1 & -2 \\ 0 & 0 & 0 & 0 & -1 & 4 & -2 & -1 \\ 0 & 0 & 0 & 0 & -1 & -2 & 4 & -1 \\ 0 & 0 & 0 & 0 & -2 & -1 & -1 & 4 \end{pmatrix}$$

Convection matrix $C(\boldsymbol{u}_h)$:

$$\frac{1}{72}\begin{pmatrix}
-6 & 6 & -2 & 2 & 0 & 0 & 0 & 0 \\
-3 & 3 & -1 & 1 & 0 & 0 & 0 & 0 \\
-2 & 2 & -2 & 2 & 0 & 0 & 0 & 0 \\
-1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\
6 & 2 & -6 & -2 & 0 & 0 & 0 & 0 \\
2 & 2 & -2 & -2 & 0 & 0 & 0 & 0 \\
3 & 1 & -3 & -1 & 0 & 0 & 0 & 0 \\
1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
-3 & 3 & -1 & 1 & 0 & 0 & 0 & 0 \\
-6 & 6 & -2 & 2 & 0 & 0 & 0 & 0 \\
-1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\
-2 & 2 & -2 & 2 & 0 & 0 & 0 & 0 \\
2 & 2 & -2 & -2 & 0 & 0 & 0 & 0 \\
2 & 6 & -2 & -6 & 0 & 0 & 0 & 0 \\
1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
1 & 3 & -1 & -3 & 0 & 0 & 0 & 0 \\
-2 & 2 & -2 & 2 & 0 & 0 & 0 & 0 \\
-1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\
-2 & 2 & -6 & 6 & 0 & 0 & 0 & 0 \\
-1 & 1 & -3 & 3 & 0 & 0 & 0 & 0 \\
3 & 1 & -3 & -1 & 0 & 0 & 0 & 0 \\
1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
6 & 2 & -6 & -2 & 0 & 0 & 0 & 0 \\
2 & 2 & -2 & -2 & 0 & 0 & 0 & 0 \\
-1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 \\
-2 & 2 & -2 & 2 & 0 & 0 & 0 & 0 \\
-1 & 1 & -3 & 3 & 0 & 0 & 0 & 0 \\
-2 & 2 & -6 & 6 & 0 & 0 & 0 & 0 \\
1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
1 & 3 & -1 & -3 & 0 & 0 & 0 & 0 \\
2 & 2 & -2 & -2 & 0 & 0 & 0 & 0 \\
2 & 6 & -2 & -6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -6 & 6 & -2 & 2 \\
0 & 0 & 0 & 0 & -3 & 3 & -1 & 1 \\
0 & 0 & 0 & 0 & -2 & 2 & -2 & 2 \\
0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 0 & 6 & 2 & -6 & -2 \\
0 & 0 & 0 & 0 & 2 & 2 & -2 & -2 \\
0 & 0 & 0 & 0 & 3 & 1 & -3 & -1 \\
0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\
0 & 0 & 0 & 0 & -3 & 3 & -1 & 1 \\
0 & 0 & 0 & 0 & -6 & 6 & -2 & 2 \\
0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 0 & -2 & 2 & -2 & 2 \\
0 & 0 & 0 & 0 & 2 & 2 & -2 & -2 \\
0 & 0 & 0 & 0 & 2 & 6 & -2 & -6 \\
0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\
0 & 0 & 0 & 0 & 1 & 3 & -1 & -3 \\
0 & 0 & 0 & 0 & -2 & 2 & -2 & 2 \\
0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 0 & -2 & 2 & -6 & 6 \\
0 & 0 & 0 & 0 & -1 & 1 & -3 & 3 \\
0 & 0 & 0 & 0 & 3 & 1 & -3 & -1 \\
0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\
0 & 0 & 0 & 0 & 6 & 2 & -6 & -2 \\
0 & 0 & 0 & 0 & 2 & 2 & -2 & -2 \\
0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 0 & -2 & 2 & -2 & 2 \\
0 & 0 & 0 & 0 & -1 & 1 & -3 & 3 \\
0 & 0 & 0 & 0 & -2 & 2 & -6 & 6 \\
0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\
0 & 0 & 0 & 0 & 1 & 3 & -1 & -3 \\
0 & 0 & 0 & 0 & 2 & 2 & -2 & -2 \\
0 & 0 & 0 & 0 & 2 & 6 & -2 & -6
\end{pmatrix}$$

A.1.2. Matrices for divergence–free FEM

Mass matrix A:

$$\frac{1}{24}\begin{pmatrix} 3 & 1 & 1 & 1 & -1 & 0 & 0 & -1 \\ 1 & 3 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 3 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 3 & -1 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 & 3 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 3 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 3 & 1 \\ -1 & 0 & 0 & -1 & 1 & 1 & 1 & 3 \end{pmatrix}$$

Discretization of ∇ via M:

$$\frac{1}{2}\begin{pmatrix} -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 \end{pmatrix}$$

Diffusion matrix D

$$\frac{\eta}{2\rho}\begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

Convection matrix $C(\boldsymbol{u}_h)$:

$$\frac{1}{96}\begin{pmatrix}
-9 & 7 & -2 & 4 & -1 & 2 & 3 & -4 \\
-3 & 5 & -2 & 0 & 1 & -2 & 1 & 0 \\
-1 & 3 & -4 & 2 & -1 & 0 & 3 & -2 \\
-3 & 1 & 0 & 2 & 1 & 0 & 1 & -2 \\
9 & 2 & -7 & -4 & 1 & -3 & -2 & 4 \\
1 & 4 & -3 & -2 & 1 & -3 & 0 & 2 \\
3 & 2 & -5 & 0 & -1 & -1 & 2 & 0 \\
3 & 0 & -1 & -2 & -1 & -1 & 0 & 2 \\
-5 & 3 & 0 & 2 & -2 & 1 & 0 & 1 \\
-7 & 9 & -4 & 2 & 2 & -1 & -4 & 3 \\
-1 & 3 & -2 & 0 & 0 & 1 & -2 & 1 \\
-3 & 1 & -2 & 4 & 0 & -1 & -2 & 3 \\
4 & 1 & -2 & -3 & 3 & -1 & -2 & 0 \\
2 & 9 & -4 & -7 & 3 & -1 & -4 & 2 \\
0 & 3 & -2 & -1 & 1 & 1 & -2 & 0 \\
2 & 3 & 0 & -5 & 1 & 1 & 0 & -2 \\
-4 & 2 & -1 & 3 & -3 & 2 & 1 & 0 \\
0 & 2 & -3 & 1 & -1 & 2 & -1 & 0 \\
-2 & 4 & -9 & 7 & -3 & 4 & 1 & -2 \\
-2 & 0 & -3 & 5 & -1 & 0 & -1 & 2 \\
5 & 0 & -3 & -2 & 2 & 0 & -1 & -1 \\
1 & 2 & -3 & 0 & 0 & 2 & -1 & -1 \\
7 & 4 & -9 & -2 & -2 & 4 & 1 & -3 \\
3 & 2 & -1 & -4 & 0 & 2 & 1 & -3 \\
-2 & 0 & -1 & 3 & 2 & -1 & 0 & -1 \\
-2 & 4 & -3 & 1 & 2 & -3 & 0 & 1 \\
0 & 2 & -5 & 3 & 0 & -1 & 2 & -1 \\
-4 & 2 & -7 & 9 & 4 & -3 & -2 & 1 \\
2 & 1 & 0 & -3 & -2 & 0 & 1 & 1 \\
0 & 5 & -2 & -3 & 0 & -2 & 1 & 1 \\
2 & 3 & -4 & -1 & -2 & 0 & 3 & -1 \\
4 & 7 & -2 & -9 & -4 & 2 & 3 & -1 \\
-1 & 2 & 3 & -4 & -9 & 7 & -2 & 4 \\
1 & -2 & 1 & 0 & -3 & 5 & -2 & 0 \\
-1 & 0 & 3 & -2 & -1 & 3 & -4 & 2 \\
1 & 0 & 1 & -2 & -3 & 1 & 0 & 2 \\
1 & -3 & -2 & 4 & 9 & 2 & -7 & -4 \\
1 & -3 & 0 & 2 & 1 & 4 & -3 & -2 \\
-1 & -1 & 2 & 0 & 3 & 2 & -5 & 0 \\
-1 & -1 & 0 & 2 & 3 & 0 & -1 & -2 \\
-2 & 1 & 0 & 1 & -5 & 3 & 0 & 2 \\
2 & -1 & -4 & 3 & -7 & 9 & -4 & 2 \\
0 & 1 & -2 & 1 & -1 & 3 & -2 & 0 \\
0 & -1 & -2 & 3 & -3 & 1 & -2 & 4 \\
3 & -1 & -2 & 0 & 4 & 1 & -2 & -3 \\
3 & -1 & -4 & 2 & 2 & 9 & -4 & -7 \\
1 & 1 & -2 & 0 & 0 & 3 & -2 & -1 \\
1 & 1 & 0 & -2 & 2 & 3 & 0 & -5 \\
-3 & 2 & 1 & 0 & -4 & 2 & -1 & 3 \\
-1 & 2 & -1 & 0 & 0 & 2 & -3 & 1 \\
-3 & 4 & 1 & -2 & -2 & 4 & -9 & 7 \\
-1 & 0 & -1 & 2 & -2 & 0 & -3 & 5 \\
2 & 0 & -1 & -1 & 5 & 0 & -3 & -2 \\
0 & 2 & -1 & -1 & 1 & 2 & -3 & 0 \\
-2 & 4 & 1 & -3 & 7 & 4 & -9 & -2 \\
0 & 2 & 1 & -3 & 3 & 2 & -1 & -4 \\
2 & -1 & 0 & -1 & -2 & 0 & -1 & 3 \\
2 & -3 & 0 & 1 & -2 & 4 & -3 & 1 \\
0 & -1 & 2 & -1 & 0 & 2 & -5 & 3 \\
4 & -3 & -2 & 1 & -4 & 2 & -7 & 9 \\
-2 & 0 & 1 & 1 & 2 & 1 & 0 & -3 \\
0 & -2 & 1 & 1 & 0 & 5 & -2 & -3 \\
-2 & 0 & 3 & -1 & 2 & 3 & -4 & -1 \\
-4 & 2 & 3 & -1 & 4 & 7 & -2 & -9
\end{pmatrix}$$

A.2. M–files defining the basis functions

A.2.1. pagoda basis functions

```
function [ u,v ] = normal_base( x_,y_,x0,y0,h)
%NORMAL_BASE retuns the values for the pagoda basis function with
%at the position x0,y0 at x_,y_. The compact support of the basis function
%is defined via the gridsize h. Vector and matrix input of x_,y_ is also
%accepted.

u=zeros(size(x_));
v=u;

%coordinates (x_,y_) are shifted to coordinate system where (x0,y0)=(0,0)
%aditionally symmetry of the basis function is applied by considering
%absolute values
x=abs(x0*ones(size(x_))-x_);
y=abs(y0*ones(size(x_))-y_);

%looping over each point where the basis function is evaluated
for i = 1:numel(x)

    if(and(x(i)<=h,y(i)<=h))%(x,y) inside support, else (u,v)=(0,0)
        %bilinear definition of pagoda basis function
        u(i)=(h-x(i))*(h-y(i));
    end

end

end
```

A.2.2. divergence–free basis functions

```
function [ u,v ] = div_free_base( x_,y_,x0,y0,h)
%DIV_FREE_BASE retuns the values for the divergence free basis function
%with at the position x0,y0 at x_,y_. The compact support of the basis
%function is defined via the gridsize h. Vector and matrix input of x_,y_
%is also accepted.

u=zeros(size(x_));
v=u;

%coordinates (x_,y_) are shifted to coordinate system where (x0,y0)=(0,0)
%aditionally symmetry of the basis function is applied by considering
%absolute values
x=abs(x0*ones(size(x_))-x_);
y=abs(y0*ones(size(x_))-y_);

%looping over each point where the basis function is evaluated
for i = 1:numel(x)

    if(and(x(i)<=h,y(i)<=h))%(x,y) inside support, else (u,v)=(0,0)
        %piecewise definition of divergence free basis function
        if(and(x(i)>=y(i),x(i)<=h-y(i)))
            u(i)=1-x(i)/h-.5*y(i)/h;
            v(i)=.5*y(i)/h;
        elseif(and(x(i)>=y(i),x(i)>=h-y(i)))
            u(i)=.5-.5*x(i)/h;
            v(i)=u(i);
        elseif(and(x(i)<=y(i),x(i)>=h-y(i)))
            u(i)=.5-.5*y(i)/h;
            v(i)=u(i);
        elseif(and(x(i)<=y(i),x(i)<=h-y(i)))
            u(i)=1-.5*x(i)/h-y(i)/h;
            v(i)=.5*x(i)/h;
        else
            'ERROR'
        end

        %apply symmetry of y-component of basis function correctly
        if((x_(i)-x0)*(y_(i)-y0)<0)
            v(i)=-v(i);
        end
    end
end

end
```

Bibliography

[1] Cornelia Blanke. *Kontinuitätserhaltende Finite-Elemente-Diskretisierung der Navier-Stokes-Gleichungen.* Diplomarbeit, Technische Universität München, 2004.

[2] JA Evans. *Divergence-free B-spline discretizations for viscous incompressible flows.* PhD thesis, 2012.

[3] P M Gresho and R L Sani. *Incompressible Flow and the Finite Element Method, Isothermal Laminar Flow.* Incompressible Flow and the Finite Element Method. John Wiley & Sons, 2000.

[4] M Griebel, T Dornseifer, and T Neunhoeffer. *Numerische Simulation in der Strömungsmechanik: eine praxisorientierte Einführung.* Vieweg Lehrbuch. Vieweg, 1995.

[5] T Neckel. *The PDE Framework Peano: An Environment for Efficient Flow Simulations.* Verlag Dr. Hut, 2009.

[6] Tobias Neckel, Miriam Mehl, and Christoph Zenger. Enhanced divergence-free elements for efficient incompressible flow simulations in the PDE framework Peano. *Fifth European Conference on Computational Fluid Dynamics*, (June):14–17, 2010.

[7] Nicola Warneke. *Divergenzfreie Finite-Elemente für die 3D Navier-Stokes Gleichungen auf kartesischen Gittern.* Masterarbeit, Technische Universität München, 2013.