DISS. ETH NO. 25166

A parallel space-time solver for the Navier–Stokes equations with periodic forcing

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

DANIEL HUPP

MSc ETH CSE, ETH Zurich

born on 09.09.1987

citizen of Germany

accepted on the recommendation of

Prof. Dr. Peter Arbenz (ETH Zurich), examiner

Prof. Dr. Dominik Obrist (University of Bern), co-examiner

Prof. Dr. Patrick Jenny (ETH Zurich), co-examiner

Prof. Dr. Rolf Krause (Università della Svizzera italiana), co-examiner

2018

Abstract

Many problems in science and engineering are driven by time-periodic forces. In fluid dynamics, this occurs for example in turbines, rotors or in human blood flow. These problems are described by the Navier–Stokes equations. The traditional way to solve them is to use a time-stepping method and compute the whole transitional phase until the time-periodic steady state is reached.

We are using a different approach. We model directly the steady state by using periodic boundary conditions in time. This results into time-periodic Navier–Stokes equations. We use two different ways to discretize these equations in time. One way is to use a truncated Fourier series to represent the solution. The equations are then discretized by a spectral Galerkin method. The other way is to use the finite difference method in time. Both temporal discretization are accompanied by spatial finite differences in space.

We solve the resulting nonlinear space-time problem with a Picard iteration method. A linear system of equation has to be solved in each Picard iteration. This linear system is solved by preconditioned Krylov methods. We provide problem specific preconditioners. For the spectral in time method, we use a block-tridiagonal preconditioner. The $(1, 1)$ block is a diagonal approximation of the multi-harmonic convection-diffusion matrix. The $(2, 2)$ block is an approximation of the Schur complement based on a commutator. For the finite differences in time, we use a geometric multigrid method with a space-time box smoother.

The described discretizations and solvers have been implemented in a templated C++ library. This library is very flexible and uses the message passing interface (MPI) to parallelize the space-time problems. The parallelization is done in space and time.

We applied the spectral in time method to various problems. A Rayleigh streaming has been computed to test the preconditioner of the linear solvers. A Taylor–Green vortex has been computed to compare the spectral in time method with a traditional time-stepping method. A channel flow with an oscillating obstacle has been computed to evaluate the parallel performance in time. A time-periodically disturbed swept Hiemenz flow has been computed to test the solver for turbulent flows. Such turbulent flows are not purely time-periodic. Only its statistics are time-periodic.

Zusammenfassung

Viele Probleme in Wissenschaft und Technik sind von zeitperiodischen Kräften angetrieben. In der Strömungslehre kommt das in Turbinen, Rotoren, oder im menschlichen Blutfluss vor. Solche Probleme werden durch die Navier–Stokes-Gleichungen beschrieben. Traditionell werden diese Probleme mit einem Zeitschriftverfahren gelösst, welches die transitive Phase solange durchschreitet bis ein zeitperiodischer stationärer Zustand erreicht wird.

Wir haben eine andere Herangehensweise. Wir modellieren direkt den zeitperiodischen stationären Zustand, indem wir periodische Randbedingungen in der Zeit einführen. Dies führt zu zeitperiodischen Navier–Stokes-Gleichungen. Wir diskretisieren diese Gleichungen in der Zeit auf zwei verschiedene Arten. Die erste Art verwendet eine abgeschnittene Fourier Reihe um die Lösung zu beschreiben. Die Gleichungen werden dann mit einer spektralen Galerkin-Methode diskretisiert. Die zweite Arte verwendet die Finite-Differenzen-Methode in der Zeit. Beide zeitlichen Diskretisierungen werden mit finiten Differenzen im Raum verbunden.

Wir lösen das resultierende Raumzeit Problem mit einer iterativen Picard Methode. In jedem Picard Schritt muss ein lineares Gleichungssystem gelöst werden. Dieses lineare Gleichungssystem wird mit einer vorkonditionierten Krylov Methode gelöst. Wir benutzen Problem spezifische Vorkonditioniere. Für die spektrale Methode in der Zeit verwenden wir einen Block Dreiecksvorkonditionierer. Der $(1,1)$ Block ist eine diagonale Approximation einer multi harmonischen Konvektions-Diffusions-Gleichung. Der $(2,2)$ Block ist eine Approximation des Schurkomplements, basierend auf einem Kommutator. Für die finiten Differenzen in der Zeit verwenden wir ein geometrisches Mehrgitterverfahren mit einem Raumzeit Box Glätter.

Die beschriebenen Diskretisierungen und Löser wurden in einer templierten C++ Bibliothek implementiert. Diese Bibliothek is sehr flexibel und verwendet die Message Passing Interface (MPI) um die Raumzeit Probleme zu parallelisieren. Die Parallelisierung wird im Raum und der Zeit angewendet.

Wir wenden die spektrale Method in der Zeit auf verschiedene Probleme an. Eine Rayleigh Strömung wurde berechnet um die Vorkonditionierer für die linearen Probleme zu testen. Ein Taylor–Green Wirbel wurde berechnet um die spektrale Methode in der Zeit mit einem traditionellen Zeitschrittverfahren zu verwenden. Eine Kanalströmung

mit oszillierendem Hindernis wurde berechnet um das parallele Verhalten in der Zeit zu testen. Eine zeitperiodisch angeregte schiebende Hiemenz-Strömung wurde berechnet um den Löser für turbulente Strömungen zu testen. Solche turbulenten Strömungen sind nicht rein zeitperiodisch. Nur ihre Statistiken sind zeitperidisch.

Acknowledgments

First of all I want to thank Prof. Peter Arbenz for giving me the opportunity to work on this project. I am very thankful for his support and of course his patience. He always took the time to discuss problems and new ideas, which has been a huge help to me. Also special thanks go to Prof. Dominik Obrist, whose insights and ideas have been very valuable to this project. His dedication and interest have been a big motivation to me.

I would also like to thank my co-examiners, Prof. Patrick Jenny and Prof. Rolf Krause, for taking the time to review this thesis.

I had the pleasure to supervise the master's thesis of Pietro Benedusi. His work and his rewarding collaboration made valuable contributions to this project. I want to thank him for that.

I also want to thank ETH Zürich and in particular the computer science department. They are providing a productive and motivating environment with many nice colleagues. I specially thank my colleagues Stefan Pauli, Tulin Kaman, and Marija Kranjčević for their interest and fruitful discussions.

I thank the Swiss National Science Foundation for funding this project by the Grant No. 200021 147052. I am also very grateful that I had the opportunity to present my work at many international conferences and workshops.

Last but not least I want to thank my family and friends for their nonacademic support and their understanding in times of heavy load.

Contents

Notation **xiii**

1 Introduction **1**

1.1 Motivation 1

1.2 Current research 2

1.2.1 Parallel-in-time methods 2

1.2.2 Methods for time-periodic problems 5

1.2.3 Related work 8

1.3 Outline of the thesis 9

2 Discretization of the time-periodic Navier–Stokes equations **11**

2.1 Governing equations 11

2.2 Time discretization 13

2.2.1 Spectral in time discretization 13

2.2.2 Finite differences in time 17

2.3 Spatial discretization 17

3 Numerical solvers for the time-periodic Navier–Stokes problems **21**

3.1 A nonlinear solver for the spectral in time discretization 21

3.1.1 Picard iteration 22

3.1.2 Solution based spectral refinement 29

3.1.3 Residual based spectral refinement 30

3.1.4 Preconditioning 31

3.1.4.1 Picard problem 32

3.1.4.2 Multi-harmonic convection-diffusion problem 36

3.1.4.3 Harmonic convection-diffusion problem 37

3.1.4.4 Stationary convection-diffusion problems 39
3.1.4.5 Poisson problem 41
3.2 A nonlinear solver for the finite differences in time discretization 42
3.2.1 Picard iteration 43
3.2.2 Preconditioning 44
3.2.2.1 Four-dimensional restriction and interpolation 45
3.2.2.2 Box-smoothing 45

4 Implementation of the parallel time-periodic solvers 47
4.1 Implementation process 47
4.2 Interfaces 49
4.3 Parallelization 56

5 Experiments 61
5.1 Rayleigh streaming 61
5.1.1 Problem 62
5.1.2 Performance 64
5.2 Time-periodic Taylor–Green vortex 69
5.2.1 Problem 69
5.2.2 Convergence of the time discretization 71
5.2.3 Convergence of the method 72
5.2.4 Scaling 73
5.2.5 Time to solution 74
5.2.6 Conclusions 77
5.3 Channel flow with oscillating obstacle 79
5.3.1 Problem 79
5.3.2 Performance 81
5.3.3 Scaling 84

5.3.4 Conclusions 86
5.4 Swept Hiemenz flow 88
5.4.1 Problem 88
5.4.2 Results 91
5.4.3 Conclusions 103

6 Conclusions and future work 105

6.1 Conclusions 105
6.2 Future work 107

A List of integrals of trigonometric functions 109

B Backtracking 111

C Computing null space of the pressure problem 113

D Getting Pimpact 117

Notation

Constants

α	Womersley number
Re	Reynolds number
St	Strouhal number

Picard iteration

$\cdot^{(m)}$	mth Picard iteration
$\delta\cdot$	Picard correction
$\mathbf{q}$	Space-time solution vector
$\mathbf{r}$	Space-time residual vector

Multi-harmonic variables

$\hat{p}^0$	Zero pressure coefficient
$\hat{p}_l^c$	lth cosine pressure coefficient
$\hat{p}_l^s$	lth sine pressure coefficient
$\hat{\mathbf{u}}^0$	Zero velocity coefficient
$\hat{\mathbf{u}}_l^c$	lth cosine velocity coefficient
$\hat{\mathbf{u}}_l^s$	lth sine velocity coefficient

Space-time operators

$\mathcal{D}$	Multi-harmonic divergence operator
$\mathcal{F}$	Multi-harmonic or time-periodic convection-diffusion operator
$\mathcal{G}$	Multi-harmonic gradient operator
$\mathcal{H}$	Picard matrix
$\mathcal{I}$	Multi-harmonic identity operator
$\mathcal{J}$	Multi-harmonic boundary condition operator
$\mathcal{S}$	Multi-harmonic Schur complement
K	Harmonic convection-diffusion operator

Preconditioner

$\mathcal{M}_{\mathcal{F}}$	Preconditioner of the multi-harmonic convection-diffusion operator
$\mathcal{M}_{\mathcal{H}}$	Preconditioner for the Picard matrix
$\mathcal{M}_{\mathcal{S}}$	Approximation of the multi-harmonic Schur complement
M_{K}	Preconditioner of the harmonic convection-diffusion operator

Spatial operators

$\boldsymbol{\nabla}$	Nabla operator
$\boldsymbol{\Delta}$	Laplace operator
D	Spatially divergence operator $\boldsymbol{\nabla}\cdot$
G	Spatially gradient operator $\mathrm{Re}\boldsymbol{\nabla}$
I	Identity matrix
$\bar{\mathrm{I}}$	Identity matrix for inner spatial grid points, zero for boundary conditions grid points
J	Identity matrix, with boundary conditions scaling/interpolation

1

Introduction

1.1 Motivation

The field of this thesis is computational fluid dynamics (CFD). CFD is part of computational science and engineering and deals with the simulation of fluids, such as water or air. Incompressible viscous fluids are described by the Navier–Stokes equation.

Modern solvers for the Navier–Stokes equations are optimized for massively-parallel computing platforms and can easily use many thousands of processing units by decomposing the spatial flow domain into small subdomains. This allows nowadays the solution of complex unsteady flow problems at high Reynolds numbers with billions of grid points.

The continued increase in computational power allows us to simulate ever larger flow problems as long as we are able to decompose the flow domain into sufficiently many subdomains. The excellent weak scaling performance of modern solvers suggests that we should be able to scale the simulations without any limit in sight. However, this optimistic extrapolation does not factor in the increasing number of time steps which typically comes along with larger flow domains. This problem is aggravated for a large class of flow problems which are concerned with the spatial evolution of time-harmonic perturbations, e.g., a vibrating ribbon perturbing a boundary layer flow. The numerical simulation of such problems with classical time marching methods requires long transient periods at the beginning of each simulation. These transient periods are of no particular physical interest for many applications. The relevant flow field is only established when a periodic (or at least statistically steady) solution has been reached. In terms of dynamical systems, this can be understood as the asymptotic approach to a limit cycle of a given system.

For such problems with periodic forcing, the spatial decomposition of the computational domain might not yield sufficient parallelism for the efficient usage of modern massively-parallel supercomputers. Only a parallelization of the time integration could ease this potential scaling limit for flow problems of this type. The development and implementation of such parallel-in-time integration of harmonic perturbed problems is the topic of this thesis. But first we give a short overview of current and related research that also deals with these problems.

1.2 Current research

Domain decomposition is the state-of-the-art technique to parallelize the solution of partial differential equations with many degrees of freedom on massively parallel computer systems. Domain decomposition divides the computational domain into subdomains which are then distributed to different computing units. The global problem is finally solved using iterative methods which are either based on distributed linear algebra [78] or based on mathematically modeling a subproblem for each subdomain [91].

For time-dependent problems a time integrator is used to compute the evolution of the spatially discretized problem. If the time integrator is explicit, a discretized spatial operator has to be applied many times sequentially in time. If the time integrator is implicit, a discretized spatial problem has to be solved many times sequentially in time. It follows that if we have to compute long time spans, the restriction of parallelism to space becomes the limiting factor if processor numbers increase. This can be improved only if the time integration is parallelized as well. The development of parallel-in-time methods is an active field of research.

1.2.1 Parallel-in-time methods

For an extensive overview of parallel-in-time methods over the last 50 years we refer to [33]. A short overview of the most promising methods and their applications is given here.

The “parareal” approach [64] is one of the first broadly used methods to parallelize the time integration. Similar to the domain decomposition idea the time dimension is divided. The time interval $[0, T]$ is split into multiple subintervals $[t_i,\ t_{i+1}]$. A ‘coarse integrator’ and a ‘fine integrator’ is used on these subintervals. The objective is to obtain a solution with the accuracy of the fine integrator in shorter computation time then the fine integrator. In a first initializing step the time-dependent problem is solved by the coarse integrator on the subinterval bounds t_i. This solution is used as initial guess and initial value for each subinterval. Then the main “parareal” iteration ist started. The iteration contains two steps.

1. A solution is computed on the right of each subinterval, by the fine and coarse integrator. The initial value on each subinterval is the value from the previous “parareal” iteration. This step is computed in parallel for each subinterval.
2. The solutions on the interfaces are updated. The new value on the right bound of a subinterval is computed by a coarse integrated value of the new value from the left bound and the difference of the two results of the parallel-in-time step. As the new value on the right bound depends on the new value from the left, this step has to be done sequentially.

The global solution is obtained through iterating these two steps. To obtain a good speedup with regard to the sequential fine integrator over the whole time interval $[0,\ T]$, it is important that the coarse integrator is computationally much cheaper than the fine integrator, and that the number of “parareal” iterations is smaller than the number of subintervals. This “parareal” method has also recently been applied to the Navier–Stokes equations [19, 23] and to plasma turbulence [80].

The parallel full approximation scheme in space and time (PFASST) [27] combines three concepts, the spectral deferred correction scheme, parallel-in-time integration, and the multigrid full approximation scheme [92]. For the spectral deferred correction scheme, an ordinary differential equation is formulated for the error given an approximate solution.

1. An approximate solution is computed by a low order time integrator.
2. The error is estimated by a higher order Gauss quadrature rule.
3. The solution is improved by subtracting the estimated error.

Through iterating these steps, a high order solution is obtained. To parallelize the spectral deferred correction scheme, multiple subintervals are treated simultaneously. To do that instead of waiting until the spectral deferred correction iteration is finished, the next subinterval is already started with the current approximate solution. This hybrid “parareal” spectral deferred correction method [66] can be used as a smoother plugged into a multigrid full approximation scheme. This allows to use more levels of spectral deferred correction scheme and makes it easier to combine with spatial multigrid. This method has been successfully applied to N-body problems [87]. Recently, in [11], the convergence has been investigated by analyzing PFASST as a multigrid method.

The multigrid reduction in time (MGRIT) [32] can be seen as a more direct extension of the "parareal" method. Here the idea of having two different time levels for the "parareal" method is extended to multiple levels. These levels can be traversed in an F, V, or W-cycle way, using a full approximation scheme [92].

1. The restriction is done by injection.
2. The interpolation is done in an optimal reduction way.
3. The relaxation is done by a fine relaxation sweep followed by a coarse relaxation sweep, followed by another fine relaxation sweep. The coarse relaxation means to solve multiple coarse time-step sequentially. The fine relaxation means to solve for every coarse time-step a finer discretized problem in parallel.

These are the components of the multigrid reduction in time method, which shows good scalability. The advantage of this method is that existing time-stepping methods and implementation can be easily plugged in. Only a propagator with adjustable time-step size is needed. It has been applied to nonlinear parabolic problems [29] and to compressible fluid problems [28].

In [35, 68] a multigrid method is proposed but not only in time but also in space. The authors propose to discretize partial differential equations with finite elements in space and discontinuous Galerkin in time. This results in a lower block bidiagonal matrix. Each block corresponds to a spatial problem. A multigrid method is applied to this matrix.

1. Standard multigrid restriction and interpolation is used.
2. A block Jacobi iteration is used as smoother, which is the key idea.

The convergence is analyzed analytically and experimentally. Also good parallel performance could be shown experimentally. Time periodicity has been assumed for the theoretical convergence analysis. Therefore it can be expected that this method might work well for time-periodic problems.

All these methods are data parallel methods as time subintervals or space-time subdomains are processed in parallel. But there are also task parallel methods, for example the revisionist integral defect correction method (RIDC) [17]. RIDC can be seen as a prediction-correction method, that is like PFASST based on the spectral deferred correction method. The same integral error formulation is used that allows to increase iteratively the order of a time integrator. This is done by integrating the error for a given approximate solution. Then the approximate solution is improved by subtracting the error. PFASST does multiple time-intervals in parallel, using the current best solution from the previous time-interval. Instead RIDC pipelines the prediction and correction steps.

1. One core computes sequentially predictions.
2. The next core computes the first corrections lagging one time step behind.
3. The next cores compute the next corrections and so on.

This works very well on multi-core machines, as communication is cheap. So if the pipeline is filled up, this method can produce high order results in the same computational time as a low order method. However one disadvantage of this method is that the achieved parallelism directly depend on the used order. So for most applications, only a small number of additional parallelism can be achieved. In [18] this RIDC method has been coupled with spatial parallelization. Additive Schwartz iteration is used as domain decomposition method in space.

1.2.2 Methods for time-periodic problems

There has also research been done that is focusing on time-periodic problems, by assuming the existence of a time-periodic steady state. The problem has already been considered before the advent of massively parallel computer systems in 1964 in [90]. There the time-periodic steady state is computed for the heat equation. The heat equation has been discretized by finite differences in space and time. The resulting linear system of equations has a p-cyclic structure. A matrix is p-cyclic if it can be permuted such that it has p blocks on the diagonal; $p-1$ blocks on the first lower off-diagonal and one entry in the top right block. The convergence rate of the block Jacobi iteration and block symmetric overrelaxation is shown by using properties of p-cyclic matrices. The successive overrelaxation is optimized for this specific matrix structure. It is shown that this successive overrelaxations is more efficient than the standard step by step solution process. The problem has been picked up in [74], where an efficient direct solver has been constructed.

Later Hackbusch studied time-periodic linear and nonlinear problems in [39]. He reformulated the time-periodic steady state problem into a fixed-point problem. He showed that the according fixed-point iteration has good smoothing properties. Thus he proposed to use the multigrid method as a solver for this kind of problem. Therefore he constructs a hierarchy of grids, which are traversed in a full multigrid fashion. Full multigrid means that one starts to solve the problem on the coarsest grid. The coarse grid solutions is interpolated to the next finest grid and is used as initial guess for a V or W-cycle. This is repeated till one has obtained a solution on the finest grid. By numerical experiments he observed that one iteration of the full multigrid yields a satisfactory approximation of the solution. Additionally, the same approach has been formulated and applied to nonlinear time-periodic problems.

Also in computational electrodynamics the efficient computation of time-periodic steady states has been of interest. In [43] the authors computed a time-periodic steady state of a nonlinear diffusion equation by using finite differences in space and first order backward finite differences in time with periodic boundary conditions. The nonlinear space-time system is solved by a modified Newton method. The iteration matrix has a p-cyclic structure so the top right entry is moved to the right hand side, with the values from the previous Newton iteration step, such that one can solve the system by block wise forward substitution. The authors showed that this method provides correct results and that the time to solution is shorter in comparison to a time-stepping method. In [96] time-periodic magnetic fields have been computed by using a Fourier ansatz in time and finite elements in space. The authors could verify the correctness of the method by simulating a reactor and a magnet of shading coils. In [21], this multi-harmonic finite element approach has been used for three-dimensional eddy current problems. The nonlinear system of equations has been solved with Picard iteration. The authors showed that for specific quasi-coupled problems the multi-harmonic approach is beneficial. This has been pursued in [4, 5] for time-periodic eddy current problems. An inexact Newton method is used to solve the nonlinear space-time problem. As the complex Fourier series is not differentiable the authors are using a real Fourier series. An optimal preconditioner is constructed for this specific type of problem.

Motivated by the simulation of turbomachinery with rotating blades the harmonic balance method has been developed [42]. Again the solution is expressed as a truncated Fourier series, assuming the existence of a time-periodic steady state. The truncated Fourier series is plugged into the governing equations. As the method is developed for turbomachinery, the governing equations are the Euler equations or the Reynolds averaged Navier–Stokes equations. The equations with the plugged-in Fourier ansatz are then simplified such that each coefficient of each simple function is expected to become zero. Hence the method's name includes balance. This gives rise to a system of nonlinear equations. But instead of solving directly the nonlinear system of equations, an additional artificial time derivative is introduced in the frequency space. This artificial time is called pseudo time. It is used to drive the individual Fourier coefficients into a steady state where the solution is balanced. The advantage of using a pseudo time step to solve the nonlinear system of equations is that existing and well developed accelerating techniques for steady state flow solvers can be applied. This is the basic harmonic balance method which has been further developed. For an overview we refer to [41]. The harmonic balance method is similar to the method presented later in this thesis as it uses a Fourier series to describe the solution. It differs in the derivation of the equations and the nonlinear system solver. The harmonic balance method is actively used to solve time-periodic fluid problems, as, for example, incompressible Navier–Stokes problems [20], or for multi-frequent turbomachinery problems [37].

Following the harmonic balance idea, the time-spectral method [36] has been developed using the same spectral description of the time-periodic steady-state solution. Here, instead of the pseudo time-stepping in the frequency domain, the time-spectral method does the pseudo time-stepping in the time domain. This means that instead of a diagonal time derivative operator and fully populated nonlinear operator one obtains a fully populated time derivative operator and a diagonal nonlinear operator. So the time-spectral method belongs to the pseudo-spectral methods. In [58, 59] the time-spectral method has been used to simulate periodically moving solid bodies in fluids. This causes problems for the time-spectral method, as the Fourier coefficients have an infinite support in time for every individual spatial grid point. But some spatial grid points can belong to the fluid phase at one point in time and to the solid body at another point in time. The authors suggest to treat these points separately and not to use a Fourier series for them. Instead they suggest to use as local approximation for the temporal subintervals that belong to the fluid phase. To use the same temporal collection points in those subdomains, barycentric rational interpolants are the functions of choice. In [65], the time-spectral method has been extended to quasi-periodic problems. Quasi-periodic problems are driven by a fast strong periodic part and a slow weak transient part, e.g. flying helicopters. The authors suggest to solve such problems by adding to the time-periodic Fourier ansatz a linear or quadratic polynomial. Furthermore the authors suggest to parallelize not only in space but also in time, which means here that different time-slices are advanced on different processors. This has been also pursued in [60], where the parallelized time-spectral method has been compared with a finite difference in time method. The finite difference in time method uses second order central finite differences for the time-derivative operator. This second order central difference operator is quite sparse, such that less communication is necessary as for the fully populated time-spectral operator.

There is also current research done that combines the parallel-in-time method “parareal” with the idea of having periodic boundary conditions in time. In [34] the authors suggest two so-called periodic “parareal” methods. Both methods use the same parallel-in-time step using a fine and coarse integrator in the same way as the standard “parareal” method. But they differ in the correction step. One method solves the whole coarse periodic problem for updating the values. The other method uses the same sequential update process as the standard “parareal” method, but uses as new initial value on the left side of the time interval the value from the previous iteration from the right side of the time interval. The convergence of both methods is analyzed for linear and nonlinear problems. The periodic “parareal” with periodic coarse problem showed overall better convergence, but is also computationally more expensive. This approach has also been further developed by including waveform relaxation for the periodic coarse problem in [84] and for the initial value coarse problem in [85].

1.2.3 Related work

This thesis has been preceded by investigating different methods that directly solve for the time-periodic steady state assuming periodic boundary conditions in time. In [70] the authors proposed a temporal spectral discretization mixed with a spatial finite difference method for the Burgers equation. This discretized problem is solved by a special nonlinear iteration method. The iteration matrix for this iteration method is the block diagonal part of the Newton iteration matrix. Each block contains all spatial values of one mode. It shows relatively slow convergence but a large degree of parallelism.

In [1] and [48] the authors used the finite difference method to discretize the shallow-water equations in space and time. This discretization is parallelized in space and time. The nonlinear problem is solved by an inexact Newton iteration. The linear system has been solved with preconditioned GMRES. The preconditioner is a circulant approximation of the original Newton matrix. The Fourier transformed circulant approximation leads to uncoupled modal problems that can be solved in parallel. These modal problems were again circulant approximated in the spatial directions and then Fourier transformed. The iteration number for the inner problems varied significantly such that the individual modes had to be load balanced.

In [2] and [49] the same space-time discretization and parallelization has been applied to the Navier–Stokes equation. The nonlinear problem has been solved with Newton iteration again. The circulant preconditioner has been replaced by a block incomplete LU-decomposition. Additionally, immersed boundaries have been introduced to the space-time formulation to model an oscillating disc. This method shows good parallelization properties which leads to a strong reduction of the time to solution.

1.3 Outline of the thesis

The goals of this thesis are to develop and to implement new efficient solvers for time-periodic forced Navier–Stokes problems. The new solvers are parallel-in-time, so that the new implementation has better scalability than just using spatial domain decomposition. Furthermore, the solvers are able to circumvent the necessity to simulate long transition phases for obtaining the time-periodic steady state. We introduce two different temporal discretizations in Chapter 2. Namely a spectral time discretization and a finite differences in time discretization. Also the spatial discretization is shown. In Chapter 3 we show how these space-time problems can be solved efficiently. The implementation of these solvers is explained in Chapter 4. The physical problems that are solved for verification and optimization are a two-dimensional Taylor–Green vortex, a two-dimensional Rayleigh streaming, a three-dimensional channel flow which is disturbed by an oscillating sphere, and a periodically disturbed three-dimensional swept Hiemenz flow. The results of these experiments are discussed in Chapter 5. At last we will conclude in Chapter 6.

2

Discretization of the time-periodic Navier–Stokes equations

In this chapter we first introduce the governing equations. The governing equations are the time-periodic Navier–Stokes equations. They are the basis of this thesis. We use two different methods for the temporal discretization of the time-periodic Navier–Stokes equations. In section 2.2.1 we describe a temporal spectral Galerkin method. In section 2.2.2 we describe a temporal finite differences method. These two temporal discretizations lead to continuous spatial problems. We describe a possible discretization of these problems in section 2.3,

2.1 Governing equations

The Navier–Stokes equations are conservation laws for density and momentum of an incompressible Newtonian fluid, such as air or water. The incompressible Navier–Stokes equations are written in their dimensional form [57] as

$$\begin{aligned} \rho\partial'_t\mathbf{u}' + \rho(\mathbf{u}'\cdot\boldsymbol{\nabla}')\mathbf{u}' - \mu\boldsymbol{\Delta}'\mathbf{u}' + \boldsymbol{\nabla}'p' &= \rho\mathbf{f}', \\ \boldsymbol{\nabla}'\cdot\mathbf{u}' &= 0, \end{aligned} \qquad \mathbf{x}'\in\Omega',\ t'>0. \tag{2.1}$$

We use bold variables for vectors. There are three components for the coordinate vector $\mathbf{x}' = [x',\ y',\ z']^T$ and for the dimensional fluid velocity vector $\mathbf{u}' = [u',\ v',\ w']^T$. p' denotes the dimensional pressure, μ the dynamic viscosity coefficient, ρ the density, and $\mathbf{f}'$ an external force density. The dimensional nabla and Laplace operators are defined by $\boldsymbol{\nabla}' = [\partial'_x,\ \partial'_y,\ \partial'_z]^T$, $\boldsymbol{\Delta}' = [\partial'^2_x + \partial'^2_y + \partial'^2_z,\ \partial'^2_x + \partial'^2_y + \partial'^2_z,\ \partial'^2_x + \partial'^2_y + \partial'^2_z]^T$, respectively. To obtain a dimensionless form of the Navier–Stokes equations we use the following substitutions $\mathbf{x}' = L_{\text{ref}}\mathbf{x}$, $t' = (T_{\text{ref}}/2\pi)t$, $\mathbf{u}' = U_{\text{ref}}\mathbf{u}$, $p' = (\rho U_{\text{ref}}^2)p$, and

$\mathbf{f}' = (U_{\mathrm{ref}}^2/L_{\mathrm{ref}})\mathbf{f}$. $\mathbf{x}$, t, $\mathbf{u}$, p, and $\mathbf{f}$ are the according dimensionless variables. U_{ref}, L_{ref} and T_{ref} are reference velocity, reference length, and reference time (typically taken to be the inverse fundamental frequency f'_{ref}). With these substitutions (2.1) becomes

$$\begin{aligned} 2\pi\frac{L_{\mathrm{ref}}}{U_{\mathrm{ref}}T_{\mathrm{ref}}}\frac{\rho U_{\mathrm{ref}}L_{\mathrm{ref}}}{\mu}\partial_t\mathbf{u}+\frac{\rho U_{\mathrm{ref}}L_{\mathrm{ref}}}{\mu}(\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{u}-\boldsymbol{\Delta}\mathbf{u}+\frac{\rho U_{\mathrm{ref}}L_{\mathrm{ref}}}{\mu}\boldsymbol{\nabla}p=\frac{\rho U_{\mathrm{ref}}L_{\mathrm{ref}}}{\mu}\mathbf{f}, \\ \boldsymbol{\nabla}\cdot\mathbf{u}=0, \end{aligned} \quad \mathbf{x}\in\Omega,\ t>0. \tag{2.2}$$

The *Reynolds number* is defined as

$$\mathrm{Re} = \frac{\rho U_{\mathrm{ref}}L_{\mathrm{ref}}}{\mu}. \tag{2.3}$$

The Reynolds number can be seen as the ratio between inertia forces and viscous forces of the according flow [57]. The *Strouhal number* is defined as

$$\mathrm{St} = \frac{L_{\mathrm{ref}}}{U_{\mathrm{ref}}T_{\mathrm{ref}}} = \frac{L_{\mathrm{ref}}f_{\mathrm{ref}}}{U_{\mathrm{ref}}}. \tag{2.4}$$

The Strouhal number can be seen as the ratio between unsteady acceleration and advective acceleration [57]. The *Womersley number* α is the geometric mean of Reynolds number and Strouhal number,

$$\alpha = \sqrt{2\pi\mathrm{St}\mathrm{Re}} = L_{\mathrm{ref}}\sqrt{\frac{\rho\omega}{\mu}}. \tag{2.5}$$

The Womersley number can be seen as the ratio between unsteady inertial forces and viscous forces [57]. As one of the three above numbers can be computed from the other two, we provide just two. Normally we provide the Reynolds number Re and either the Womersley number α or Strouhal number St. We get the dimensionless formulation

$$\begin{aligned} \alpha^2\partial_t\mathbf{u}+\mathrm{Re}(\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{u}-\boldsymbol{\Delta}\mathbf{u}+\mathrm{Re}\boldsymbol{\nabla}p=\mathrm{Re}\,\mathbf{f}, \\ \boldsymbol{\nabla}\cdot\mathbf{u}=0, \end{aligned} \quad \mathbf{x}\in\Omega,\ t>0. \tag{2.6}$$

These equations are complemented by boundary conditions, for example by Dirichlet boundary conditions

$$\mathbf{u}(\mathbf{x},t) = \mathbf{u}_{\mathrm{bc}}(\mathbf{x},t), \quad \mathbf{x}\in\partial\Omega,\ t>0, \tag{2.7}$$

where $\mathbf{u}_{\mathrm{bc}}$ is a prescribed velocity on the boundary $\partial\Omega$. For rectangular domains $\Omega = [0, L_x] \times [0, L_y] \times [0, L_z]$, also periodic boundary conditions are possible

$$\begin{aligned}
\mathbf{u}([0, y, z]^T, t) &= \mathbf{u}([L_x, y, z]^T, t), \quad y \in [0, L_y],\ z \in [0, L_z],\ t > 0,\\
\mathbf{u}([x, 0, z]^T, t) &= \mathbf{u}([x, L_y, z]^T, t), \quad x \in [0, L_x],\ z \in [0, L_z],\ t > 0,\\
\mathbf{u}([x, y, 0]^T, t) &= \mathbf{u}([x, y, L_z]^T, t), \quad x \in [0, L_x],\ y \in [0, L_y],\ t > 0.
\end{aligned} \tag{2.8}$$

Also the combination of different boundary conditions on different parts of the boundary is possible.

2.2 Time discretization

For a time-periodic external forcing $\mathbf{f}'(\mathbf{x}', t') = \mathbf{f}'(\mathbf{x}', t' + 1/f_{\mathrm{ref}})$ and time-periodic boundary conditions $\mathbf{u}'_{\mathrm{bc}}(\mathbf{x}', t') = \mathbf{u}'_{\mathrm{bc}}(\mathbf{x}', t' + 1/f_{\mathrm{ref}})$, the steady-state solution will be periodic in time with the same fundamental frequency f_{ref}. Note that stationary forcing or stationary boundary conditions can be seen as time-periodic. The quadratic nonlinear term transfers energy only to higher harmonics $2f_{\mathrm{ref}}$, $3f_{\mathrm{ref}}$, $\ldots$ or to the stationary mean flow. At this point, we exclude cases with high Reynolds number. A high Reynolds number may lead to a turbulent non-harmonic flow.

Hence, we can assume that in the time-periodic steady state, the condition holds that $\mathbf{u}'(\mathbf{x}', t') = \mathbf{u}'(\mathbf{x}', t' + 1/f_{\mathrm{ref}})$. We employ this condition as periodic boundary conditions in time. So, the equations can be discretized by a multi-harmonic ansatz, which will be discussed in subsection 2.2.1, or with finite differences in four dimension, which will be discussed in subsection 2.2.2. In contrast to the method of lines [81] we first discretize the equations in time and then in space.

2.2.1 Spectral in time discretization

We discretize the time-periodic Navier–Stokes equations (2.6), using a spectral Galerkin method [16]. To this end we use a multi-harmonic ansatz, here a truncated Fourier series,

$$\begin{aligned} \mathbf{u}(\mathbf{x},t) \approx \mathbf{u}^{N_f}(\mathbf{x},t) &= \sum_{l=-N_f}^{N_f} \tilde{\mathbf{u}}_l(\mathbf{x}) \exp(\mathrm{i}lt), \\ p(\mathbf{x},t) \approx p^{N_f}(\mathbf{x},t) &= \sum_{l=-N_f}^{N_f} \tilde{p}_k(\mathbf{x}) \exp(\mathrm{i}lt). \end{aligned} \tag{2.9}$$

We call N_f the number of frequencies. This series can be easily written as a real trigonometric series by using the transformations $\tilde{\mathbf{u}}_l = \left(\hat{\mathbf{u}}^c_{|l|} - \operatorname{sgn}(l)\hat{\mathbf{u}}^s_{|l|}\mathrm{i}\right)/2$ for $l = \pm 1, \ldots \pm N_f$ and $\tilde{\mathbf{u}}_0 = \hat{\mathbf{u}}^0 + \mathbf{0}\mathrm{i}$ and the identity $\exp(\mathrm{i}lt) = \cos(lt) + \mathrm{i}\sin(lt)$. The same applies for the pressure coefficients. We call the $\cdot^0$, $\cdot^c$, and $\cdot^s$ the zero coefficient, the cosine coefficient and the sine coefficient, respectively. We refer to a mode as either the zero mode coefficient, or as the combined cosine and sine coefficient that have the same index. The sign function $\operatorname{sgn}(l)$ extracts the sign of l such that for positive l it gives plus one and for negative l it gives minus one. Thus the real trigonometric series reads as

$$\begin{aligned} \mathbf{u}^{N_f}(\mathbf{x},t) &:= \hat{\mathbf{u}}^0(\mathbf{x}) + \sum_{k=1}^{N_f} \hat{\mathbf{u}}^c_k(\mathbf{x}) \cos(kt) + \sum_{k=1}^{N_f} \hat{\mathbf{u}}^s_k(\mathbf{x}) \sin(kt), \\ p^{N_f}(\mathbf{x},t) &:= \hat{p}^0(\mathbf{x}) + \sum_{k=1}^{N_f} \hat{p}^c_k(\mathbf{x}) \cos(kt) + \sum_{k=1}^{N_f} \hat{p}^s_k(\mathbf{x}) \sin(kt). \end{aligned} \tag{2.10}$$

In our publications [3], [50], and [51] the concise complex formulation (2.9) has been used. But our implementation is using the real formulation (2.10), because it has been easier to integrate it into the preexisting software. We stick to the real formulation for the rest of this thesis to better reflect the implementation.

As test functions we use the set $\{1/2\pi,\ \cos(lt)/\pi,\ \sin(lt)/\pi : 1 \leq l \leq N_f\}$. As inner product for two functions $\phi(t)$ and $\nu(t)$ we use $(\phi,\nu) = \int_0^{2\pi} \phi(t) \cdot \nu(t)\,\mathrm{d}t$. We plug the truncated Fourier series (2.10) into the Navier–Stokes equations (2.6), multiply them with a test function $\phi(t)$, and integrate over the time domain, which gives rise to the weak formulation of the problem

$$\begin{aligned} \alpha^2\int_0^{2\pi}(\partial_t\mathbf{u}^{N_f})\phi\,\mathrm{d}t + \mathrm{Re}\int_0^{2\pi}(\mathbf{u}^{N_f}\cdot\boldsymbol{\nabla})\mathbf{u}^{N_f}\phi\,\mathrm{d}t - \int_0^{2\pi}\boldsymbol{\Delta}\mathbf{u}^{N_f}\phi\,\mathrm{d}t + \mathrm{Re}\int_0^{2\pi}\boldsymbol{\nabla}p^{N_f}\phi\,\mathrm{d}t &= \mathrm{Re}\int_0^{2\pi}\mathbf{f}\phi\,\mathrm{d}t, \\ \int_0^{2\pi}\boldsymbol{\nabla}\cdot\mathbf{u}^{N_f}\phi\,\mathrm{d}t &= 0. \end{aligned} \tag{2.11}$$

We substitute $\mathbf{u}^{N_f}$ and p^{N_f} and isolate the integrals. As this leads to rather long expressions the integrals are treated individually. The weak formulation of the time derivative is

$$\begin{aligned}
\alpha^2\int_0^{2\pi}(\partial_t\mathbf{u}^{N_f})\phi\,\mathrm{d}t =&\alpha^2\int_0^{2\pi}(\partial_t\hat{\mathbf{u}}^0)\phi\,\mathrm{d}t\\
&+\sum_{k=1}^{N_f}\alpha^2\hat{\mathbf{u}}_k^c\int_0^{2\pi}(\partial_t\cos(kt))\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\alpha^2\hat{\mathbf{u}}_k^s\int_0^{2\pi}(\partial_t\sin(kt))\phi\,\mathrm{d}t\\
=&-\sum_{k=1}^{N_f}\alpha^2 k\hat{\mathbf{u}}_k^c\int_0^{2\pi}\sin(kt)\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\alpha^2 k\hat{\mathbf{u}}_k^s\int_0^{2\pi}\cos(kt)\phi\,\mathrm{d}t.
\end{aligned}\tag{2.12}$$

The nonlinear convective term expands to

$$\begin{aligned}
&\mathrm{Re}\int_0^{2\pi}(\mathbf{u}^{N_f}\cdot\boldsymbol{\nabla})\mathbf{u}^{N_f}\phi\,\mathrm{d}t=\\
&\mathrm{Re}(\hat{\mathbf{u}}^0\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}^0\int_0^{2\pi}\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\mathrm{Re}(\hat{\mathbf{u}}^0\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^c\int_0^{2\pi}\cos(kt)\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\mathrm{Re}(\hat{\mathbf{u}}^0\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^s\int_0^{2\pi}\sin(kt)\phi\,\mathrm{d}t\\
&+\sum_{n=1}^{N_f}\mathrm{Re}(\hat{\mathbf{u}}_n^c\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}^0\int_0^{2\pi}\cos(nt)\phi\,\mathrm{d}t+\sum_{n=1}^{N_f}\sum_{k=1}^{N_f}\mathrm{Re}(\hat{\mathbf{u}}_n^c\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^c\int_0^{2\pi}\cos(nt)\cos(kt)\phi\,\mathrm{d}t\\
&+\sum_{n=1}^{N_f}\sum_{k=1}^{N_f}\mathrm{Re}(\hat{\mathbf{u}}_n^c\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^s\int_0^{2\pi}\cos(nt)\sin(kt)\phi\,\mathrm{d}t+\sum_{n=1}^{N_f}\mathrm{Re}(\hat{\mathbf{u}}_n^s\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}^0\int_0^{2\pi}\sin(nt)\phi\,\mathrm{d}t\\
&+\sum_{n=1}^{N_f}\sum_{k=1}^{N_f}\mathrm{Re}(\hat{\mathbf{u}}_n^s\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^c\int_0^{2\pi}\sin(nt)\cos(kt)\phi\,\mathrm{d}t+\sum_{n=1}^{N_f}\sum_{k=1}^{N_f}\mathrm{Re}(\hat{\mathbf{u}}_n^s\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^s\int_0^{2\pi}\sin(nt)\sin(kt)\phi\,\mathrm{d}t
\end{aligned}\tag{2.13}$$

The linear terms (diffusion, gradient, and divergence) give rise to

$$\begin{aligned}
\int_0^{2\pi}\boldsymbol{\Delta}\mathbf{u}^{N_f}\phi\,\mathrm{d}t&=\boldsymbol{\Delta}\hat{\mathbf{u}}^0\int_0^{2\pi}\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\boldsymbol{\Delta}\hat{\mathbf{u}}_k^c\int_0^{2\pi}\cos(kt)\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\boldsymbol{\Delta}\hat{\mathbf{u}}_k^s\int_0^{2\pi}\sin(kt)\phi\,\mathrm{d}t,\\
\int_0^{2\pi}\boldsymbol{\nabla}p^{N_f}\phi\,\mathrm{d}t&=\boldsymbol{\nabla}\hat{p}^0\int_0^{2\pi}\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\boldsymbol{\nabla}\hat{p}_k^c\int_0^{2\pi}\cos(kt)\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\boldsymbol{\nabla}\hat{p}_k^s\int_0^{2\pi}\sin(kt)\phi\,\mathrm{d}t,\\
\int_0^{2\pi}\boldsymbol{\nabla}\cdot\mathbf{u}^{N_f}\phi\,\mathrm{d}t&=\boldsymbol{\nabla}\cdot\hat{\mathbf{u}}^0\int_0^{2\pi}\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\boldsymbol{\nabla}\cdot\hat{\mathbf{u}}_k^c\int_0^{2\pi}\cos(kt)\phi\,\mathrm{d}t+\sum_{k=1}^{N_f}\boldsymbol{\nabla}\cdot\hat{\mathbf{u}}_k^s\int_0^{2\pi}\sin(kt)\phi\,\mathrm{d}t.
\end{aligned}\tag{2.14}$$

After plugging in the different test functions $\phi(t)=1/2\pi$, $\phi(t)=\cos(lt)/\pi$, $\phi(t)=\sin(lt)/\pi$, for $l=1,\dots,N_f$, we get integrals of products of one to three trigonometric functions. To conveniently express the results of these integrals we introduce the

Kronecker symbol,

$$\delta_{lk} = \begin{cases} 1 & \text{if } l = k,\ l \leq N_f \text{ and } k \leq N_f, \\ 0 & \text{otherwise.} \end{cases}$$

All occurring integrals of trigonometric functions in (2.12)–(2.14) are stated in (A.1)–(A.11). For the constant test function $\phi(t) = 1/2\pi$, we arrive at

$$\begin{aligned} \mathrm{Re}(\hat{\mathbf{u}}^0 \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}^0 + \sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_k^c \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_k^c + \sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_k^s \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_k^s - \boldsymbol{\Delta}\hat{\mathbf{u}}^0 + \mathrm{Re}\boldsymbol{\nabla}\hat{p}^0 &= \mathrm{Re}\,\hat{\mathbf{f}}^0, \\ \boldsymbol{\nabla} \cdot \hat{\mathbf{u}}^0 &= 0. \end{aligned} \tag{2.15}$$

So we obtain for the test functions $\phi(t) = \cos(lt)/\pi$ for $l = 1, \ldots, N_f$

$$\begin{aligned} &\alpha^2 l \hat{\mathbf{u}}_l^s + \mathrm{Re}(\hat{\mathbf{u}}^0 \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_l^c + \mathrm{Re}(\hat{\mathbf{u}}_l^c \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}^0 \\ &+ \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^c \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_k^c \left[\delta_{n(k+l)} + \delta_{(n+k)l} + \delta_{(n+l)k}\right] \\ &+ \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^s \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_k^s \left[\delta_{n(k+l)} - \delta_{(n+k)l} + \delta_{(n+l)k}\right] - \boldsymbol{\Delta}\hat{\mathbf{u}}_l^c + \mathrm{Re}\boldsymbol{\nabla}\hat{p}_l^c \ = \mathrm{Re}\,\hat{\mathbf{f}}_l^c, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \boldsymbol{\nabla} \cdot \hat{\mathbf{u}}_l^c \ = 0. \end{aligned} \tag{2.16}$$

Then we obtain for the test functions $\phi(t) = \sin(lt)/\pi$ with $l = 1, \ldots, N_f$

$$\begin{aligned} &-\alpha^2 l \hat{\mathbf{u}}_l^c + \mathrm{Re}(\hat{\mathbf{u}}^0 \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_l^s + \mathrm{Re}(\hat{\mathbf{u}}_l^s \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}^0 \\ &+ \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^c \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_k^s \left[-\delta_{n(k+l)} + \delta_{(n+k)l} + \delta_{(n+l)k}\right] \\ &+ \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^s \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_k^c \left[+\delta_{n(k+l)} + \delta_{(n+k)l} - \delta_{(n+l)k}\right] - \boldsymbol{\Delta}\hat{\mathbf{u}}_l^s + \mathrm{Re}\boldsymbol{\nabla}\hat{p}_l^s \ = \mathrm{Re}\,\hat{\mathbf{f}}_l^s, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \boldsymbol{\nabla} \cdot \hat{\mathbf{u}}_l^s \ = 0. \end{aligned} \tag{2.17}$$

Note that $\hat{\mathbf{f}}^0$, $\hat{\mathbf{f}}_l^c$ and $\hat{\mathbf{f}}_l^s$ are defined by the integrals $(1/2\pi)\int_0^{2\pi} \mathbf{f}\,\mathrm{d}t$, $(1/\pi)\int_0^{2\pi} \mathbf{f}\cos(lt)\,\mathrm{d}t$, and $(1/\pi)\int_0^{2\pi} \mathbf{f}\sin(lt)\,\mathrm{d}t$. In most cases, these integrals can be computed analytically. Otherwise the terms can be computed numerically by an appropriate quadrature rule. This can happen if the periodic forcing is moving in space and time.

So (2.15)–(2.17) are $2N_f + 1$ spatial systems of equations. Each spatial system has one equation for the conservation of momentum and one for the conservation of mass.

Assuming that $\mathbf{u}(\mathbf{x}, t)$ and $p(\mathbf{x}, t)$ exist and are continuously differentiable with regard to t, $\mathbf{u}^{N_f}$ and p^{N_f}, it can be expected that they converge exponentially to the solution with growing N_f [16].

2.2.2 Finite differences in time

Another possibility to discretize the equations (2.6) is using finite differences in time. We use equidistant grid points in time

$$t_l = l\Delta t, \quad \text{for } l = 0, \ldots, N_t - 1, \tag{2.18}$$

where N_t is the number of grid points in the time domain $[0, 2\pi]$. The time mesh width is

$$\Delta t = \frac{2\pi}{N_t}. \tag{2.19}$$

We consider the values of the velocity and pressure just at the time grid points

$$\mathbf{u}(\mathbf{x}, t_l) = \mathbf{u}_l(\mathbf{x}), \quad p(\mathbf{x}, t) = p_l(\mathbf{x}), \quad \mathbf{f}(\mathbf{x}, t) = \mathbf{f}_l(\mathbf{x}), \quad \text{for} \quad l = 0, \ldots, N_t - 1. \tag{2.20}$$

We use a first order backward finite differences stencil for the time derivative, so that we obtain the strong formulation

$$\left.\begin{aligned} \frac{\alpha^2}{\Delta t}(\mathbf{u}_l - \mathbf{u}_{l-1}) + \mathrm{Re}(\mathbf{u}_l \cdot \boldsymbol{\nabla})\mathbf{u}_l - \boldsymbol{\Delta}\mathbf{u}_l + \mathrm{Re}\boldsymbol{\nabla} p_l &= \mathrm{Re}\,\mathbf{f}_l, \\ \boldsymbol{\nabla} \cdot \mathbf{u}_l &= 0, \end{aligned}\right. \quad \text{for} \quad l = 0, \ldots, N_t - 1. \tag{2.21}$$

There are N_t coupled nonlinear systems of equations to solve. Each spatial system has one equation for the conservation of momentum and one for the conservation of mass. The solution of this system is converging with order one in time.

2.3 Spatial discretization

The spectral in time and the finite differences in time discretizations presented in the previous section and partly the solvers presented in the next chapter are independent of the spatial discretization. So, any spatial discretization scheme (such as finite volume method, finite element method, spectral methods, or finite differences method) could be used for (2.15)–(2.17) or for (2.21). In this thesis however the high order finite differences method is used as described in [15], [44] and [45]. A rectangular domain $\Omega = [0, L_x] \times [0, L_y] \times [0, L_z]$ is assumed, such that it can perfectly be approximated by a Cartesian grid $\Omega_{\mathbf{N_x}}$. The Cartesian grid $\Omega_{\mathbf{N_x}}$ has $N_x \times N_y \times N_z$ grid points, with the coordinates

$$\mathbf{x}_{ijk} = \begin{bmatrix} x_i \\ y_j \\ z_k \end{bmatrix} \quad \text{for} \quad \begin{aligned} i &= 1, \dots, N_x, \\ j &= 1, \dots, N_y, \\ k &= 1, \dots, N_z. \end{aligned} \tag{2.22}$$

The distances between grid points can be equal or can be varied such that the accuracy is increased in certain areas of the domain. We assume

$$\begin{aligned} 0 = x_1 < x_2 < \cdots < x_{N_x-1} < x_{N_x} = L_x, \\ 0 = y_1 < y_2 < \cdots < y_{N_y-1} < y_{N_y} = L_y, \\ 0 = z_1 < z_2 < \cdots < z_{N_z-1} < z_{N_z} = L_z. \end{aligned} \tag{2.23}$$

The grid is staggered to avoid the decoupling of the velocity and pressure variables, which otherwise can lead to a checkerboard pattern in the pressure field. The pressure unknowns are defined on the grid vertices $p_{ijk} = p(\mathbf{x}_{ijk})$, and each velocity component is defined on the grid edges that are parallel to their velocity direction, cf. Figure 2.1.

$$\begin{aligned} u_{i+\frac{1}{2}jk} &= u(\mathbf{x}_{i+\frac{1}{2}jk}) \quad \text{for } i = 0, \dots, N_x,\ j = 1, \dots, N_y,\ k = 1, \dots, N_z, \\ v_{ij+\frac{1}{2}k} &= v(\mathbf{x}_{ij+\frac{1}{2}k}) \quad \text{for } i = 1, \dots, N_x,\ j = 0, \dots, N_y,\ k = 1, \dots, N_z, \\ w_{ijk+\frac{1}{2}} &= w(\mathbf{x}_{ijk+\frac{1}{2}}) \quad \text{for } i = 1, \dots, N_x,\ j = 1, \dots, N_y,\ k = 0, \dots, N_z. \end{aligned} \tag{2.24}$$

The staggered coordinates $x_{i+\frac{1}{2}}$, $y_{j+\frac{1}{2}}$, and $z_{k+\frac{1}{2}}$ are not necessarily in the exact middle of their neighboring coordinates. In three dimension we get $N_x \times N_y \times N_z$ grid points for the pressure, $N_x \times N_y \times N_z + N_y \times N_z$ grid points for the velocity in x-direction, $N_x \times N_y \times N_z + N_x \times N_z$ grid points for the velocity in y-direction, and $N_x \times N_y \times N_z + N_x \times N_y$ grid points for the velocity in z-direction.

As described in [44] we compute the finite-difference coefficients directly on the stretched grids from a truncated Taylor series. The kth derivative of a function $f(x)$ at x' is approximated by

$$\partial_x^k f(x') \approx \sum_{i=0}^{n} \eta_i f(x_i) \tag{2.25}$$

Note that for $k=0$ we obtain the interpolation operator. To compute the coefficients η_i

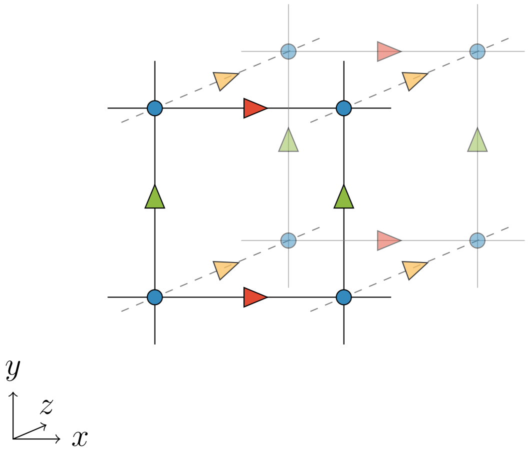


Figure 2.1: Three-dimensional staggered grid cell with grid points for p ●, u ▶, v ▲, and w ▼.

for the n grid points, we define a square Vandermonde matrix $\mathrm{V} \in \mathbb{R}^{n \times n}$ by

$$\mathrm{V}_{ij} = (x' - x_j)^{i-1}, \quad \text{for } i, j = 1, \dots, n. \tag{2.26}$$

The stencil coefficients are computed by the $k+1$-row of the inverse Vandermonde matrix [61, Chapter 1]

$$\eta_i = k! \mathrm{V}^{-1}_{i,1+k} \quad \text{for } i = 1, \dots, n. \tag{2.27}$$

The derivatives of the Polynomials of order $n-1$ are represented correctly by (2.25). Otherwise the convergence order can be expected to be n–1, assuming that the distances of the grid points are evenly distributed and that x' is close to the middle of x_1 and x_n. All stencils to approximate the spacial derivatives in (2.15)–(2.17) and (2.21) can be constructed by (2.27). The divergence operator $\boldsymbol{\nabla} \cdot \mathbf{u} = \partial_x u + \partial_y v + \partial_z w$ is evaluated on the pressure grid. The gradient operator $\boldsymbol{\nabla} p = [\partial_x p,\ \partial_y p,\ \partial_z p]^T$ is evaluated on the individual velocity grids. The diffusion operator $\boldsymbol{\Delta} \mathbf{u} = [\partial_x^2 u + \partial_y^2 u + \partial_z^2 u,\ \partial_x^2 v + \partial_y^2 v + \partial_z^2 v,\ \partial_x^2 w + \partial_y^2 w + \partial_z^2 w]^T$ is also evaluated on the different velocity grids. The convective terms $(\mathbf{u} \cdot \boldsymbol{\nabla})\mathbf{u} = [u\partial_x u + v\partial_y u + w\partial_z u,\ u\partial_x v + v\partial_y v + w\partial_z v,\ u\partial_x w + v\partial_y w + w\partial_z w]^T$ are also discretized on the different velocity grids. Therefore the convective velocities have to be interpolated between the different velocity grids. To increase the stability these derivatives are discretized by an upwinding stencil. An upwinding stencil means that the stencil at each grid point is shifted in the opposite direction of the convection direction.

We use $n = 7$ for inner grid points. At boundaries n reduces up to three. Also for the upwinding stencils n is reduces to five.

3

Numerical solvers for the time-periodic Navier–Stokes problems

In Chapter 2, we introduced the governing equations and showed how they can be discretized in space and time. In this chapter, we show how the discretized nonlinear space-time problem can be solved efficiently.

In general there are many ways to solve a nonlinear system of equation. A general strategy is to linearize the nonlinear system of equations, which leads then to a fixed point iteration. In each iteration step a linear system has to be solved. The two most prominent linearization strategies are Newton's method and Picard's method, which lead to Newton iteration and Picard iteration, respectively. Newton iteration has locally quadratic convergence, whereas Picard iteration has only linear convergence. But Picard iteration has a much larger ball of convergence [25, Chapter 8]. For certain cases it can even be shown that Picard iteration is globally converging [53]. We are valuing a robust method over a potential faster converging method, therefore we are limiting ourself to Picard iteration in this thesis.

We introduce the Picard iteration for the spectral in time discretization in section 3.1 and for finite differences in time in section 3.2.

3.1 A nonlinear solver for the spectral in time discretization

First we formulate the Picard iteration for the spectral in time discretization in 3.1.1. Then we extend the Picard iteration by spectral refinement in 3.1.2 and 3.1.3. Finally we construct an efficient preconditioner for the linearized problem in 3.1.4.

3.1.1 Picard iteration

We apply Picard's method to the equations (2.15)–(2.17). To this end we write the velocity coefficients as

$$\begin{aligned} \hat{\mathbf{u}}^0 &= \hat{\mathbf{u}}^{0^{(m)}} + \delta\hat{\mathbf{u}}^0, \\ \hat{\mathbf{u}}^c_k &= \hat{\mathbf{u}}^{c^{(m)}}_k + \delta\hat{\mathbf{u}}^c_k, \qquad \text{for} \quad k = 1, \dots, N_f. \\ \hat{\mathbf{u}}^s_k &= \hat{\mathbf{u}}^{s^{(m)}}_k + \delta\hat{\mathbf{u}}^s_k, \end{aligned} \tag{3.1}$$

and the pressure coefficients by

$$\begin{aligned} \hat{p}^0 &= \hat{p}^{0^{(m)}} + \delta\hat{p}^0, \\ \hat{p}^c_k &= \hat{p}^{c^{(m)}}_k + \delta\hat{p}^c_k, \qquad \text{for} \quad k = 1, \dots, N_f. \\ \hat{p}^s_k &= \hat{p}^{s^{(m)}}_k + \delta\hat{p}^s_k, \end{aligned} \tag{3.2}$$

In (3.1)–(3.2) quantities indicated by the superscript $\cdot^{(m)}$ denote the solution at the mth Picard iteration step. We write $\delta\cdot$ for the Picard correction. We plug the expression (3.1) and (3.2) into the equations (2.15)–(2.17) to determine the Picard corrections $\delta\cdot$. All terms that just contain coefficients of the current solution are moved to the right hand side. The right hand sides define the residual coefficients. The terms that contain the Picard corrections are kept on the left side. Applying Newton's method all the nonlinear terms that contain the higher order terms $(\delta\hat{\mathbf{u}}^{0/c/s} \cdot \boldsymbol{\nabla})\delta\hat{\mathbf{u}}^{0/c/s}$ are dropped. Applying Picard's method the terms that contain the correction term as the convection velocity $(\delta\hat{\mathbf{u}}^{0/c/s} \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}^{0/c/s^{(m)}}$ are additionally dropped. With that the equations (2.15) of the zero coefficient become

$$\begin{aligned} \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}} \cdot \boldsymbol{\nabla})\delta\hat{\mathbf{u}}^0 + \sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}^{c^{(m)}}_k \cdot \boldsymbol{\nabla})\delta\hat{\mathbf{u}}^c_k + \sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}^{s^{(m)}}_k \cdot \boldsymbol{\nabla})\delta\hat{\mathbf{u}}^s_k & \\ -\boldsymbol{\Delta}\delta\hat{\mathbf{u}}^0 + \mathrm{Re}\boldsymbol{\nabla}\delta\hat{p}^0 &= \hat{\mathbf{r}}_{\mathbf{u}}^{0^{(m)}}, \\ \boldsymbol{\nabla} \cdot \delta\hat{\mathbf{u}}^0 &= \hat{r}_p^{0^{(m)}}, \end{aligned} \tag{3.3}$$

with the residual coefficients

$$\begin{aligned}\hat{\mathbf{r}}_{\mathbf{u}}^{0^{(m)}} :=& \mathrm{Re}\,\hat{\mathbf{f}}^0 - \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}}\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}^{0^{(m)}} - \sum_{k=1}^{N_f}\frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_k^{c^{(m)}}\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^{c^{(m)}} - \sum_{k=1}^{N_f}\frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_k^{s^{(m)}}\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^{s^{(m)}}\\ &+ \boldsymbol{\Delta}\hat{\mathbf{u}}^{0^{(m)}} - \mathrm{Re}\boldsymbol{\nabla}\hat{p}^0,\\ \hat{r}_p^{0^{(m)}} :=& -\boldsymbol{\nabla}\cdot\hat{\mathbf{u}}^{0^{(m)}}.\end{aligned} \tag{3.4}$$

For the equations (2.16) of the cosine coefficient and $l=1,\ldots,N_f$, we obtain

$$\begin{aligned}&\alpha^2 l\delta\hat{\mathbf{u}}_l^s + \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}}\cdot\boldsymbol{\nabla})\delta\hat{\mathbf{u}}_l^c + \mathrm{Re}(\hat{\mathbf{u}}_l^{c^{(m)}}\cdot\boldsymbol{\nabla})\delta\hat{\mathbf{u}}^0\\ &\quad+\sum_{n=1}^{N_f}\sum_{k=1}^{N_f}\frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^{c^{(m)}}\cdot\boldsymbol{\nabla})\delta\hat{\mathbf{u}}_k^c\left[+\delta_{n(k+l)}+\delta_{(n+k)l}+\delta_{(n+l)k}\right]\\ &\quad+\sum_{n=1}^{N_f}\sum_{k=1}^{N_f}\frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^{s^{(m)}}\cdot\boldsymbol{\nabla})\delta\hat{\mathbf{u}}_k^s\left[+\delta_{n(k+l)}-\delta_{(n+k)l}+\delta_{(n+l)k}\right]\\ &\qquad\qquad\qquad\qquad -\boldsymbol{\Delta}\delta\hat{\mathbf{u}}_l^c + \mathrm{Re}\boldsymbol{\nabla}\delta\hat{p}_l^c = \hat{\mathbf{r}}_{\mathbf{u}_l}^{c^{(m)}},\\ &\qquad\qquad\qquad\qquad\qquad\qquad \boldsymbol{\nabla}\cdot\delta\hat{\mathbf{u}}_l^c = \hat{r}_{p_l}^{c^{(m)}},\end{aligned} \tag{3.5}$$

with the residual coefficients

$$\begin{aligned}\hat{\mathbf{r}}_{\mathbf{u}_l}^{c^{(m)}} :=& \mathrm{Re}\,\hat{\mathbf{f}}_l^c - \alpha^2 l\hat{\mathbf{u}}_l^{s^{(m)}} - \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}}\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_l^{c^{(m)}} - \mathrm{Re}(\hat{\mathbf{u}}_l^{c^{(m)}}\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}^{0^{(m)}}\\ &-\sum_{n=1}^{N_f}\sum_{k=1}^{N_f}\frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^{c^{(m)}}\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^{c^{(m)}}\left[+\delta_{n(k+l)}+\delta_{(n+k)l}+\delta_{(n+l)k}\right]\\ &-\sum_{n=1}^{N_f}\sum_{k=1}^{N_f}\frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^{s^{(m)}}\cdot\boldsymbol{\nabla})\hat{\mathbf{u}}_k^{s^{(m)}}\left[+\delta_{n(k+l)}-\delta_{(n+k)l}+\delta_{(n+l)k}\right]\\ &+\boldsymbol{\Delta}\hat{\mathbf{u}}_l^{c^{(m)}} - \mathrm{Re}\boldsymbol{\nabla}\hat{p}_l^{c^{(m)}},\\ \hat{r}_{p_l}^{c^{(m)}} :=& -\boldsymbol{\nabla}\cdot\hat{\mathbf{u}}_l^{c^{(m)}}.\end{aligned} \tag{3.6}$$

For the equations (2.17) of the sine coefficient and $l = 1, \dots, N_f$, we get

$$\begin{aligned}
-\alpha^2 l \delta\hat{\mathbf{u}}_l^c &+ \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}} \cdot \boldsymbol{\nabla})\delta\hat{\mathbf{u}}_l^s + \mathrm{Re}(\hat{\mathbf{u}}_l^{s^{(m)}} \cdot \boldsymbol{\nabla})\delta\hat{\mathbf{u}}^0 \\
&+ \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^{c^{(m)}} \cdot \boldsymbol{\nabla})\delta\hat{\mathbf{u}}_k^s \left[-\delta_{n(k+l)} + \delta_{(n+k)l} + \delta_{(n+l)k}\right] \\
&+ \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^{s^{(m)}} \cdot \boldsymbol{\nabla})\delta\hat{\mathbf{u}}_k^c \left[+\delta_{n(k+l)} + \delta_{(n+k)l} - \delta_{(n+l)k}\right] \\
&\qquad\qquad\qquad - \boldsymbol{\Delta}\delta\hat{\mathbf{u}}_l^s + \mathrm{Re}\boldsymbol{\nabla}\delta\hat{p}_l^s = \hat{\mathbf{r}}_{\mathbf{u}_l}^{s^{(m)}}, \\
&\qquad\qquad\qquad\qquad\qquad \boldsymbol{\nabla} \cdot \delta\hat{\mathbf{u}}_l^s = \hat{r}_{p_l}^{s^{(m)}},
\end{aligned} \tag{3.7}$$

with the residual coefficients

$$\begin{aligned}
\hat{\mathbf{r}}_{\mathbf{u}_l}^{s^{(m)}} :=& \mathrm{Re}\,\hat{\mathbf{f}}_l^s + \alpha^2 l \hat{\mathbf{u}}_l^{c^{(m)}} - \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}} \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_l^{s^{(m)}} - \mathrm{Re}(\hat{\mathbf{u}}_l^{s^{(m)}} \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}^{0^{(m)}} \\
&- \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^{c^{(m)}} \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_k^{s^{(m)}} \left[-\delta_{n(k+l)} + \delta_{(n+k)l} + \delta_{(n+l)k}\right] \\
&- \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_n^{s^{(m)}} \cdot \boldsymbol{\nabla})\hat{\mathbf{u}}_k^{c^{(m)}} \left[+\delta_{n(k+l)} + \delta_{(n+k)l} - \delta_{(n+l)k}\right] \\
&+ \boldsymbol{\Delta}\hat{\mathbf{u}}_l^{s^{(m)}} - \mathrm{Re}\boldsymbol{\nabla}\hat{p}_l^{s^{(m)}}, \\
\hat{r}_{p_l}^{s^{(m)}} :=& - \boldsymbol{\nabla} \cdot \hat{\mathbf{u}}_l^{s^{(m)}}.
\end{aligned} \tag{3.8}$$

The equations (3.3), (3.5), and (3.7) are linear with regard to the correction terms so that they can be solved by a linear solver. We put all the coefficients of the solution, the correction, and the residual into individual vectors

$$\mathbf{q}^{(m)} := \begin{bmatrix} \hat{\mathbf{u}}^{0^{(m)}} \\ \hat{\mathbf{u}}_1^{c^{(m)}} \\ \hat{\mathbf{u}}_1^{s^{(m)}} \\ \vdots \\ \hat{\mathbf{u}}_{N_f}^{c^{(m)}} \\ \hat{\mathbf{u}}_{N_f}^{s^{(m)}} \\ \hat{p}^{0^{(m)}} \\ \hat{p}_1^{c^{(m)}} \\ \hat{p}_1^{s^{(m)}} \\ \vdots \\ \hat{p}_{N_f}^{c^{(m)}} \\ \hat{p}_{N_f}^{s^{(m)}} \end{bmatrix}, \quad \delta\mathbf{q} := \begin{bmatrix} \delta\hat{\mathbf{u}}^{0} \\ \delta\hat{\mathbf{u}}_1^{c} \\ \delta\hat{\mathbf{u}}_1^{s} \\ \vdots \\ \delta\hat{\mathbf{u}}_{N_f}^{c} \\ \delta\hat{\mathbf{u}}_{N_f}^{s} \\ \delta\hat{p}^{0} \\ \delta\hat{p}_1^{c} \\ \delta\hat{p}_1^{s} \\ \vdots \\ \delta\hat{p}_{N_f}^{c} \\ \delta\hat{p}_{N_f}^{s} \end{bmatrix}, \quad \mathbf{r}^{(m)} := \begin{bmatrix} \hat{\mathbf{r}}_{\mathbf{u}}^{0^{(m)}} \\ \hat{\mathbf{r}}_{\mathbf{u}_1}^{c^{(m)}} \\ \hat{\mathbf{r}}_{\mathbf{u}_1}^{s^{(m)}} \\ \vdots \\ \hat{\mathbf{r}}_{\mathbf{u}_{N_f}}^{c^{(m)}} \\ \hat{\mathbf{r}}_{\mathbf{u}_{N_f}}^{s^{(m)}} \\ \hat{r}_{p}^{0^{(m)}} \\ \hat{r}_{p_1}^{c^{(m)}} \\ \hat{r}_{p_1}^{s^{(m)}} \\ \vdots \\ \hat{r}_{p_{N_f}}^{c^{(m)}} \\ \hat{r}_{p_{N_f}}^{s^{(m)}} \end{bmatrix}. \tag{3.9}$$

Each entry in these vectors denotes a full spatial field, such that in case of finite differences one entry contains all values for all the spatial grid points. Now we can write the equations (3.3), (3.5), and (3.7) in matrix form

$$\underbrace{\left(\begin{array}{cccccccccccc} & & & & & & \mathrm{Re}\boldsymbol{\nabla} & & & & & \\ & & & & & & & \mathrm{Re}\boldsymbol{\nabla} & & & & \\ & & \mathcal{F}^{(m)} & & & & & & \mathrm{Re}\boldsymbol{\nabla} & & & \\ & & & & & & & & & \ddots & & \\ & & & & & & & & & & \mathrm{Re}\boldsymbol{\nabla} & \\ & & & & & & & & & & & \mathrm{Re}\boldsymbol{\nabla} \\ \boldsymbol{\nabla}\cdot & & & & & & & & & & & \\ & \boldsymbol{\nabla}\cdot & & & & & & & & & & \\ & & \boldsymbol{\nabla}\cdot & & & & & & & & & \\ & & & \ddots & & & & & & & & \\ & & & & \boldsymbol{\nabla}\cdot & & & & & & & \\ & & & & & \boldsymbol{\nabla}\cdot & & & & & & \end{array}\right)}_{=:\mathcal{H}^{(m)}} \underbrace{\begin{bmatrix} \delta\hat{\mathbf{u}}^{0} \\ \delta\hat{\mathbf{u}}_1^{c} \\ \delta\hat{\mathbf{u}}_1^{s} \\ \vdots \\ \delta\hat{\mathbf{u}}_{N_f}^{c} \\ \delta\hat{\mathbf{u}}_{N_f}^{s} \\ \delta\hat{p}^{0} \\ \delta\hat{p}_1^{c} \\ \delta\hat{p}_1^{s} \\ \vdots \\ \delta\hat{p}_{N_f}^{c} \\ \delta\hat{p}_{N_f}^{s} \end{bmatrix}}_{=\delta\mathbf{q}} = \underbrace{\begin{bmatrix} \hat{\mathbf{r}}_{\mathbf{u}}^{0^{(m)}} \\ \hat{\mathbf{r}}_{\mathbf{u}_1}^{c^{(m)}} \\ \hat{\mathbf{r}}_{\mathbf{u}_1}^{s^{(m)}} \\ \vdots \\ \hat{\mathbf{r}}_{\mathbf{u}_{N_f}}^{c^{(m)}} \\ \hat{\mathbf{r}}_{\mathbf{u}_{N_f}}^{s^{(m)}} \\ \hat{r}_{p}^{0^{(m)}} \\ \hat{r}_{p_1}^{c^{(m)}} \\ \hat{r}_{p_1}^{s^{(m)}} \\ \vdots \\ \hat{r}_{p_{N_f}}^{c^{(m)}} \\ \hat{r}_{p_{N_f}}^{s^{(m)}} \end{bmatrix}}_{=\mathbf{r}^{(m)}}. \tag{3.10}$$

Note that the different Fourier modes are only coupled in $\mathcal{F}^{(m)}$, due to the nonlinearity of the convective term of the Navier–Stokes equations. We call $\mathcal{F}^{(m)}$ the multi-harmonic convection-diffusion operator. $\mathcal{F}^{(m)}$ comprises $2N_f+1$ coupled spatial problems, which means that $\mathcal{F}^{(m)}$ has $2N_f+1$ non-zero block entries. The first row corresponds to the equations for the zero coefficient in (3.3), which is defined as

$$\mathcal{F}_{1j}^{(m)} = \begin{cases} \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}} \cdot \boldsymbol{\nabla}) - \boldsymbol{\Delta} & \text{if } j=1, \\ \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{j/2}^{c^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j \text{ is even,} \\ \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(j-1)/2}^{s^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j \text{ is odd and } j>1. \end{cases} \tag{3.11}$$

If $1<i<2N_f+1$ and i is even then the rows correspond to the equations (3.5) of the cosine coefficients. Those rows are defined as

$$\mathcal{F}_{ij}^{(m)} = \begin{cases} \mathrm{Re}(\hat{\mathbf{u}}_{i/2}^{c^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j=1, \\ \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}} \cdot \boldsymbol{\nabla}) + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{i}^{c^{(m)}} \cdot \boldsymbol{\nabla}) - \boldsymbol{\Delta} & \text{if } j=i, \\ \alpha^2 i/2 + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{i}^{s^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j=i+1, \\ \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(i+j)/2}^{c^{(m)}} \cdot \boldsymbol{\nabla}) + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(i-j)/2}^{c^{(m)}} \cdot \boldsymbol{\nabla}) + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(j-i)/2}^{c^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j \text{ is even and } j \neq i, \\ \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(i+j-1)/2}^{s^{(m)}} \cdot \boldsymbol{\nabla}) - \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(i-j+1)/2}^{s^{(m)}} \cdot \boldsymbol{\nabla}) + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(j-i-1)/2}^{s^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j \text{ is odd and } j \neq i+1. \end{cases} \tag{3.12}$$

Note that $\hat{\mathbf{u}}_l^c = \hat{\mathbf{u}}_l^s = \mathbf{0}$ for $l>N_f$ and $l \leq 0$. If $1<i \leq 2N_f+1$ and i is odd then the rows correspond to the equations (3.7) of the sine coefficients. Those rows are defined as

$$\mathcal{F}_{ij}^{(m)} = \begin{cases} \mathrm{Re}(\hat{\mathbf{u}}_{(i-1)/2}^{s^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j=1, \\ \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}} \cdot \boldsymbol{\nabla}) + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{i-1}^{s^{(m)}} \cdot \boldsymbol{\nabla}) - \boldsymbol{\Delta} & \text{if } j=i, \\ -\alpha^2 j/2 + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{j}^{s^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j=i-1, \\ -\frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(i+j-2)/2}^{c^{(m)}} \cdot \boldsymbol{\nabla}) + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(i-i)/2}^{c^{(m)}} \cdot \boldsymbol{\nabla}) + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(i-j)/2}^{c^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j \text{ is odd and } j \neq i, \\ \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(j+i-1)/2}^{s^{(m)}} \cdot \boldsymbol{\nabla}) + \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(i-j-1)/2}^{s^{(m)}} \cdot \boldsymbol{\nabla}) - \frac{\mathrm{Re}}{2}(\hat{\mathbf{u}}_{(j-i+1)/2}^{s^{(m)}} \cdot \boldsymbol{\nabla}) & \text{if } j \text{ is even and } j \neq i-1. \end{cases} \tag{3.13}$$

The multi-harmonic convection-diffusion matrix $\mathcal{F}^{(m)}$ is nonsymmetric and nonsingular. The individual blocks $\mathcal{F}_{ij}^{(m)}$ are sparse, as we are using finite differences in space. So the matrix $\mathcal{F}^{(m)}$ is sparse even though every block of $\mathcal{F}^{(m)}$ is populated.

The Picard matrix $\mathcal{H}^{(m)}$ is sparse, nonsymmetric, and singular. Because $\mathcal{F}^{(m)}$ is nonsingular and the rows of the spatial gradient operator $\mathrm{Re}\boldsymbol{\nabla}$ sum up to zero, the null space of $\mathcal{H}$ is thus given by

$$\operatorname{null}\Big(\mathcal{H}^{(m)}\Big) = \operatorname{span}\left\{ \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \right\}. \tag{3.14}$$

Here the block entry 1 means that the pressure coefficient is one over the whole spatial domain. The null space has dimension $2N_f + 1$. The pressure coefficients of $\delta\mathbf{q}$ are only given up to a constant. After we solved for $\delta\mathbf{q}$ we orthogonalize it to the null space (3.14), such that the average of each pressure correction coefficient is zero.

For such problems the generalized minimum residual method (GMRES) [79] is the method of choice. GMRES is suited to solve sparse linear systems, as it requires from the matrix only the matrix-vector product. The sparse matrix-vector product is computationally cheap. Also it is known that GMRES converges to a least squares solution for singular problems [14]. We use the zero vector as initial guess for the GMRES solver, $\delta\mathbf{q} = \mathbf{0}$. When we use a preconditioner that varies in each of its application, we use flexible GMRES [77].

The Picard iteration is started with an initial guess $\mathbf{q}^{(0)}$. Then the linear system (3.10) is solved for the Picard correction $\delta\mathbf{q}$. Then the current solution is updated $\mathbf{q}^{(m+1)} = \mathbf{q}^{(m)} + \delta\mathbf{q}$, and a new correction can be computed. This is repeated until the solution has converged. There are different ways to deem a solution converged. We consider three stopping criteria. They can be used individually or combined. The first simple stopping criterion compares the residual norml with a prescribed tolerance

$$\|\mathbf{r}^{(m)}\| < \mathrm{tol}_{\mathbf{r}}. \tag{3.15}$$

This stopping criterion indicates how well the current solution satsifys the equations. But this does not necessarily indicate how close the current solution is to the real solution, especially if the nonlinear system is not well conditioned. Note that we are using an absolute stopping criterion which means that the tolerances have to be choosen carefully and adpeted to the problem. Other possibilites are to consider the residual norm relative to the intial residual, as in [25, Chapter 9]; or a combination of relative and absolute residual can be considered as in [55, Chapter 5].

The second stopping criterion is

$$\|\delta \mathbf{q}\| < \mathrm{tol}_{\delta \mathbf{q}}. \tag{3.16}$$

This stopping criterion is indicating how much the current solution has been changed. Again this does not necessarily mean that the current solution is close to the real solution. In any way it is good to stop if the improvement is minor. The third stopping criterion is a stagnation stopping criterion for $m > 1$

$$\frac{\|\mathbf{r}^{(m)}\|}{\|\mathbf{r}^{(m-1)}\|} > \mathrm{tol}_{\mathrm{stag}} \quad \text{and} \quad \frac{\|\mathbf{r}^{(m-1)}\|}{\|\mathbf{r}^{(m-2)}\|} > \mathrm{tol}_{\mathrm{stag}}. \tag{3.17}$$

This stopping criterion is used for practical and safety reasons. It is indicating that the method has been stagnating for the last two iterations. Reasons for that can be that the discretization error is reached, the linear solver is not finding any good direction anymore. This can occur in combination with backtracking. Backtracking is introduced in the Appendix B. In any case if this criterion is satisfied, we cannot expect any further improvement of the solution.

This completes the Picard iteration, summarized in Algorithm 1. Solving the linear system in each iteration step is the most time consuming part in this method. Thus it is important to solve it as efficiently as possible which is discussed in detail in section 3.1.4.

Algorithm 1 Picard iteration

```
$m \leftarrow 0$.
Choose an initial guess $\mathbf{q}^{(0)}$.
Compute $\mathbf{r}^{(0)}$.
while not converged do
    Solve $\mathcal{H}^{(m)}\delta\mathbf{q} = \mathbf{r}^{(m)}$ for $\delta\mathbf{q}$.
    Update $\mathbf{q}^{(m+1)} \leftarrow \mathbf{q}^{(m)} + \delta\mathbf{q}$.
    Compute $\mathbf{r}^{(m+1)}$.
    $m \leftarrow m + 1$.
```

3.1.2 Solution based spectral refinement

We complement the Picard iteration with *spectral refinement*. We start the solution process with a small number N_f^{start} of Fourier coefficients and execute the Picard iteration until the approximated solution is deemed converged. Then we check if the norm of the highest Fourier coefficient of the solution is smaller than a certain tolerance tol_f scaled by the norm of the lowest Fourier coefficient

$$\frac{\|\hat{\mathbf{u}}_{N_f}\|}{\|\hat{\mathbf{u}}_1\|} \leq \text{tol}_f. \tag{3.18}$$

The norm $\|\hat{\mathbf{u}}_l\|$ reads as the norm of the mode, meaning it is the norm of the combined cosine and sine coefficient $\left\|[\hat{\mathbf{u}}_l^c,\ \hat{\mathbf{u}}_l^s]^T\right\|$. If the criterion (3.18) is not satisfied, we refine the solution. We increase the number of Fourier coefficients N_f by N_f^{inc}, which means that additional Fourier coefficients are appended to the solution vector $\mathbf{q}^{(m)}$. The additional Fourier modes are set to zero initially. The Picard iteration is restarted. This is repeated until the Fourier modes associated with the highest mode are small enough or the requested number N_f^{end} of Fourier coefficients is attained. This method is summarized in Algorithm 2. Spectral refinement has two advantages over fixed N_f. First, the computation is accelerated and, second, the truncation error of the ansatz (2.10) can be controlled. This is demonstrated in section 5.3.

Algorithm 2 Picard iteration with *solution based spectral refinement*

```
$m \leftarrow 0$.
Choose an initial guess $\mathbf{q}^{(0)}$ with $N_f = N_f^{\text{start}}$.
Compute $\mathbf{r}^{(0)}$.
while $N_f \leq N_f^{\text{end}}$ do
    while not converged do
        Solve $\mathcal{H}^{(m)}\delta\mathbf{q} = \mathbf{r}^{(m)}$ for $\delta\mathbf{q}$.
        Update $\mathbf{q}^{(m+1)} \leftarrow \mathbf{q}^{(m)} + \delta\mathbf{q}$.
        Compute $\mathbf{r}^{(m+1)}$.
        $m \leftarrow m + 1$.
    if $\left\|\hat{\mathbf{u}}_{N_f}^{(m)}\right\| < \text{tol}_f \left\|\hat{\mathbf{u}}_1^{(m)}\right\|$ then
        exit.
    else
        Refine the solution vector $\mathbf{q}^{(m)}$ by appending additional Fourier coefficients.
        $N_f \leftarrow N_f + N_f^{\text{inc}}$.
```

3.1.3 Residual based spectral refinement

The *solution based spectral refinement* is an improvement over fixed number of frequencies. But to use the refinement criterion (3.18) it is necessary to solve the non-refined problem till full accuracy is achieved. Otherwise the criterion might stop the iteration prematurely. The problem is solved completely in each refinement level which can lead to large increases in the residual, cf. section 5.3. With hindsight we would like to already have refined when the residual of the non-refined solution is in the same order of magnitude as the refined solution. Luckily we can precisely compute the gain in the residual through a potential refinement. We even can do that efficiently, which means we do not have to construct the whole refined system matrix and do not have to compute the whole matrix-vector product again for that. Having a current solution with the number of frequencies N_f, only N_f additional modes $\hat{\mathbf{u}}_l^{(m)} = \mathbf{0}$ and $\hat{p}_l^{(m)} = 0$ for $l = N_f + 1, \ldots, 2N_f + 1$ can increase the residual. To compute the residual for the refined system we can see from equation (3.10) that the residual coefficients $\hat{\mathbf{r}}_{\mathbf{u}_l}^{(m)}$ and $\hat{r}_{p_l}^{(m)}$ for $l = 1, \ldots, N_f$ are not affected by considering more coefficients. The coefficients $\hat{r}_{p_l}^{(m)}$ are zero for $l = N_f + 1, \ldots, 2N_f + 1$ as the divergence of zero is zero. So the difference in the residual is only due to the terms $\hat{\mathbf{r}}_{\mathbf{u}_l}^{(m)}$ for $l = N_f + 1, \ldots, 2N_f + 1$. For those terms (3.6) and (3.8) simplify to

$$\begin{aligned}
\hat{\mathbf{r}}_{\mathbf{u}_l}^{c(m)} &= -\sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2} (\hat{\mathbf{u}}_n^{c(m)} \cdot \boldsymbol{\nabla}) \hat{\mathbf{u}}_k^{c(m)} \delta_{(n+k)l} + \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2} (\hat{\mathbf{u}}_n^{s(m)} \cdot \boldsymbol{\nabla}) \hat{\mathbf{u}}_k^{s(m)} \delta_{(n+k)l}, \\
\hat{\mathbf{r}}_{\mathbf{u}_l}^{s(m)} &= -\sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2} (\hat{\mathbf{u}}_n^{c(m)} \cdot \boldsymbol{\nabla}) \hat{\mathbf{u}}_k^{s(m)} \delta_{(n+k)l} - \sum_{n=1}^{N_f}\sum_{k=1}^{N_f} \frac{\mathrm{Re}}{2} (\hat{\mathbf{u}}_n^{s(m)} \cdot \boldsymbol{\nabla}) \hat{\mathbf{u}}_k^{c(m)} \delta_{(n+k)l}.
\end{aligned} \tag{3.19}$$

We compute the norm of these residual terms

$$r_l = \|\hat{\mathbf{r}}_{\mathbf{u}_l}^{(m)}\|, \quad l = N_f + 1, \ldots, 2N_f + 1. \tag{3.20}$$

From that we determine the number of additional frequencies N_f^{inc} for which we get a substantial contribution to the overall residual. We want to compute the global solution such that $\|\mathbf{r}^{(m)}\| < \mathrm{tol}_{\mathbf{r}}$. So we take the biggest possible number of additional frequencies $0 \leq N_f^{\mathrm{inc}} \leq N_f$ such that

$$r_{N_f + N_f^{\mathrm{inc}}} > \mathrm{tol}_{\mathbf{r}}. \tag{3.21}$$

Having the number of additional frequencies and the residuals r_l we can compute the residual increase

$$\Delta r = \sqrt{\sum_{l=N_f+1}^{N_f+N_f^{\text{inc}}} r_l^2}. \tag{3.22}$$

We use the refinement criterion

$$\frac{\|\mathbf{r}^{(m)}\|}{\Delta r} < \text{tol}_{\text{refine}}, \tag{3.23}$$

to decide if we refine the solution by N_f^{inc} or not. Namely we refine if the current residual is $\text{tol}_{\text{refine}}$-times smaller than the increase of the residual Δr. This spectral residual based refinement method is summarized in Algorithm 3.

Algorithm 3 Picard iteration with *residual based spectral refinement*

```
1:  $m \leftarrow 0$.
2:  Choose an initial guess $\mathbf{q}^{(0)}$ with $N_f = N_f^{\text{start}}$.
3:  Compute $\mathbf{r}^{(0)}$.
4:  while $\|\mathbf{r}^{(m)}\| > \text{tol}_{\mathbf{r}}$ do
5:      Solve $\mathcal{H}^{(m)}\delta\mathbf{q} = \mathbf{r}^{(m)}$ for $\delta\mathbf{q}$.
6:      Update $\mathbf{q}^{(m+1)} \leftarrow \mathbf{q}^{(m)} + \delta\mathbf{q}$.
7:      $m \leftarrow m + 1$.
8:      Compute $\mathbf{r}^{(m+1)}$, $N_f^{\text{inc}}$, and $\Delta r$.
9:      if $\frac{\|\mathbf{r}^{(m)}\|}{\Delta r} < \text{tol}_{\text{refine}}$ then
10:         Refine the solution vector $\mathbf{q}^{(m)}$ by appending additional Fourier coefficients.
11:         $N_f \leftarrow N_f + N_f^{\text{inc}}$.
```

3.1.4 Preconditioning

As noted before the computationally most expensive portion of Algorithm 1, 2, and 3 is step 5 or 6, in which a large linear system of equations (3.25) has to be solved. We are using GMRES to solve the linear system. The convergence of GMRES depends on the condition number κ of $\mathcal{H}^{(m)}$. Naturally the condition number increases with bigger system sizes and finer discretizations. To ease this deficit we precondition the matrix with an preconditioner $\mathcal{M}_{\mathcal{H}}^{-1}$.

A good preconditioner has two properties. Firstly, systems with the preconditioner can inexpensively be solved. Secondly, the preconditioner is a good approximation of the original linear system, which leads to low iteration numbers of the iterative Krylov solver. Ideally these iteration numbers are then independent of the discretization and the flow problem. An additional desirable property of a preconditioner is that it can be applied in parallel. General preconditioning strategies are algebraic multigrid, incomplete matrix factorization, or matrix splitting methods. These methods are working on general matrices and can be used as a black boxes, but there performance is often not ideal.

We are employing a problem specific preconditioner. This problem specific preconditioner divides the whole problem into subproblems. The different subproblems are shown in the next five subsections. First we start in section 3.1.4.1 with the most outer layer, where the whole Picard system is tackled. This is done in such a way that it possible to deal with the velocity and pressure problems individually. The preconditioner for the velocity problems is then discussed in sections 3.1.4.2–3.1.4.4. The efficient solution of the problem arising for the pressure is shown in section 3.1.4.5.

3.1.4.1 Picard problem

The linear system (3.10) which has to be solved in each Picard step has a saddle point structure. To construct the preconditioner in a compact way we split the vectors $\delta\mathbf{q}$, and $\mathbf{r}^{(m)}$ into two subvectors. One subvector contains all the velocity coefficients, the other subvector contains all the pressure coefficients, such that

$$\delta\mathbf{u} := \begin{bmatrix} \delta\hat{\mathbf{u}}^0 \\ \delta\hat{\mathbf{u}}_1^c \\ \delta\hat{\mathbf{u}}_1^s \\ \vdots \\ \delta\hat{\mathbf{u}}_{N_f}^c \\ \delta\hat{\mathbf{u}}_{N_f}^s \end{bmatrix}, \quad \delta p := \begin{bmatrix} \delta\hat{p}^0 \\ \delta\hat{p}_1^c \\ \delta\hat{p}_1^s \\ \vdots \\ \delta\hat{p}_{N_f}^c \\ \delta\hat{p}_{N_f}^s \end{bmatrix}, \quad \mathbf{r}_{\mathbf{u}}^{(m)} := \begin{bmatrix} \hat{\mathbf{r}}_{\mathbf{u}}^{0(m)} \\ \hat{\mathbf{r}}_{\mathbf{u}_1}^{c(m)} \\ \hat{\mathbf{r}}_{\mathbf{u}_1}^{s(m)} \\ \vdots \\ \hat{\mathbf{r}}_{\mathbf{u}_{N_f}}^{c(m)} \\ \hat{\mathbf{r}}_{\mathbf{u}_{N_f}}^{s(m)} \end{bmatrix}, \quad r_p^{(m)} := \begin{bmatrix} \hat{r}_p^{0(m)} \\ \hat{r}_{p_1}^{c(m)} \\ \hat{r}_{p_1}^{s(m)} \\ \vdots \\ \hat{r}_{p_{N_f}}^{c(m)} \\ \hat{r}_{p_{N_f}}^{s(m)} \end{bmatrix}. \tag{3.24}$$

The linear system (3.10) can then be written in compact form as

$$\begin{pmatrix} \mathcal{F}^{(m)} & \mathcal{G} \\ \mathcal{D} & 0 \end{pmatrix} \begin{bmatrix} \delta\mathbf{u} \\ \delta p \end{bmatrix} = \begin{bmatrix} \mathbf{r}_{\mathbf{u}}^{(m)} \\ r_p^{(m)} \end{bmatrix}. \tag{3.25}$$

The operators $\mathcal{D}$ and $\mathcal{G}$ correspond to the multi-harmonic divergence and gradient operators. They are applied to each Fourier coefficient individually. So they have a block diagonal structure

$$\mathcal{D}_{ij} = \begin{cases} \boldsymbol{\nabla}\cdot & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \qquad \mathcal{G}_{ij} = \begin{cases} \mathrm{Re}\boldsymbol{\nabla} & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \qquad 1 \leq i, j \leq 2N_f + 1. \tag{3.26}$$

The multi-harmonic convection-diffusion operator $\mathcal{F}^{(m)}$ has already been defined in (3.11)–(3.13). The matrix in (3.25) can easily be block LU-decomposed

$$\begin{pmatrix} \mathcal{F}^{(m)} & \mathcal{G} \\ \mathcal{D} & 0 \end{pmatrix} = \begin{pmatrix} \mathcal{I} & 0 \\ \mathcal{D}{\mathcal{F}^{(m)}}^{-1} & \mathcal{I} \end{pmatrix} \begin{pmatrix} \mathcal{F}^{(m)} & \mathcal{G} \\ 0 & -\mathcal{D}{\mathcal{F}^{(m)}}^{-1}\mathcal{G} \end{pmatrix}. \tag{3.27}$$

The matrix $\mathcal{I}$ is the identity matrix. The negative lower right block of the U-factor is called the Schur complement $\mathcal{S} = \mathcal{D}{\mathcal{F}^{(m)}}^{-1}\mathcal{G}$. Note that if the U-factor is used as preconditioner, the GMRES algorithm converges in three steps [67]. But for that the Schur complement has to be inverted. As the Schur complement is dense, it is computationally too expensive to do so. Therefore we use an approximation $\mathcal{M}_{\mathcal{S}}^{(m)}$ of the Schur complement. We employ the block triangular preconditioner

$$\mathcal{M}_{\mathcal{H}} = \begin{pmatrix} \mathcal{F}^{(m)} & \mathcal{G} \\ 0 & -\mathcal{M}_{\mathcal{S}}^{(m)} \end{pmatrix}. \tag{3.28}$$

To solve with this preconditioner it is necessary to approximately solve with the multi-harmonic convection-diffusion operator $\mathcal{F}^{(m)}$ and the approximated Schur complement $\mathcal{M}_{\mathcal{S}}^{(m)}$. Furthermore the application of $\mathcal{G}$ is needed. The efficient solution with the multi-harmonic convection-diffusion operator $\mathcal{F}^{(m)}$ will be discussed in section 3.1.4.2. The efficient solution and approximation of the Schur complement will be discussed next.

There are different choices to construct a good Schur complement approximation $\mathcal{M}_{\mathcal{S}}^{(m)}$. In [54] and [82] the authors proposed to approximate the Schur complement by commuting the gradient operator with the convection-diffusion operator. This is reasonable as the commutator of the continuous gradient and the continuous convection-diffusion operator,

$$((\mathbf{u}\cdot\boldsymbol{\nabla}) - \boldsymbol{\Delta})\boldsymbol{\nabla} - \boldsymbol{\nabla}((\mathbf{u}\cdot\boldsymbol{\nabla}) - \Delta), \tag{3.29}$$

can be assumed to be small for smooth $\mathbf{u}$. We discretize the resulting convection-diffusion operator $\mathcal{F}_p$ for the pressure variable in the same fashion as the velocities variables. We conclude from a small continuous commutator (3.29) that the discretized operators

$$\mathcal{F}\mathcal{G} \approx \mathcal{G}\mathcal{F}_p. \tag{3.30}$$

By left multiplying (3.30) by $\mathcal{D}\mathcal{F}^{-1}$ and right multiplying by $\mathcal{F}_p^{-1}$ we get an approximation of the Schur complement

$$\mathcal{D}\mathcal{G}\mathcal{F}_p^{-1} \approx \mathcal{D}\mathcal{F}^{-1}\mathcal{G} = \mathcal{S}. \tag{3.31}$$

The solution with this Schur complement approximation, involves the application of a convection-diffusion operator and the solution of a Poisson problem $\mathcal{D}\mathcal{G}$. The application of a convection-diffusion operator is computationally cheap anyway and the solution of a Poisson problem can be done very efficiently. A drawback is that the convection-diffusion operator $\mathcal{F}_p$ has to be constructed. This can prevented by expressing $\mathcal{F}_p$ in terms of $\mathcal{D}$, $\mathcal{F}$, and $\mathcal{G}$, which has been proposed by Elman et al. [24]. We determine the columns j of $\mathcal{F}_p$ by minimizing columnwise the squared 2-norm of the discretized commutator

$$\min \left\| [\mathcal{F}\mathcal{G}]_j - \mathcal{G}[\mathcal{F}_p]_j \right\|_2^2 . \tag{3.32}$$

Assuming $\mathcal{D} = \mathcal{G}^T$, 3.32 has the least squares solution by

$$\mathcal{F}_p = (\mathcal{D}\mathcal{G})^{-1}(\mathcal{D}\mathcal{F}\mathcal{G}). \tag{3.33}$$

By plugging (3.33) into (3.31) we obtain

$$(\mathcal{D}\mathcal{G})(\mathcal{D}\mathcal{F}\mathcal{G})^{-1}(\mathcal{D}\mathcal{G}) \approx \mathcal{D}\mathcal{F}^{-1}\mathcal{G}. \tag{3.34}$$

In [24] and [26] it has been show that scaling the velocity variables by $\mathcal{J}$ (determined later) can improve the performance of the preconditioner. So with the scaling $\mathcal{J}$ the Schur complement approximation is

$$\mathcal{M}_{\mathcal{S}}^{(m)} = (\mathcal{D}\mathcal{J}^{-1}\mathcal{G})(\mathcal{D}\mathcal{J}^{-1}\mathcal{F}^{(m)}\mathcal{J}^{-1}\mathcal{G})^{-1}(\mathcal{D}\mathcal{J}^{-1}\mathcal{G}). \tag{3.35}$$

The authors showed that this preconditioner is robust with regard to the mesh size and is moderately sensitive to the Reynolds number. The authors propose to use the diagonal of the velocity mass matrix as $\mathcal{J}$ and a special scaling for equations on the boundaries. As we are using finite differences in space the mass matrix is the identity matrix. In case for the multi-harmonic approach in time, we propose the block diagonal matrix,

$$\mathcal{J}_{ij} = \begin{cases} \mathrm{J} & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{for } 1 \leq i, j \leq 2N_f + 1. \tag{3.36}$$

Here, J is the identity matrix except on the boundaries. It operates on the three velocity grids. In the inner domain, the row of J has a one on the diagonal and is zero everywhere else. On the Dirichlet boundaries that are tangential to the velocity direction, the row of J has ϵ on the diagonal and is zero everywhere else. On the boundaries that belong to Dirichlet boundary conditions and that are normal to the velocity direction, the row contains the coefficients to interpolate the velocity to the boundaries, cf. Figure 3.1. Also these coefficients are scaled with ϵ. The magnitude of ϵ does not have a strong impact to the performance of the solver. The purpose of ϵ is just to scale the equations for the Dirichlet boundary conditions up to give them a stronger weight over the inner points. If not mentioned otherwise we set $\epsilon = 10^2$.

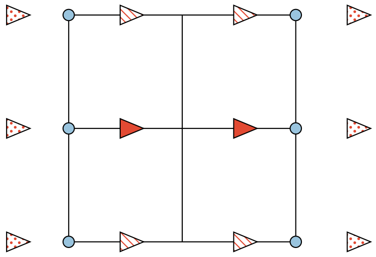


Figure 3.1: The J operator in x-direction. The rows that belong to the inner points ▶, have just a one on the diagonal. The rows that belong to values that are on tangential Dirichlet boundaries ▷ have the scaling factor ϵ on the diagonal. The rows that belong to values that are on normal Dirichlet boundary condition ▷, store the coefficients to interpolate from the velocity grid to the boundary ●. Also these coefficients are scaled with ϵ.

The main advantage of this preconditioner is that the dense the Schur complement is replaced by two efficiently solvable systems (Poisson equations) and the application of $\mathcal{D}$, two times $\mathcal{J}^{-1}$, $\mathcal{H}^{(m)}$, and $\mathcal{G}$. The efficient solution of the Poisson problems is shown in section 3.1.4.5. For the multi-harmonic ansatz (2.10) there is an additional advantage since the structure of $\mathcal{D}\mathcal{J}^{-1}\mathcal{G}$ in (3.26) leads to $2N_f + 1$ independent spatial Poisson problems that can be solved in parallel.

3.1.4.2 Multi-harmonic convection-diffusion problem

In the application of the triangular preconditioner (3.28), a system with the multi-harmonic convection-diffusion operator $\mathcal{F}^{(m)}$ has to be solved

$$\left(\quad \mathcal{F}^{(m)} \quad \right) \begin{bmatrix} \delta\hat{\mathbf{u}}^0 \\ \delta\hat{\mathbf{u}}_1^c \\ \delta\hat{\mathbf{u}}_1^s \\ \vdots \\ \delta\hat{\mathbf{u}}_{N_f}^c \\ \delta\hat{\mathbf{u}}_{N_f}^s \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{r}}_{\mathbf{u}}^{0(m)} \\ \hat{\mathbf{r}}_{\mathbf{u}_1}^{c(m)} \\ \hat{\mathbf{r}}_{\mathbf{u}_1}^{s(m)} \\ \vdots \\ \hat{\mathbf{r}}_{\mathbf{u}_{N_f}}^{c(m)} \\ \hat{\mathbf{r}}_{\mathbf{u}_{N_f}}^{s(m)} \end{bmatrix}. \tag{3.37}$$

We assume that the sine and cosine coefficients $\hat{\mathbf{u}}_l^{c/s}$ for $l = 1, \ldots, N_f$ are much smaller than the zero coefficient $\hat{\mathbf{u}}^0$. So for the preconditioner we set the sine and cosine coefficients to zero. We obtain a block diagonal preconditioner. On the main diagonal we get the stationary convection-diffusion operator $\mathcal{F}_{11}^{(m)} = \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}} \cdot \boldsymbol{\nabla}) - \boldsymbol{\Delta}$. For each harmonic mode l we get an entry $\alpha^2 l \bar{\mathrm{I}}$ on the first upper off-diagonal, and $-\alpha^2 l \bar{\mathrm{I}}$ on the first lower off-diagonal. The elements of $\bar{\mathrm{I}}$ are zero except of the diagonal elements that are not corresponding to equations of Dirichlet or Neumann boundary conditions. These diagonal elements are zero. Thus the preconditioner has the form

$$\mathcal{M}_{\mathcal{F}}^{(m)} = \begin{pmatrix} \mathcal{F}_{11}^{(m)} & & & & & & \\ & \mathcal{F}_{11}^{(m)} & \alpha^2 \bar{\mathrm{I}} & & & & \\ & -\alpha^2 \bar{\mathrm{I}} & \mathcal{F}_{11}^{(m)} & & & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & \mathcal{F}_{11}^{(m)} & \alpha^2 N_f \bar{\mathrm{I}} \\ & & & & & -\alpha^2 N_f \bar{\mathrm{I}} & \mathcal{F}_{11}^{(m)} \end{pmatrix}. \tag{3.38}$$

This preconditioner works very well if the matrix is block diagonally dominant. The matrix $\mathcal{F}^{(m)}$ is block diagonal dominant if the assumption of small higher velocity modes holds. But the matrix $\mathcal{F}^{(m)}$ can be also block diagonal dominant if the assumption of small higher velocity modes does not hold. This is the case when the Womersley number α is large, or the Reynolds number Re is small. In these cases $\mathcal{F}^{(m)}$ can be safely replaced with $\mathcal{M}_{\mathcal{F}}^{(m)}$ in the block triangular preconditioner (3.28). The preconditioner for the Picard problem reads then as

$$\mathcal{M}_{\mathcal{H}}^{(m)} = \begin{pmatrix} \mathcal{M}_{\mathcal{F}}^{(m)} & \mathcal{G} \\ 0 & -\mathcal{M}_{\mathcal{S}}^{(m)} \end{pmatrix}. \tag{3.39}$$

The quality of the preconditioner (3.38) for different Womersley numbers, different Reynolds number and different flows is studied in Chapter 5, especially sections 5.1 and 5.3. The preconditioner (3.38) has the advantage that it leads to N_f+1 independent problems, which makes it appealing for the parallelization in time.

3.1.4.3 Harmonic convection-diffusion problem

As seen in the previous section the application of the block diagonal matrix (3.38) gives rise to N_f independent harmonic convection-diffusion problems

$$\mathrm{K}_l := \begin{pmatrix} \mathcal{F}_{11}^{(m)} & \alpha^2 l \bar{\mathrm{I}} \\ -\alpha^2 l \bar{\mathrm{I}} & \mathcal{F}_{11}^{(m)} \end{pmatrix} \begin{bmatrix} \hat{\mathbf{u}}_l^{c(m)} \\ \hat{\mathbf{u}}_l^{s(m)} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{r}}_{\mathbf{u}_l}^{c(m)} \\ \hat{\mathbf{r}}_{\mathbf{u}_l}^{s(m)} \end{bmatrix}, \quad l = 1, \ldots, N_f. \tag{3.40}$$

We precondition the harmonic convection-diffusion problems by

$$\mathrm{M}_{\mathrm{K}l} = \begin{pmatrix} \mathrm{I} & \mathrm{I} \\ \mathrm{I} & -\mathrm{I} \end{pmatrix} \begin{pmatrix} \alpha^2 l \bar{\mathrm{I}} + \mathcal{F}_{11}^{(m)} & \\ & \alpha^2 l \bar{\mathrm{I}} + \mathcal{F}_{11}^{(m)} \end{pmatrix}. \tag{3.41}$$

The identity matrix is denoted by I. This preconditioner was originally proposed [4] for time-periodic eddy-current problems that have been discretized with a multi-harmonic ansatz in time and finite elements in space. In [5] the authors showed that this preconditioner holds the condition number constant for the linear problem independent of the discretization parameters, using just algebraic arguments.

Following their proof, we introduce the matrix $\mathrm{A} := \alpha^2 l \bar{\mathrm{I}} + \mathcal{F}_{11}^{(m)}$ for the stationary convection-diffusion problem. Looking at the linear case when $\hat{\mathbf{u}}^0 = \mathbf{0}$ and periodic boundary conditions, the matrix $A = \alpha^2 l \bar{\mathrm{I}} - \boldsymbol{\Delta}$ is symmetric positive definite. Such that $(\mathrm{A}\mathbf{x}^c \cdot \mathbf{y}^c) = (\mathbf{x}^c \cdot \mathrm{A}^T \mathbf{y}^c)$ defines a scalar product for the vectors for the vectors $\mathbf{x}^c$ and $\mathbf{y}^c$. Building on that, we introduce a special scalar product and its induced vector norm for the harmonic vectors $\mathbf{x}=[\mathbf{x}^c,\ \mathbf{x}^s]^T$ and $\mathbf{y}=[\mathbf{y}^c,\ \mathbf{y}^s]^T$. Namely,

$$\begin{aligned} (\mathbf{x}\cdot\mathbf{y})_A &:= (\mathrm{A}\mathbf{x}^c\cdot\mathbf{y}^c) + (\mathrm{A}\mathbf{x}^s\cdot\mathbf{y}^s) \quad \text{and} \\ \|\mathbf{x}\|_{\mathrm{A}} &:= \sqrt{(\mathbf{x}\cdot\mathbf{x})_{\mathrm{A}}}. \end{aligned} \tag{3.42}$$

We are computing the condition number of $\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}$ relative to the norm (3.42). The condition number is defined by

$$\kappa\left(\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\right) = \left\|\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\right\|_{\mathrm{A}} \cdot \left\|\left(\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\right)^{-1}\right\|_{\mathrm{A}}. \tag{3.43}$$

To find an upper bound for (3.43) we look at the two terms individually. We start with the first term which can be simplified to

$$\begin{aligned} \mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K} &= \begin{pmatrix} A & \\ & A \end{pmatrix}^{-1} \frac{1}{2}\begin{pmatrix} \mathrm{I} & \mathrm{I} \\ \mathrm{I} & -\mathrm{I} \end{pmatrix}\begin{pmatrix} \mathcal{F}_{11}^{(m)} & \alpha^2 l\bar{\mathrm{I}} \\ -\alpha^2 l\bar{\mathrm{I}} & \mathcal{F}_{11}^{(m)} \end{pmatrix} \\ &= \begin{pmatrix} A & \\ & A \end{pmatrix}^{-1} \underbrace{\frac{1}{2}\begin{pmatrix} -\alpha^2 l\bar{\mathrm{I}} + \mathcal{F}_{11}^{(m)} & \alpha^2 l\bar{\mathrm{I}} + \mathcal{F}_{11}^{(m)} \\ \alpha^2 l\bar{\mathrm{I}} + \mathcal{F}_{11}^{(m)} & \alpha^2 l\bar{\mathrm{I}} - \mathcal{F}_{11}^{(m)} \end{pmatrix}}_{:=\mathrm{B}}. \end{aligned} \tag{3.44}$$

We bound the euclidean scalar product of $\mathrm{B}\mathbf{x}$ with $\mathbf{y}$

$$\begin{aligned} (\mathrm{B}\mathbf{x}\cdot\mathbf{y}) =& \frac{1}{2}\left[\left((-\alpha^2 l\bar{\mathrm{I}} + \mathcal{F}_{11}^{(m)})\mathbf{x}^c\cdot\mathbf{y}^c\right) + (\mathbf{x}^s\cdot\mathbf{y}^c)_{\mathrm{A}} + (\mathbf{x}^c\cdot\mathbf{y}^s)_{\mathrm{A}} + \left((\alpha^2 l\bar{\mathrm{I}} - \mathcal{F}_{11}^{(m)})\mathbf{x}^s\cdot\mathbf{y}^s\right)\right] \\ \le& \frac{1}{2}\left[\|\mathbf{x}^c\|_{\mathrm{A}}\|\mathbf{y}^c\|_{\mathrm{A}} + \|\mathbf{x}^s\|_{\mathrm{A}}\|\mathbf{y}^c\|_{\mathrm{A}} + \|\mathbf{x}^c\|_{\mathrm{A}}\|\mathbf{y}^s\|_{\mathrm{A}} + \|\mathbf{x}^c\|_{\mathrm{A}}\|\mathbf{y}^s\|_{\mathrm{A}}\right] \\ \le& \left[\|\mathbf{x}^c\|_{\mathrm{A}}^2 + \|\mathbf{x}^s\|_{\mathrm{A}}^2\right]^{\frac{1}{2}}\left[\|\mathbf{y}^c\|_{\mathrm{A}}^2 + \|\mathbf{y}^s\|_{\mathrm{A}}^2\right]^{\frac{1}{2}} \\ =& \|\mathbf{x}\|_{\mathrm{A}}\|\mathbf{y}\|_{\mathrm{A}}. \end{aligned} \tag{3.45}$$

Using those two results (3.44) and (3.45), we can bound the first term from (3.43)

$$\left\|\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\right\|_{\mathrm{A}} = \sup_{\mathbf{x}} \frac{\|\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\mathbf{x}\|_{\mathrm{A}}}{\|\mathbf{x}\|_{\mathrm{A}}} = \sup_{\mathbf{x},\mathbf{y}} \frac{\left(\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\mathbf{x}\cdot\mathbf{y}\right)_{\mathrm{A}}}{\|\mathbf{x}\|_{\mathrm{A}}\|\mathbf{y}\|_{\mathrm{A}}} = \sup_{\mathbf{x},\mathbf{y}} \frac{(\mathrm{B}\mathbf{x}\cdot\mathbf{y})}{\|\mathbf{x}\|_{\mathrm{A}}\|\mathbf{y}\|_{\mathrm{A}}} \le 1. \tag{3.46}$$

The second term of (3.43) expands to

$$\left\|(\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K})^{-1}\right\|_{\mathrm{A}} = \sup_{\mathbf{u}} \frac{\|\mathbf{u}\|_{\mathrm{A}}}{\|\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\mathbf{u}\|_{\mathrm{A}}}. \tag{3.47}$$

Then the denominator of (3.47) can be bound

$$\left\|\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\mathbf{u}\right\|_{\mathrm{A}} = \sup_{\mathbf{v}} \frac{\left(\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\mathbf{u}\cdot\mathbf{v}\right)_{\mathrm{A}}}{\|\mathbf{v}\|_{\mathrm{A}}} = \sup_{\mathbf{v}} \frac{(\mathrm{B}\mathbf{u}\cdot\mathbf{v})}{\|\mathbf{v}\|_{\mathrm{A}}} \geq \frac{\left(\mathrm{B}\begin{bmatrix}\mathbf{u}^c\\ \mathbf{u}^s\end{bmatrix}\cdot\begin{bmatrix}\mathbf{u}^s\\ \mathbf{u}^c\end{bmatrix}\right)}{\|\mathbf{u}\|_{\mathrm{A}}} = \frac{1}{2}\|\mathbf{u}\|_{\mathrm{A}}, \tag{3.48}$$

which means that (3.47) is bound by two. Eventually we can bound the condition number (3.43)

$$\kappa\left(\mathrm{M}_{\mathrm{K}}^{-1}\mathrm{K}\right) \leq 2. \tag{3.49}$$

For the application of the preconditioner it is necessary to solve with $\alpha^2 l\mathrm{I} + \mathcal{F}_{11}^{(m)}$, which we replace by the application of a fixed number of multigrid cycles, which is explained in the next section. For a detailed investigation for the performance of different flow types, discretizations and parameters, we refer to Chapter 5.

More recent work [56] proposes another preconditioner for the same harmonic problem. This preconditioner has the form

$$\mathrm{M}'_{\mathrm{K}} = \begin{pmatrix}\alpha^2 l\bar{\mathrm{I}} + \mathcal{F}_{11}^{(m)} & \\ & -\left(\alpha^2 l\bar{\mathrm{I}} + \mathcal{F}_{11}^{(m)}\right)\end{pmatrix}. \tag{3.50}$$

The bound of the condition number for this preconditioner is even better. Namely $\kappa\left(\mathrm{M}_{\mathrm{K}}'^{-1}\mathrm{K}\right) \leq \sqrt{2}$. But in our experiments there has not been a significant performance improvement such that we stick to the preconditioner (3.41).

3.1.4.4 Stationary convection-diffusion problems

Stationary convection-diffusion problems have to be solved for the application of the multi-harmonic (3.38) and the harmonic convection-diffusion preconditioner (3.41). The stationary convection-diffusion problems are written as

$$\left(\beta\bar{\mathrm{I}} + \mathrm{Re}(\hat{\mathbf{u}}^{0^{(m)}}\cdot\boldsymbol{\nabla}) - \boldsymbol{\Delta}\right)\delta\hat{\mathbf{u}} = \hat{\mathbf{r}}_{\mathbf{u}}, \tag{3.51}$$

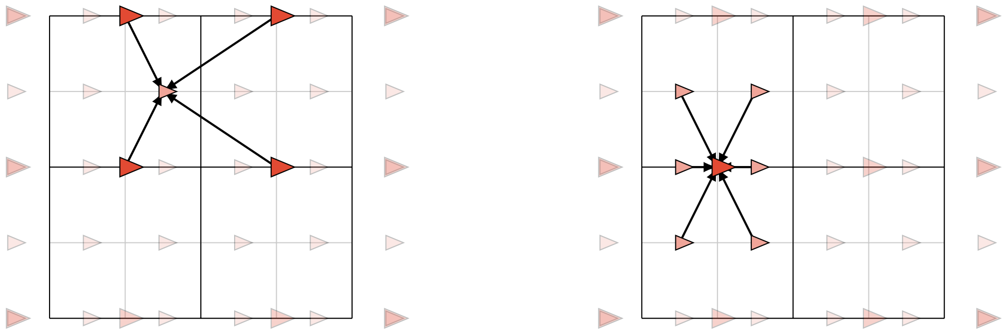
where β can be either zero or a multiple of α^2. $\delta\hat{\mathbf{u}}$ and $\hat{\mathbf{r}}_{\mathbf{u}}$ are simply any unknown correction velocity with an according residual. Note that (3.51) is strictly speaking only a stationary convection-diffusion problem if $\beta = 0$. Otherwise (3.51) is a shifted convection-diffusion problem which is time-independent. Thus we refer to it as *sta-*

tionary convection-diffusion problem, also to have a clear distinction to the harmonic and multi-harmonic convection-diffusion problem. In [92, Chapter 7] it is shown that stationary convection-diffusion problems discretized by a first order upwinding scheme can be efficiently solved by geometric multigrid. If its parameters are chosen properly it can have linear complexity. As we need the solution of a higher order discretized stationary convection diffusion problem, we incorporate the *high order defect correction* method [92, Chapter 5] to the multigrid solver, cf. Algorithm 4.

Algorithm 4 Multigrid V-cycle with *high order defect correction* to solve $A\mathbf{x}=\mathbf{b}$, where A is a high order discretization of an partial differential operator on the fine grid Ω_1. A_j is a lower order discretization, on the grid Ω_j, where j equal one is the finest grid and $j=N_{\text{grids}}$ is the coarsest grid.

```
for $i = 1, \dots, N_{\text{cycles}}$ do
    $\mathbf{r}_1 \leftarrow \mathbf{b} - A\mathbf{x}$.
    for $j = 1, \dots, N_{\text{grids}} - 1$ do
        Pre smooth($A_j$, $\mathbf{r}_j$, $\mathbf{e}_j$)
        $\mathbf{d}_j \leftarrow \mathbf{r}_j - A_j\mathbf{e}_j$
        Restrict $\mathbf{d}_j$ to $\mathbf{r}_{j+1}$
    Coarse grid solve $A_{N_{\text{grids}}}\mathbf{e}_{N_{\text{grids}}} = \mathbf{r}_{N_{\text{grids}}}$
    for $j = N_{\text{grids}} - 1, \dots, 1$ do
        Interpolate $\mathbf{e}_{j+1}$ to $\mathbf{d}_j$
        $\mathbf{e}_j \leftarrow \mathbf{e}_j + \mathbf{d}_j$
        Post smooth($A_j$, $\mathbf{r}_j$, $\mathbf{e}_j$)
    $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{e}_1$
```

Four ingredients have to be defined for the multigrid algorithm, interpolation, restriction, smoothing, and a coarse grid solver. For geometric multigrid we generate coarser grids, by using the same domain but reduce the number of grid points in each dimension. The stationary convection-diffusion operator is then constructed on each of these grids. To transfer the vectors from the coarse grid to the fine grid bi- or trilinear interpolation is used, cf. Figure 3.2a. To transfer the vectors from the fine grid to the coarse grid full weighting restriction is used, cf. Figure 3.2b. Usable smoothers are damped Jacobi or Gauss–Seidel. As coarse grid solver either a smoother or an iterative linear solver such as GMRES can be used. Note, that the performance of the Gauss–Seidel smoother is enhanced for matrices that are close to lower triangular. This occurs, when an upwinding discretization for the convective terms is used, and the unknowns are ordered according to the convection direction. In literature this is called downwinding [10], [40] or down stream relaxation [13], [97]. If there is one dominant flow direction this can be done



(a) Interpolation.

(b) Restriction.

Figure 3.2: Transfer operators for the velocity in x-direction. The black lines, the gray lines, the ▶ marker, and the ▷ marker denote the coarse grid, the fine grid, the velocity variables on the coarse grid, the velocity variables on the fine grid, respectively. The arrows in the left Figure 3.2a show the bilinear interpolation from the coarse to the fine grid. The arrows in the right Figure 3.2b show the restriction from the fine to the coarse grid.

quite easily. The lexicographic numbering has just to be adjusted by starting in the appropriate corner of the domain. For very large Reynolds numbers these smoothers come close to direct solvers [25, Chapter 7]. If the flow is recirculating, this becomes difficult. In this case we use alternating directions of the relaxation.

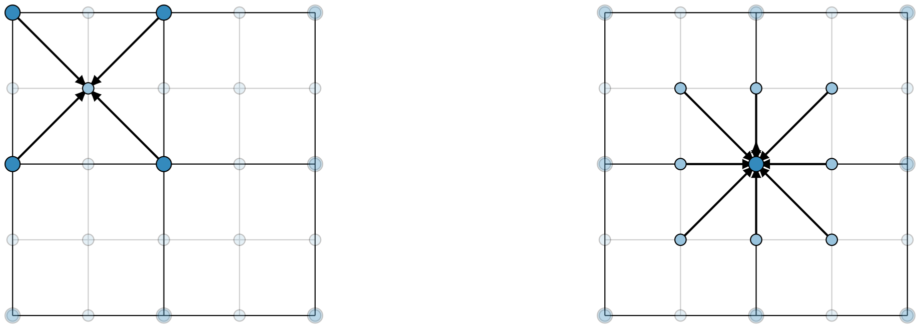
3.1.4.5 Poisson problem

The subproblems that arise in the approximation of the Schur complement (3.35) are Poisson problems. They have to be solved for each pressure coefficient and are written as

$$\left(\mathrm{DJ}^{-1}\mathrm{G}\right)\delta\hat{p} = \hat{r}_p, \tag{3.52}$$

where D and G denote the spatially discretized $\boldsymbol{\nabla}\cdot$ and $\mathrm{Re}\boldsymbol{\nabla}$, respectively. $\delta\hat{p}$ and $\hat{p}_{\mathbf{u}}$ are simply any unknown correction velocity with an according residual. The matrix J has already been introduced in section 3.1.4.1. In case of Dirichlet boundary conditions for the velocities, this problem has Neumann boundary conditions for the pressure which leads to a singular matrix. Also these problems can be solved by an iterative solver with a geometric multigrid preconditioner, cf. Algorithm 4. The convergence of the multigrid method for Poisson problems has been analyzed in great detail and is well understood [92]. We are using standard Lagrangian interpolation and full weighting restriction, illustrated for two dimensions in Figure 3.3. Chebyshev, damped Jacobi, Gauss–Seidel or line smoother can be used. As coarse grid solver either a smoother, an iterative linear solver or a direct solver can be used.

As the matrix in (3.52) is singular it can be necessary to make sure that the right hand side is in the image of the matrix [44]. To achieve that, the right hand side vector can be projected into the image of the matrix. For that the null space vector χ has to be known. A non unique null space vector χ can be computed by solving iteratively $(\mathrm{DJ}^{-1}\mathrm{G})^T \chi = 0$ or can be constructed. The construction of χ and the projection is explained in the appendix C.



(a) Interpolation.

(b) Restriction.

Figure 3.3: Transfer operators for the pressure variables. The black lines, the gray lines, the ● marker, and the • marker denote the coarse grid, the fine grid, the pressure variables on the coarse grid, the pressure variables on the fine grid, respectively. The arrows in the left Figure 3.3a show the bilinear interpolation from the coarse to the fine grid. The arrows in the right Figure 3.3b show the restriction from the fine to the coarse grid.

3.2 A nonlinear solver for the finite differences in time discretization

In the previous section, we introduced a nonlinear solver for the spectral in time discretization from section 2.2.1. In this section however, we introduce a nonlinear solver for the finite differences in time discretization from section 2.2.2. To that end we first use Picard's method to linearize the finite differences in time method in section 3.2.1. This section is quite similar to the section 3.1.1. Then in section 3.2.2 a preconditioner is presented for the linearized problem.

3.2.1 Picard iteration

As for the spectral in time method, we apply Picard's method to the nonlinear equations (2.21), that arise from the finite differences in time. We linearize the velocity and pressure for every time slice $i=0,\dots,N_t-1$

$$\begin{aligned}\mathbf{u}_i &= \mathbf{u}_i^{(m)} + \delta\mathbf{u}_i,\\ p_i &= p_i^{(m)} + \delta p_i.\end{aligned} \tag{3.53}$$

Also here the superscript $\cdot^{(m)}$ denotes the current solution of the mth Picard iteration step. And $\delta\cdot$ denotes the Picard correction. We are plugging the linearization (3.53) into (2.21). All terms that just contain coefficients of the current solution $\cdot^{(m)}$ are put to the right hand side. The right hand side defines the residual for each time slice. The terms that contain the correction terms $\delta\cdot$ are kept on the left side. Applying Picard's method we drop the terms that contain the correction term as the convection velocity $(\delta\mathbf{u}_i\cdot\boldsymbol{\nabla})\mathbf{u}_i^{(m)}$ and the terms that contain the correction term twice $(\delta\mathbf{u}_i\cdot\boldsymbol{\nabla})\delta\mathbf{u}_i$. With that the equations (2.21) at each time slice $i=0,\dots,N_t-1$ become

$$\begin{aligned}\frac{\alpha^2}{\Delta t}(\delta\mathbf{u}_i-\delta\mathbf{u}_{i-1}) + \mathrm{Re}(\mathbf{u}_i^{(m)}\cdot\boldsymbol{\nabla})\delta\mathbf{u}_i - \boldsymbol{\Delta}\delta\mathbf{u}_i + \mathrm{Re}\boldsymbol{\nabla}\delta p_i &= \mathbf{r}_{\mathbf{u}_i}^{(m)},\\ \boldsymbol{\nabla}\cdot\delta\mathbf{u}_i &= r_{p_i}^{(m)}.\end{aligned} \tag{3.54}$$

The residual at each time slice $i=0,\dots,N_t-1$ is defined by

$$\begin{aligned}&\mathbf{r}_{\mathbf{u}_i}^{(m)} := \mathrm{Re}\mathbf{f}_i - \frac{\alpha^2}{\Delta t}(\mathbf{u}_i^{(m)}-\mathbf{u}_{i-1}^{(m)}) - \mathrm{Re}(\mathbf{u}_i^{(m)}\cdot\boldsymbol{\nabla})\mathbf{u}_i^{(m)} + \boldsymbol{\Delta}\mathbf{u}_i^{(m)} - \mathrm{Re}\boldsymbol{\nabla}p_i^{(m)},\\ &r_{p_i}^{(m)} := -\boldsymbol{\nabla}\mathbf{u}_i^{(m)}.\end{aligned} \tag{3.55}$$

We put all the time slices of the solution, the correction, and the residual into one vector each

$$\mathbf{q}^{(m)} = \begin{bmatrix}\mathbf{u}_0^{(m)}\\ \vdots\\ \mathbf{u}_{N_t-1}^{(m)}\\ p_0^{(m)}\\ \vdots\\ p_{N_t-1}^{(m)}\end{bmatrix},\quad \delta\mathbf{q} = \begin{bmatrix}\delta\mathbf{u}_0\\ \vdots\\ \delta\mathbf{u}_{N_t-1}\\ \delta p_0\\ \vdots\\ \delta p_{N_t-1}\end{bmatrix},\quad \mathbf{r}^{(m)} = \begin{bmatrix}\mathbf{r}_{\mathbf{u}_0}^{(m)}\\ \vdots\\ \mathbf{r}_{\mathbf{u}_{N_t-1}}^{(m)}\\ r_{p_0}^{(m)}\\ \vdots\\ r_{p_{N_t-1}}^{(m)}\end{bmatrix}. \tag{3.56}$$

Each entry in these vectors denotes a full spatial field. We write the equations (3.54) in matrix form

$$\underbrace{\begin{pmatrix} \mathcal{F}_{11}^{(m)} & & & \mathcal{F}_{1N_t-1}^{(m)} & \mathrm{Re}\boldsymbol{\nabla} & & & \\ \mathcal{F}_{21}^{(m)} & \mathcal{F}_{22}^{(m)} & & & & \mathrm{Re}\boldsymbol{\nabla} & & \\ & \ddots & \ddots & & & & \mathrm{Re}\boldsymbol{\nabla} & \\ & & \mathcal{F}_{N_t-2N_t-1}^{(m)} & \mathcal{F}_{N_t-1N_t-1}^{(m)} & & & & \mathrm{Re}\boldsymbol{\nabla} \\ \boldsymbol{\nabla}\cdot & & & & & & & \\ & \boldsymbol{\nabla}\cdot & & & & & & \\ & & \ddots & & & & & \\ & & & \boldsymbol{\nabla}\cdot & & & & \end{pmatrix}}_{=:\mathcal{H}^{(m)}} \begin{bmatrix} \delta\mathbf{u}_0 \\ \delta\mathbf{u}_1 \\ \vdots \\ \delta\mathbf{u}_{N_t-1} \\ \delta p_0 \\ \delta p_1 \\ \vdots \\ \delta p_{N_t-1} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_{\mathbf{u}_0}^{(m)} \\ \mathbf{r}_{\mathbf{u}_1}^{(m)} \\ \vdots \\ \mathbf{r}_{\mathbf{u}_{N_t-1}}^{(m)} \\ r_{p_0}^{(m)} \\ r_{p_1}^{(m)} \\ \vdots \\ r_{p_{N_t-1}}^{(m)} \end{bmatrix}. \tag{3.57}$$

Also this Picard matrix $\mathcal{H}^{(m)}$ is sparse, nonsymmetric, and singular. It has N_t different eigenvectors corresponding to the eigenvalue zero. For the same reason that the rows of the gradient operator $\mathrm{Re}\boldsymbol{\nabla}$ sum up to zero. $\mathcal{F}^{(m)}$ is the so-called time-periodic convection-diffusion operator. $\mathcal{F}^{(m)}$ comprises N_t spatial problems. Therefore $\mathcal{F}^{(m)}$ has N_t different block entries. They are defined as

$$\mathcal{F}_{ij}^{(m)} = \begin{cases} \alpha^2\bar{\mathrm{I}} + \mathrm{Re}(\mathbf{u}_i^{(m)} \cdot \boldsymbol{\nabla}) - \mathrm{Re}\boldsymbol{\Delta} & \text{if } i = j, \\ -\alpha^2\bar{\mathrm{I}} & \text{if } i - j = 1 \text{ or } i = 1 \text{ and } j = N_t - 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.58}$$

The Picard iteration for the spectral in time discretization reads the same as for the finite differences in time. So the Algorithm 1 can be used. The difference is in the definition of $\mathbf{q}$, $\mathbf{r}$ and $\mathcal{H}$. Also the previously introduced stopping criteria (3.15)–(3.17) can be used here.

3.2.2 Preconditioning

We used the block structure to precondition the Picard matrix of the multi-harmonic approach in section 3.1.4. The blocks correspond to spatial problems that can be solved by multigrid. For the Picard matrix arising from the finite differences in time (3.57), we propose a multigrid preconditioner (Algorithm 4) for the whole space-time problem. Therefore we are introducing four dimensional restriction and interpolation in section 3.2.2.1. In section 3.2.2.2 we are introducing a box smoother.

3.2.2.1 Four-dimensional restriction and interpolation

The interpolation and restriction in space is the same as for the subproblems for the pressure and the velocity components, cf. Figure 3.2 and 3.3. We combine the spatial interpolation and restriction with temporal interpolation and restriction. Therefore we first interpolate in space and then in time. The temporal coarser grid is given by halving the number of time steps $N_t/2$, such that the coarser grid is given by

$$t_I = I2\Delta t, \qquad I = 0, \ldots, N_t/2 - 1. \tag{3.59}$$

The coarse fields $\bar{\mathbf{u}}$ and $\bar{p}$ are thus given by the already in space coarsened fine fields $\mathbf{u}$ and p by

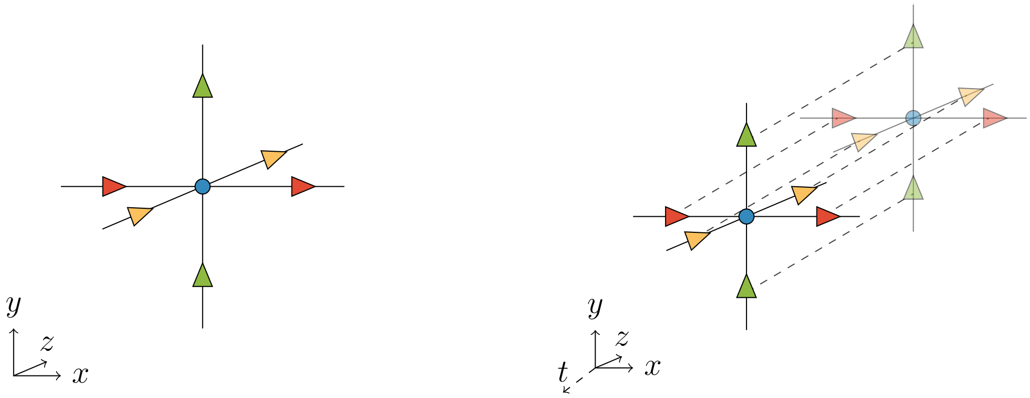
$$\begin{aligned} \bar{\mathbf{u}}_I &= \frac{\mathbf{u}_{I/2-1} + 2\mathbf{u}_{I/2} + \mathbf{u}_{I/2+1}}{4}, \\ \bar{p}_I &= \frac{p_{I/2-1} + 2p_{I/2} + p_{I/2+1}}{4}, \end{aligned} \qquad I = 0, \ldots, N_t/2 - 1. \tag{3.60}$$

The fine fields $\mathbf{u}$ and p are thus given by the already in space interpolated coarse fields $\bar{\mathbf{u}}$ and $\bar{p}$ by

$$\begin{aligned} \mathbf{u}_i &= \begin{cases} \bar{\mathbf{u}}_{i/2} & \text{if } i \text{ is even,} \\ \frac{1}{2}\bar{\mathbf{u}}_{(i-1)/2} + \frac{1}{2}\bar{\mathbf{u}}_{(i-1)/2+1} & \text{otherwise,} \end{cases} \\ p_i &= \begin{cases} \bar{p}_{i/2} & \text{if } i \text{ is even,} \\ \frac{1}{2}\bar{p}_{(i-1)/2} + \frac{1}{2}\bar{p}_{(i-1)/2+1} & \text{otherwise,} \end{cases} \end{aligned} \qquad i = 0, \ldots, N_t - 1. \tag{3.61}$$

3.2.2.2 Box-smoothing

Box-smoothers also called symmetric coupled Gauss-Seidel smoothers were introduced by Vanka [94, 95]. They are smoothing the full discretized Stokes (Re→0) or Navier–Stokes equation. The idea behind box smoothing is that a small number of neighboring variables are relaxed by solving directly a small system for those variables. The so-called box is classically chosen such that one box contains one pressure variable and the two adjacent velocity variables in each direction, cf. Figure 3.4a. We end up with seven unknowns for one box, and seven equations (one for the continuity and one momentum equation for each velocity variable). A 7×7 linear system has to be solved for each box. As each velocity variable belongs to two boxes, it is updated twice. The boxes can be traversed in a lexicographic, red-black or alternating direction fashion. This smoother has been analyzed by local Fourier analysis [63, 83]. Local Fourier analysis has been proposed by Brandt [12] and is a handy tool to analyze the smoothing properties of a smoother on infinite grids.



(a) Three-dimensional box. Seven unknowns per box.

(b) Four-dimensional box. 14 unknowns per box.

Figure 3.4: Box smoother.

The idea of box-smoothers can be extended to four-dimension for the finite differences in time method. In four dimension a box contains two pressure variables at the same spatial position but for two adjacent time steps and all neighboring velocity variables. We end up with 14 unknowns per box cf. Figure 3.4b. We are solving a linear system with 14 equations for each box, two continuum equations and 12 momentum equations. Now every pressure variable belongs to two boxes and each velocity variable to four boxes. They are updated accordingly two and four times. The four-dimensional space-time smoother has been analyzed by local Fourier analysis and experimentally tested in [8, 9]. We have shown that the four-dimensional box has better smoothing properties than the three-dimensional box in theory and experiment. Local Fourier analysis assumes periodicity or an infinite domain. But it still is used to indicate the performance on bounded domains. Therefore we expect the four-dimensional box smoother to work well for spatial bounded domains, which we have observed in experiments. But even more interesting we expect this in the temporal dimension for initial values problems.

4

Implementation of the parallel time-periodic solvers

In Chapter 3 we introduced numerical methods to solve the discretized space-time problems of Chapter 2. In this chapter we show how these space-time solvers have been implemented and parallelized. First we describe the implementation process. Then we describe the basic structure and key ideas, that are necessary to understand and to work with the code. Then we focus on the parallelization.

4.1 Implementation process

We defined as a goal to have an implementation that is as flexible, scalable, and efficient as possible. By flexible we mean that abstraction layers are used such that different temporal and spatial discretizations can be combined, and solvers can be easily constructed. This facilitates the development, the extensibility, and the usability with external libraries. An overview of the implemented interfaces will be given in the next section. The scalability is obtained by parallelizing with MPI[1] [30]. The message passing interface MPI is a library which encapsulates the communication between processors. The paralleliztion will be explained in detail in section 4.3. For the efficiency we rely on low level array manipulation and generic programming. Low level array manipulation allows the compiler to apply many optimizations such as vectorization or loop unrolling. Generic programming in C++ is done using templates. Templates have advantages over object orientation. Namely, overhead is prevented that occurs through virtual function calls, and the compiler can do more potential optimizations as templates are resolved during compile time. Furthermore we required that the spatial operators, especially the finite difference stencils, are equivalent to the so-called IMPACT code [44]. The reason

---

[1] `mpi-form.org`

for that is that results of our new implementation can be directly compared with result of Impact. This has been achieved by reusing parts of Impact, which are written in Fortran90.

The development process has been chosen to be test driven [7]. Test driven development means that firstly an interface or function is defined and then according unit tests are written for testing the defined functionality. Only then is the functionality implemented, which can afterwards immediately be tested. This is beneficial as the developer automatically focuses on providing easy to use interfaces with a clear structure. It is also beneficial as the functionality can be guaranteed and be tested throughout. So problems can be easily spotted that occur by changing the platform or by version changes of depending libraries. Furthermore these unit test can be seen as mini tutorials, which allow a new user to quickly understand what the implementation is capable of and how it is supposed to be used. In that way our implementation is accompanied by over 200 unit test, that can be run with different parameters (parallelization, boundary conditions, and so on). The tests range from tests of simple vector additions, to convergence tests of finite differences operators.

The result of this implementation process including all unit tests is available under an open source license at `github.com/huppd/PINTimpact`. The implementation has been documented with Doxygen[2]. Doxygen is a tool for writing software documentation, where the documentation is written by annotating comments directly into the code. In that way the documentation can be easily kept up to date. Additionally, our implementation is using Trilinos [46]. Trilinos is a large collection of packages, that solves various problems occurring in computational science and engineering applications. We use only three of its packages: the Teuchos package for its smart pointers, parameter lists, and unit testing framework; the Belos package [6] for its linear iterative solvers; and the NOX package for its nonlinear solvers. Note that we are not requiring the Epetra or Tpetra (the linear algebra packages of Trilinos), as we are providing our own vector and operator implementation. We introduce the interfaces to our vector and operator classes in the next section.

---

[2] `doxygen.org`

4.2 Interfaces

As noted before, we have introduced different layers of interfaces to make the implementation flexible. A brief overview over the top layer, that is used to implement the solvers from the previous chapter, is given here. But before that let us give the naming conventions used in the code and this chapter. We use upper camel case for class names and lower camel case for function names. Camel case means that each word or abbreviation in the middle of a name begins with a capital letter with no intervening underscore. Lower camel case means that the first letter of a name is written with a lowercase letter. Upper camel case means that the first letter of a name is written with uppercase letter. Class member names are written with lower camel case and are ended with an underscore. Aliases for class types are indicated by the suffix capital letter T. We show different UML-diagrams, but they only show the most important dependencies and are not extensive, also signatures have been simplified. For an extensive overview we refer to the Doxygen documentation. Note that we use a typewriter font to indicate C++ classes, functions or concepts.

In the center of our implementation is the `Grid` class. The `Grid` class represents a space-time grid. This class has four template parameters; `ScalarT` which denotes the type for scalar, usually `double`; the `OrdinalT` which is the type for indexing, usually `int`; the integer `sdimension` can be two or three and denotes the spatial dimension; the integer `dimension` can be three or four and denotes the dimension of the space-time grid; the integer `dNC` denotes a parameter to choose the order of finite differences. The `Grid` class is composited of different classes, such as `StencilWidths`, `DomainSize`, `BoundaryConditions`, `GridSize`, `IndexSpace`, `ProcGrid`, and `Coordinates`. We are not explaining them in detail, but want to remind that we are using a staggered grid, which means that in the `Coordinates` class coordinates for the pressure grid points and the individual velocity grid points are stored.

In Figure 4.1, we show the two main concepts that are built on top of the `Grid` class namely the `Field` concept and the `Operator` concept. In generic programming a concept is a description of supported operations and function members of a type. So concepts are similar to abstract base types but concepts do not require to inherit from a base type.

The field concept makes sure that the functionality is provided that is needed by the iterative solvers of BELOS and NOX. The required member function to model the

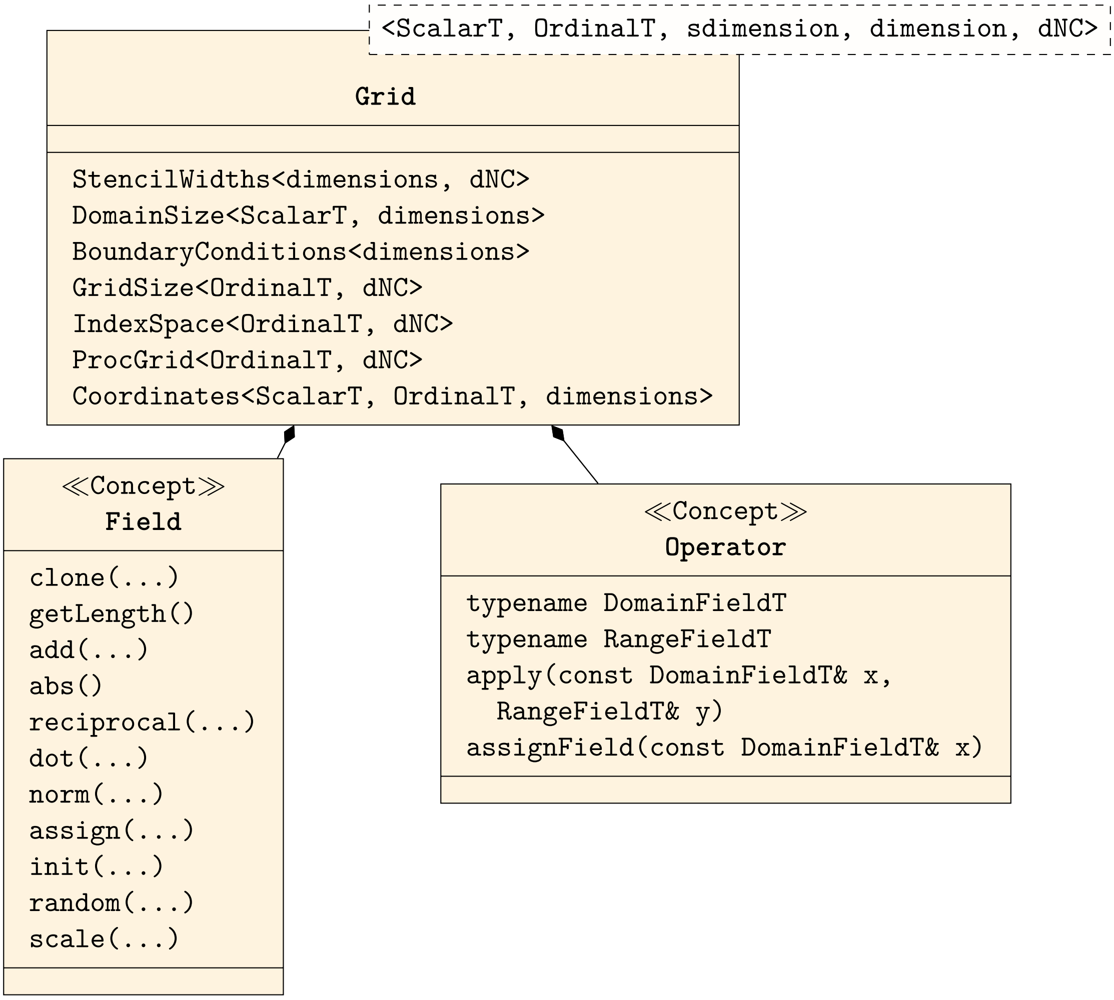

<ScalarT, OrdinalT, sdimension, dimension, dNC>
Grid
StencilWidths<dimensions, dNC>
DomainSize<ScalarT, dimensions>
BoundaryConditions<dimensions>
GridSize<OrdinalT, dNC>
IndexSpace<OrdinalT, dNC>
ProcGrid<OrdinalT, dNC>
Coordinates<ScalarT, OrdinalT, dimensions>
≪Concept≫
Field
clone(...)
getLength()
add(...)
abs()
reciprocal(...)
dot(...)
norm(...)
assign(...)
init(...)
random(...)
scale(...)
≪Concept≫
Operator
typename DomainFieldT
typename RangeFieldT
apply(const DomainFieldT& x,
RangeFieldT& y)
assignField(const DomainFieldT& x)

Figure 4.1: UML-diagram.

concept `Field` are a `clone` method that allows to create copies of a field; a `getLength` method that returns the number of elements in a field in the sense of a vector; an `add` method that adds two fields; an `abs` method that takes elementwise the absolute value; a `reciprocal` method that takes elementwise the reciprocal; a `dot` method that takes the scalar product with another field; a `norm` method that returns the norm of a field; an `init` method that initializes a field with a scalar value; a `random` method that initializes a field with random vectors; two `scale` methods one that scales the field by a scalar, a second that scale with another field.

The `Operator` concept is more concise. Two member types have to be defined, the `DomainFieldT` and the `RangeFieldT`. Ideally these two types model the `Field` concept. To implement the `Operator` concept also a member function `apply` has to be provided that takes a field of type `DomainFieldT` and returns the result of the applied operator in a field of the type `RangeFieldT`. Furthermore the `Operator` concept contains the member function `assignField`, which is necessary for operators that depend on a field like the convection operators. In that case this field can be assigned by the `assignField` function. In case the operator is not depending on a field this function can be implemented empty.

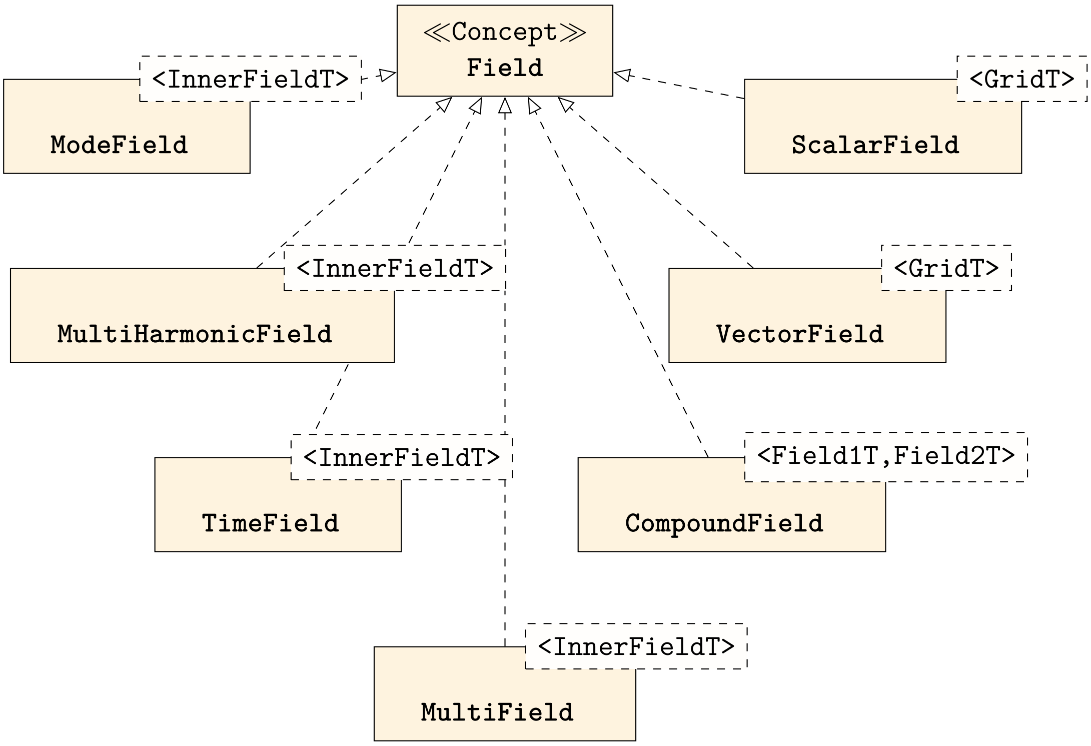


Figure 4.2: UML-diagram for the different field classes.

In Figure 4.2, we see the different field classes that model the `Field` concept. Furthermore they are managing their memory and the exchange of ghost layers, which will be explained in the next section. Each field or vector that has been defined in the previous chapter has its own C++-class representation. The spatial pressure field p is represented by the `ScalarField` class; the spatial velocity field $\mathbf{u}$ is represented by the `VectorField` class; the vector that contains a cosine and a sine inner field $[\cdot^c,\ \cdot^s]^T$ is represented by the `ModeField<·>` class; the vector that contains the complete series of coefficients $[\cdot^0, \cdot^c_1, \cdot^s_1, \ldots, \cdot^c_{N_f}, \cdot^s_{N_f}]^T$ is represented by the `MultiHarmonicField<·>` class; the vector that contains a time series $[\cdot_0, \ldots, \cdot_{N_t-1}]^T$ is represented by the `TimeField<·>` class; and the vector that contains two different types of fields $[\cdot, \cdot]^T$ is represented by the `CompoundField<·, ·>` class. Furthermore we provide the `MultiField` class, which adds the possibility to use multiple fields at once in a block fashion. This is necessary for certain linear iterative solvers. These flexible classes allow to combined them in different ways. For example we can define a `CompoundField<MultiHarmonicField<·>,MultiHarmonicField<·>>` class or a `MultiHarmonicField<CompoundField<·> >` class. Both classes provided the same functionality but differ in the acces of the inner fields and how their values are laid out in memory.

These fields are accompanied by operator classes that model the `Operator` concept. In Figure 4.3, the most important operator classes are shown. An operator class constructs the stencils and manages the according memory if necessary. The stencils are stored for each dimension individually, so we get for each dimension an array with the length of number of locally stored grid points times the number of used stencil coefficients. It also provides an apply method that takes a domain field and returns the result of the applied operator in a range field. Also each operator that has been defined in the previous chapter has its own C++-class representation. There are the basic operators that can be applied on `ScalarField` or `VectorField`, which are the divergence operator $(\boldsymbol{\nabla}\cdot)$ that is represented by the `DivOp` class; the gradient operator $(\mathrm{Re}\boldsymbol{\nabla}\cdot)$ represented by the `GradOp` class; the Laplace operator $(\boldsymbol{\Delta})$ represented by the `DiffusionOp` class; the convection operator $(\mathbf{u}\cdot\boldsymbol{\nabla})$ represented by the `ConvectionOp`. Additionally, we provide the operator classes `InterpolateV2SOp` and `InterpolateS2VOp`, that interpolate between the velocity and pressure grids.

Here we illustrate the apply method for the `DivOp` class. For this operator class the `DomainFieldT` type is a `VectorField<GridT>` and the `RangeFieldT` type is `ScalarField<GridT>`. Both have to use the same `GridT` type as template parameter.

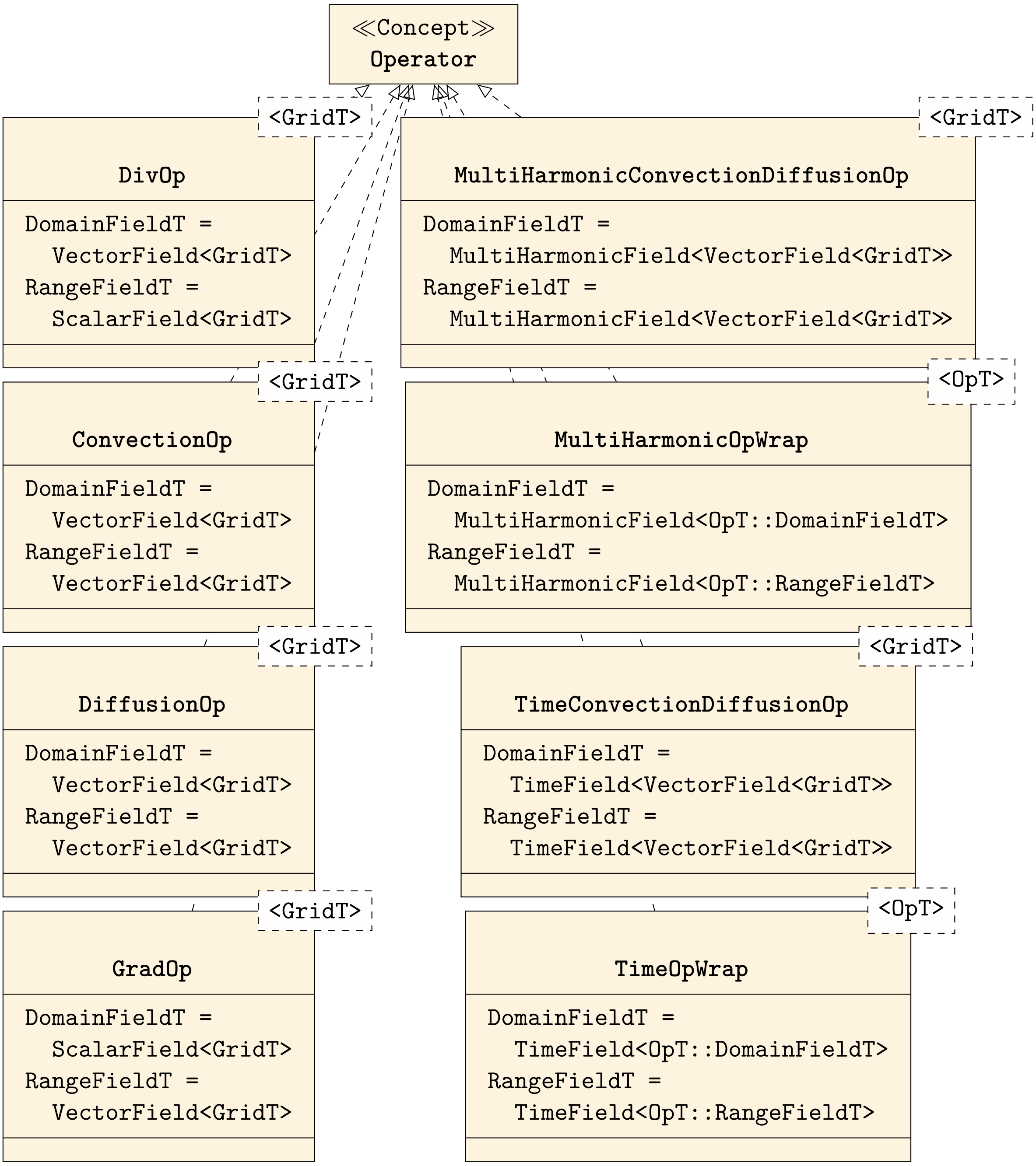


Figure 4.3: UML-diagram for the most important operator classes.

```
void apply(const VectorField<GridT>& x,
  ScalarField<GridT>& y) const {

using OT = typename GridT::OrdinalT;

for(int dir=0; dir<GridT::sdimension; ++dir)
  x.exchange(dir, dir);

for(OT k=grid_->si(F::S,Z);k<=grid_->ei(F::S,Z);++k)
  for(OT j=grid_->si(F::S,Y);j<=grid_->ei(F::S,Y);++j)
    for(OT i=grid_->si(F::S,X);i<=grid_->ei(F::S,X);++i)
      y(i, j, k) = innerStenc(x, i, j, k);

y.changed();
}
```

For simplification we use an alias for the `OrdinalT` type. The `exchange` method updates the ghost layer. The first function parameter indicates the velocity component and the second parameter indicates the direction in which has to be communicated. The operator has pointer `grid_` to an instance of type `GridT`. We are looping over all grid points. The start index and end index are given by the function `si` and `ei`. Both functions take as parameter an enum class `F`, which indicates the type of grid, as the divergence is computed on the scalar grid, it is of value `S`. The second parameter here indicates in which dimension. The `innerStenc(x, i, j, k)` function adds the first derivative in `X` direction of `x(F::U)`, with the first derivative `x(F::V)` in `Y` direction, and with the first derivative in `Z` direction `x(F::W)`. In the end, we indicate that the local values of `y` have been modified by the `changed` method.

But there are also operators that are applied to the `MultiHarmonicField` class, such as the `MultiHarmonicConvectionDiffusionOp` that represents the multi-harmonic convection-diffusion operator $\mathcal{F}$. But also the `MultiHarmonicOpWrap`, that has as template parameter another operator which will be applied to every coefficient in `MultiHarmonicField` individually, which corresponds the multi-harmonic diagonal operator for the inner operator.

Here we illustrate `MultiHarmonicOpWrap`, the `DomainFieldT` type and `RangeFieldT` type are `MultiHarmonicField` version of the `DomainFieldT` and `RangeFieldT` of the operator type that is wrapped.

```
void apply(
  const MultiHarmonicField<typename OpT::DomainFieldT>& x,
  MultiHarmonicField<typename OpT::RangeFieldT>& y) const {

using OT = typename GridT::OrdinalT;

if(0==grid_->si(F::S, T))
  op_->apply(x.get0Field(), y.get0Field());

for(OT i=std::max(grid_)->si(F::S, T), 1);
    i<=grid_->ei(F::S, T); ++i) {
  op_->apply(x.getCField(i), y.getCField(i));
  op_->apply(x.getSField(i), y.getSField(i));
}

y.changed();
}
```

This operator basically applies the wrapped operator to all locally stored coefficients. So, no communication is necessary. The class stores an pointer `op_` to the wrapped operator. Also here are the locally stored coefficients are indicated by the member functions `si` and `ei` of the grid class. The member function `get0Field`, `getCField(i)`, and `getSField(i)`, return a reference to the zero coefficient, to the ith cosine coefficient, and to the ith sine coefficient, respectively.

Similarly there are also operators that are applied to the `TimeField` classes, such as the `TimeConvectionDiffusionOp` that represents the time convection-diffusion operator $\mathcal{F}$. But also the `TimeOpWrap`, that works in the same way as the `MultiHarmonicOpWrap`, i.e., every inner operator is applied to each temporal time-step individually.

4.3 Parallelization

In this section we explain the parallelization using MPI. First we show how the data is distributed by domain decomposition. Then we explain how the communication is handled.

The domain decomposition of the spatial domain it is described in detail in [44]. In short the spatial domain is partitioned into $np_x \times np_y \times np_z$ subdomains, such that $(N_x/np_x) \times (N_y/np_y) \times (N_z/np_z)$ grid points belong to one MPI-rank. We require that the number of grid points N is integer divisible by the number of ranks np in each dimension and that the resulting subdomain size is equal or larger than the stencil width. On top of the spatial parallelization also the temporal domain is partitioned into np_t subdomains. For the spectral in time approach this means that the individual Fourier coefficients are distributed. We allow a maximum of N_f+1 MPI-ranks in the time domain. Theoretically, it would be also possible to distribute the coefficients of one mode to two different MPI-ranks, but this has not been implemented as the benefit can be assumed to be rather small. For the finite differences in time approach the different time steps are divided into different time slices. In total each MPI-rank is responsible for $((2N_f + 1)/np_t) \times (N_x/np_x) \times (N_y/np_y) \times (N_z/np_z)$ grid points for the spectral in time method or $(N_t/np_t) \times (N_x/np_x) \times (N_y/np_y) \times (N_z/np_z)$ grid points for the finite differences in time. Usually we run our applications such that every MPI-rank is attributed to one core.

MPI allows to add a virtual topology to an MPI-communicator, for example a Cartesian topology. The Cartesian topology simplifies the communication with directly neighboring MPI-ranks and the creation of subcommunicators containing only one or multiple subdimensions. The MPI-topology is oblivious of the underlying Cartesian grid, such that communication of for example ghost layer exchange cannot directly be handled by MPI. The MPI runtime environment can also match this virtual topology to the hardware network topology. We add such a four-dimensional Cartesian topology to the according communicator. This has been abstracted into a `ProcGrid` class. The `Grid` class has a pointer to a `ProcGrid`, so that each field that models the `Field` concept has access to it. The process grid is illustrated for one temporal and two spatial dimensions in Figure 4.4.

The user has to set the parameters np_x, np_y, np_z, and np_t. These parameter are best chosen such that communication is minimized. The communication can be minimized by having the largest volume with the smallest surface for each subdomain, which is the

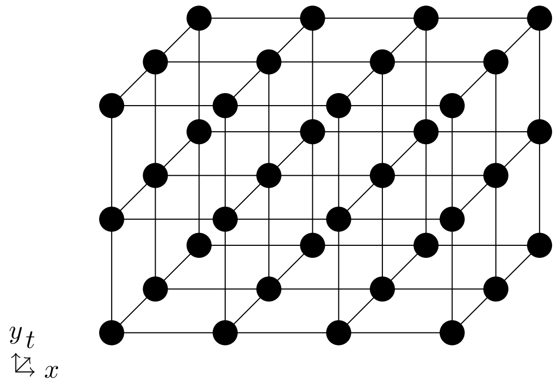


Figure 4.4: Process grid with a three-dimensional Cartesian topology ($np_x = 4$, $np_y = 3$, $np_t = 3$). The black spheres correspond to MPI-ranks.

case for cubical subdomains. If a downwinding Gauss–Seidel smoother is used for the convection-diffusion problems, the quality of this smoother is improved if the subdomains are elongated in the downwind direction. In this case one has to trade between smoother quality and minimal communication.

After setting up the domain decomposition, the parallelization and communication is mostly hidden from the user and the solver developer. This makes using and developing the code easier and is less error-prone. The main communication is internally handled by the individual field classes. There are two kinds of communication patterns. The first kind are collective communication such as reductions that have to be done, when norms or scalar products are called; or gathers that have to be done by specific multi-harmonic operators. The second kind is the exchange of ghost layers. Ghost layers are grid cells that are additionally stored on each rank and are mirroring grid cells that are stored on different ranks. They have to be updated by point to point communication whenever grid points have to be accessed that are stored by neighboring ranks.

Reductions are done when an user or an external library is calling the norm or scalar product of a field instance. Then the field is computing locally the norm or scalar product, and afterwards an `MPI_Allreduce` is done. The field decides on which communicator this is done, so for `ScalarField`, `VectorField`, and `ModeField` this done on the subcommunicator that contains only the ranks from the same time slice or frequency slice, see Figure 4.5. The `TimeField` and `MultiHarmonicField` use the whole space-time communicator. The `CompoundField` and `MultiField` use the inner fields communicator so either the communicator for the temporal slice, or the whole space-time communicator.

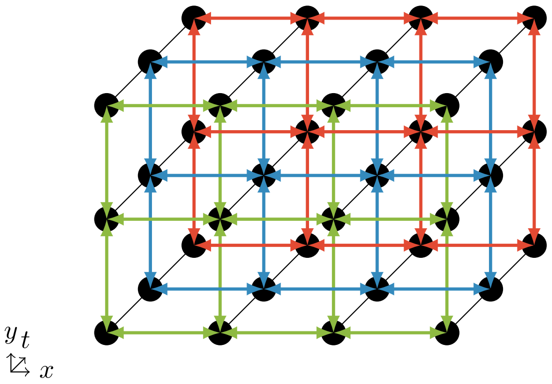


Figure 4.5: Independent parallel-in-time communication pattern.

As said before the exchange of the ghost layers is only necessary when values have to be accessed that are not stored in local memory but in the memory of MPI-ranks. This only happens by the application of stencils, which occurs in the apply method of certain operator classes. So if those operators need to access this data, they call an exchange method which is implemented in the according field class. Each field has an exchange state witch is set to true, if the ghost layers have been updated and is set to false if the local field values are modified. This makes sure that for applying different operators successively to a field does not lead to repeated unnecessary updates of the ghost layer. This is done for the `ScalarField`, `VectorField`, `TimeField`. Also for certain multi-harmonic operators it is necessary to communicate the individual coefficients stored in the `MultiHarmonicField`, which are strictly speaking no ghost layer, but are handled in the same way. We discuss them separately.

We are starting with the `ScalarField` class. The `VectorField` class is basically a composition of multiple `ScalarField` classes, so the communication works the same for each component as for `ScalarField`. The reason for the composition is that each component of the velocity is defined on a different grid. The exchange of the ghost layer has been already explained in [44] and is done the same way for the most operators. Only for the restriction operator it differs, as we introduced full weighting restriction instead of half weighting restriction. Therefore we have to communicate also values to diagonal neighboring ranks. We are doing this by consecutively exchanging in each direction and widen the ghost layer length, for two dimensions see Figure 4.6. These point to point communications are done with `MPI_Send` and `MPI_Irecv`.

The `TimeField` class implements its own exchange method to communicate its ghost layers. As for the time derivative also a finite difference stencil is used, the ghost layer

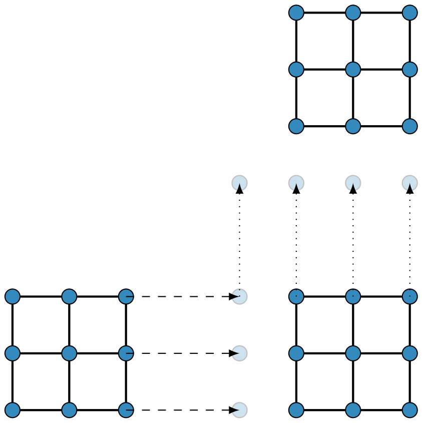


Figure 4.6: Exchange of ghost layers, such that diagonal grid points are exchanged too. First values are sent along the dashed arrows, then along the dotted arrows.

only contains the direct neighboring cells, with the only difference that one cell here means the whole spatial subdomain of one time slice. They are as well exchanged by point to point messages.

The exchange method of the `MultiHarmonicField` is quite different to the previous discussed ones. All coefficient have to be accessed for the computation of one block element of the range vector of the multi-harmonic convection-diffusion operator $\mathcal{F}$. The coefficients are communicated by an `MPI_Allgatherv` on the subcommunicator that contains all the ranks of the same spatial subdomain, cf. Figure 4.7.

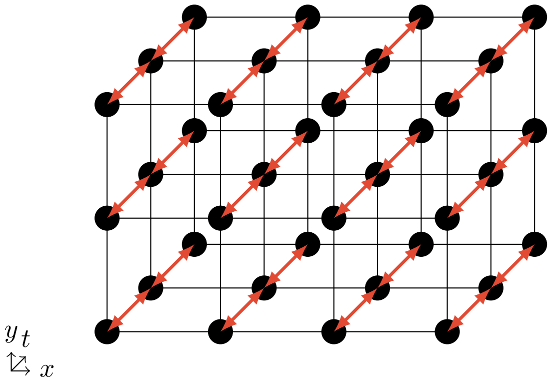


Figure 4.7: Communication for nonlinear terms along time dimension `MPI_Allgather`.

As a small final remark in this section we want to mention that each implemented field class can be written to a file and also be restored from file using HDF5[3]. HDF5 is a hierarchical data format and is supported on most computer cluster. It allows to write

---

[3] `hdfgroup.org`

from many processors to one file in an efficient and high performance way. This allows to restart an application in the case of an occurring failure. It is also used to analyze and visualize the solution.

5

Experiments

In this chapter we show the performance and applicability of the *spectral in time* solver to different time-periodic flow problems. The first problem is a steady streaming between two plates with a time-periodic in and out flow. This flow problem is used to investigate the performance of the different preconditioners. The second flow problem is a time-periodic Taylor–Green vortex. It is used to validate the correctness of the spectral in time method and to investigate the parallel performance in comparison to a time-stepping method. The third flow problem is a channel flow with an oscillating obstacle. It is used to show the performance of the spectral refinement algorithm and the scaling of the time parallelization. The fourth problem is a swept Hiemenz flow. This flow problem is used to investigate the applicability of the spectral in time method to a highly complex three-dimensional flow. The structure of the flow allows to investigate the performance of the spectral in time method on laminar, transitional and turbulent flows.

5.1 Rayleigh streaming

In this section a Rayleigh streaming problem is computed to validate our spectral in time method. Rayleigh [75] observed in 1884 that a steady flow between to two plates can be induced through acoustic waves. This has lead to many studies and experiments under the name acoustic streaming [62, 69, 88]. But also incompressible flows can show a time-averaged motion induced by oscillating components, such as oscillating boundaries, oscillating in and out-flows, or oscillating body forces. If a streaming is occurring in an incompressible flow it is called steady streaming [76].

In this section we are focusing on the performance of the preconditioner for the *spectral in time* solver. The preconditioners are described in section 3.1.4. The performance of the preconditioners is examined with regard to varying Reynolds number, Strouhal number, and spatial discretization. The preconditioners will be investigated with regard to the temporal discretization in detail in section 5.3. Parts of this section have been published in [51].

5.1.1 Problem

We investigate a two-dimensional flow between two plates with time-periodic in and outflow. This flow is a solution of the two-dimensional Navier–Stokes equations

$$\begin{aligned} \alpha^2 \partial_t \mathbf{u} + \mathrm{Re}(\mathbf{u} \cdot \boldsymbol{\nabla})\mathbf{u} - \boldsymbol{\Delta}\mathbf{u} + \mathrm{Re}\boldsymbol{\nabla} p = \mathbf{0}, \\ \boldsymbol{\nabla} \cdot \mathbf{u} = 0, \end{aligned} \tag{5.1}$$

for $\mathbf{x} \in \Omega = [0,\ 2] \times [0,\ 2]$ and $t \in [0,\ 2\pi]$. The reference length scale L_{ref} is set to be half the channel width. The reference velocity U_{ref} is the maximum inflow velocity. The parameters Re, St, and the spatial system size $N_{\mathbf{x}} = N_x \times N_y$ are varied. The number of frequencies is fixed $N_f = 14$ throughout. Periodic boundary conditions are used in x-direction, whereas Dirichlet boundary conditions are used in y-direction. The prescribed velocity profile at the boundaries $y=0$ and $y=2$ is

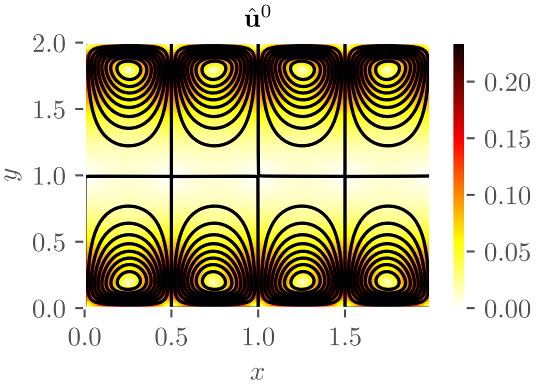


Figure 5.1: Streamlines of the velocity field of the zero coefficient at Re = 100 and St = 1. The zero coefficient is the constant coefficient from the Fourier ansatz (2.10), which corresponds to the time-averaged velocity field. Colors indicate the velocity magnitude.

$$\mathbf{u}_{\mathrm{bc}}(x,t) = \begin{bmatrix} 0 \\ \sin(t)\sin(\pi x) \end{bmatrix}.$$

Although the time-averaged inflow and outflow is zero, one can still observe a time-averaged flow. In our method, this streaming field corresponds to the zero coefficient $\hat{\mathbf{u}}^0$. This coefficient is illustrated in Figure 5.1 for $\mathrm{Re}=100$ and $\mathrm{St}=1$. The first and second mode is illustrated in Figures 5.2 and 5.3, respectively. The time snapshot for $t=0$ and $t=\pi/2$ can be seen in Figure 5.4. One can observe the different behavior of fluid in the area of suction ($x>1$, $y<1$) and ejection ($x<1$, $y<1$).

We mainly investigate the performance of the linear solver. Therefore a fixed number of

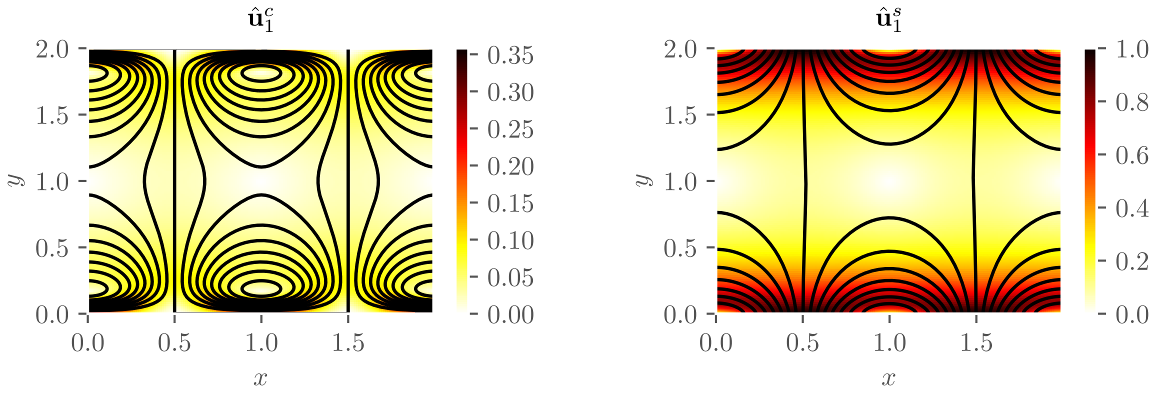


Figure 5.2: Streamlines of the velocity field of the first mode at Re = 100 and St = 1. Colors indicate the velocity magnitude.

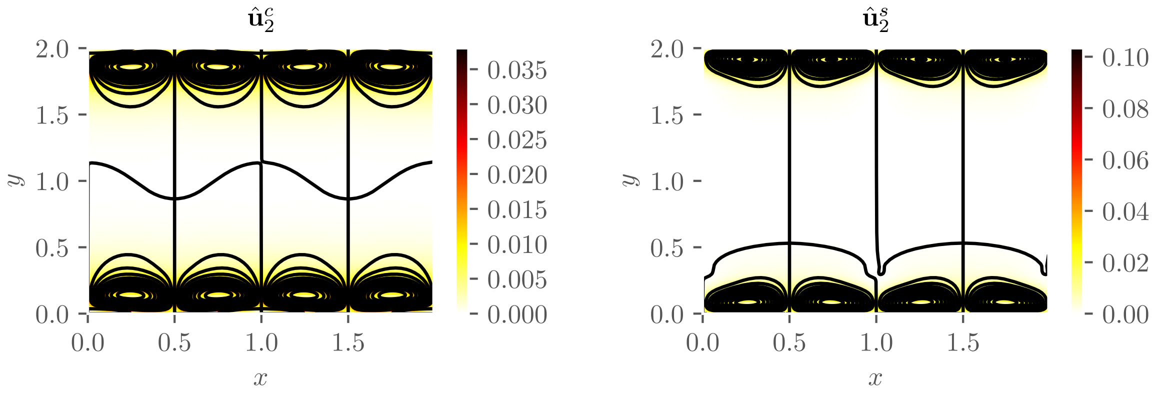


Figure 5.3: Streamlines of the velocity field of the second mode at Re = 100 and St = 1. Colors indicate the velocity magnitude.

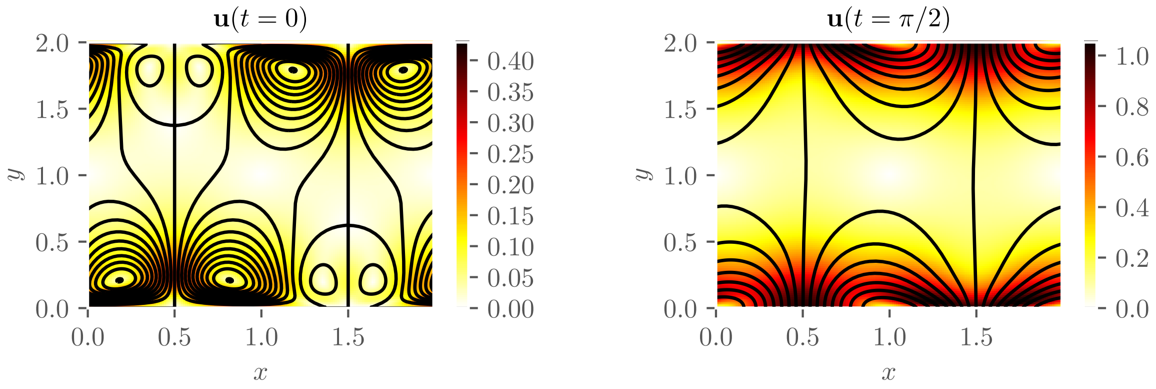


Figure 5.4: Streamlines of the velocity field at Re = 100 and St = 1. Colors indicate the velocity magnitude.

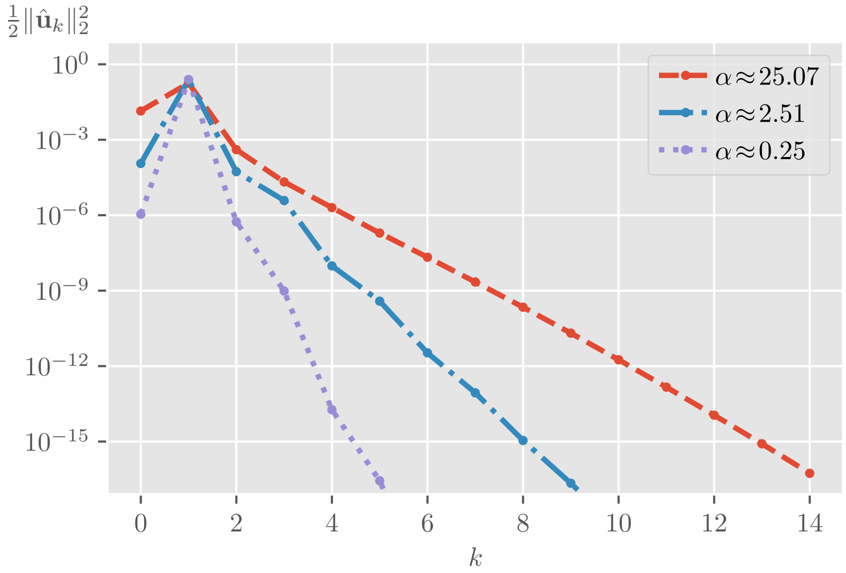


Figure 5.5: Energy spectrum for varying α.

frequencies is used in Algorithm 1 with the update stopping criterion (3.16) with a tolerance $\mathrm{tol}_{\delta\mathbf{q}} = 10^{-6}$. The linear Picard problem (3.10) is solved with flexible GMRES up to a tolerance of 10^{-2} with the 2-by-2 block-triangular preconditioner (3.28), where the (1,1) block is a multi-harmonic convection-diffusion operator (3.37) and the (2,2) block is a least squares commutator based approximation of the Schur complement (3.35). The inner subproblems are also solved with flexible GMRES up to a tolerance of 10^{-6} which gives results almost as accurate as with a direct solver. So we can neglect influences of solving the problems only inexactly. We note that this is not the most efficient strategy. For the stationary convection-diffusion problems we use as preconditioner two multigrid V-cycles. As smoother we use two Jacobi iteration steps with a damping factor of one half. As coarse grid solver we use a four-direction Gauss–Seidel iteration. For explanations and references we refer to section 3.1.4.4. For the Poisson problem we use Jacobi iteration as smoother and coarse grid solver with a damping factor of 6/7 with eight smoothing sweeps and one V-cycle.

5.1.2 Performance

In this section the Womersley numbers $\alpha\approx 0.25$, $\alpha\approx 2.51$, and $\alpha\approx 25.1$ refer to $\mathrm{Re}=1$ and $\mathrm{St}=0.01$, $\mathrm{Re}=10$ and $\mathrm{St}=0.1$, $\mathrm{Re}=100$ and $\mathrm{St}=1$, respectively. In Figure 5.5, we can see the half of the squared 2-norm of each velocity coefficient, which corresponds to the energy as the norm of a velocity coefficient is the square root of the doubled energy. We observe that for higher Womersley numbers α the energy is decaying much slower, and that the energy transfer to the zero mode is also higher. This also relates to the spectral in time discretization error. The discretization error is the truncation error as higher modes for $k > N_f$ are neglected.

In Figure 5.6, we can see the necessary Picard steps to satisfy the update stopping criterion (3.16) for varying Re and St. It can be observed that for small Reynolds numbers only a few steps are needed. But for larger Reynolds and lower Strouhal numbers more Picard steps are needed.

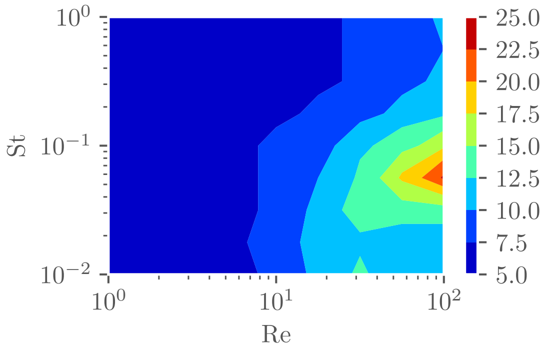


Figure 5.6: Number of Picard steps for varying Re and St.

In Figure 5.7–5.11, we can see the number of iteration steps necessary to solve the different problems (3.25), (3.37), (3.40), (3.51), and (3.52) for different Re, St and a system size $N_\mathbf{x} = 128^2$. We considered the average number of one linear iteration solve. For the different system sizes we also indicate the standard deviation.

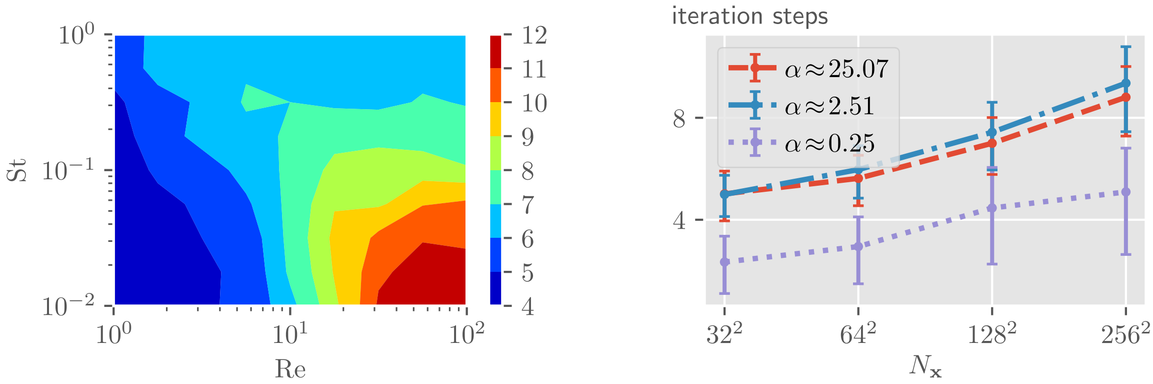


Figure 5.7: Average number of iteration steps for the *Picard problem*. On the left side for varying Re and St. On the right side for different system sizes.

All problems show optimal behavior for the different system sizes. Only the Picard problem shows a slight increase of the iteration numbers with growing system size. The solvers for the stationary convection-diffusion problem (Figure 5.10) and the Poisson problem (Figure 5.11) show also the same number of iterations steps for all the Reynolds and Womersley numbers.

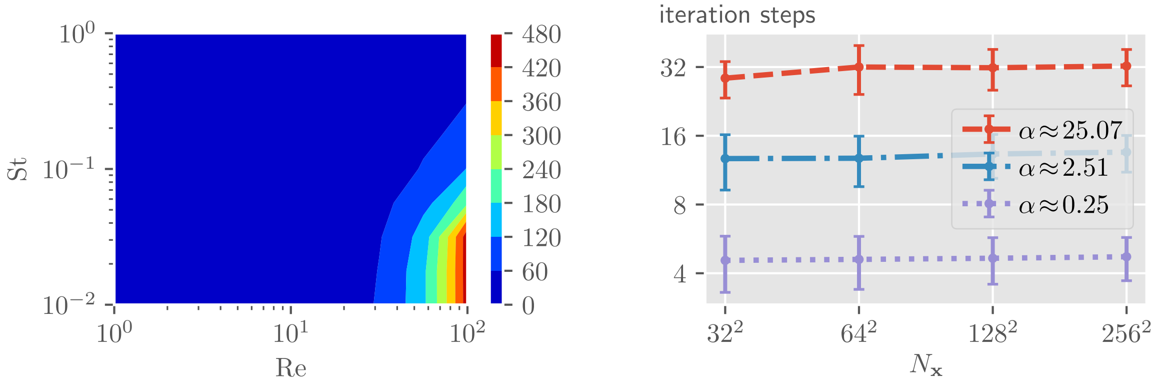


Figure 5.8: Average number of iteration steps for the *multi-harmonic convection-diffusion problem*. On the left side for varying Re and St. On the right side for different system sizes.

The preconditioner for the Picard problems leads to small numbers of iteration steps. The number of iterations steps are slightly higher for small Strouhal numbers and low Reynolds numbers. But they are still acceptable. The solver for the multi-harmonic convection-diffusion problem on the other hand is suffering under a huge increase of the number of linear iteration steps. The maximum of 500 iteration steps is even reached in some cases, see Figure 5.8. This is due to the non block diagonal dominant property in these cases. The solver for the harmonic convection-diffusion problem shows a good behavior for a broad range of Reynolds and Strouhal numbers. For small Reynolds and Strouhal numbers it is even better.

The standard deviation of the problems of stationary, harmonic convection-diffusion and Poisson problems is small, which is a very desired result. As these solves are done in parallel and a high standard deviation would lead otherwise to load balancing problems.

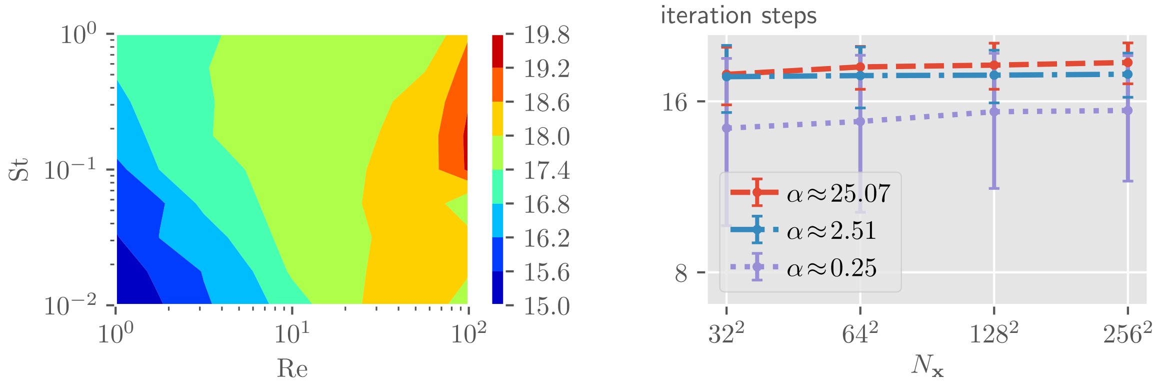


Figure 5.9: Average number of iteration steps for the *harmonic convection-diffusion problem.* On the left side for varying Re and St. On the right side for different system sizes.

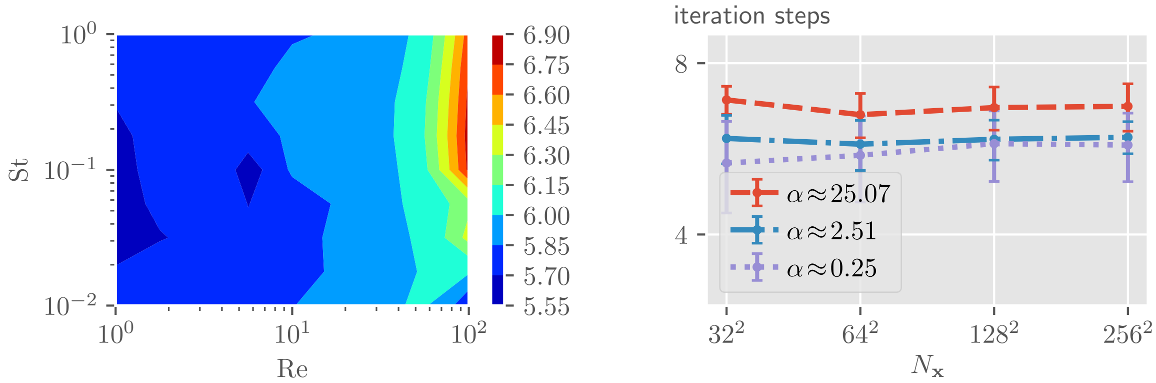


Figure 5.10: Average number of iteration steps for the *convection-diffusion problem.* On the left side for varying Re and St. On the right side for different system sizes.

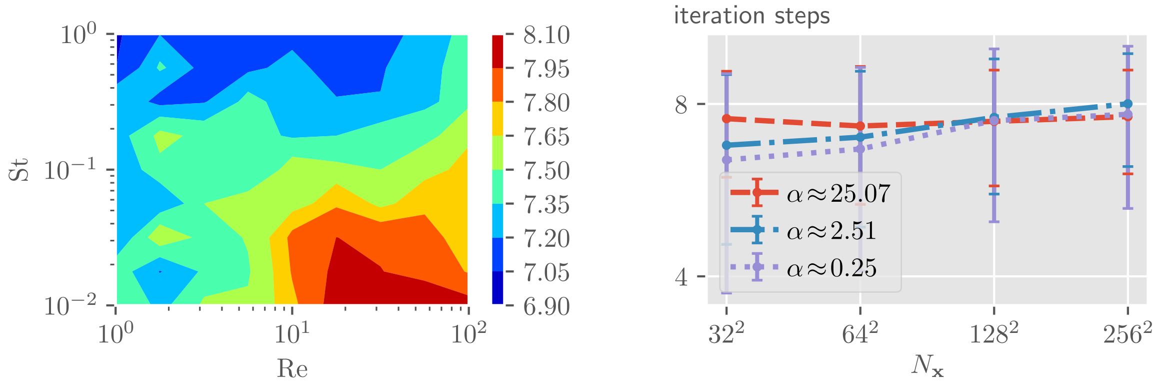


Figure 5.11: Average number of iteration steps for the *Poisson problem.* On the left side for varying Re and St. On the right side for different system sizes.

These results show that the proposed preconditioners are close to optimal for a broad range of parameters. Except for certain cases where the block diagonal approximation of the multi-harmonic convection-diffusion operator is losing quality. Optimal preconditioners are a necessity to solve larger problems and more complex flows. We study larger problems and more complex flows in the next sections.

5.2 Time-periodic Taylor–Green vortex

In this section we compare the performance of a classical time-stepping method and the *spectral in time* method presented in section 3.1. For this comparison a time-periodic Taylor–Green vortex is computed. Taylor–Green vortices have been studied to get a deeper understanding of the production of smaller eddies from larger eddies [89]. Furthermore Taylor–Green vortices are studied to validate and benchmark the time-integration of fluid solvers [93]. We compare the scaling behavior and the time to solution of the two mentioned methods. These findings have been published in [3].

5.2.1 Problem

We consider of the two-dimensional Navier–Stokes equations

$$\begin{aligned} \alpha^2 \partial_t \mathbf{u} + \mathrm{Re}(\mathbf{u} \cdot \boldsymbol{\nabla})\mathbf{u} - \boldsymbol{\Delta}\mathbf{u} + \mathrm{Re}\boldsymbol{\nabla} p &= \mathrm{Re}\,\mathbf{f}, \\ \boldsymbol{\nabla} \cdot \mathbf{u} &= 0, \end{aligned} \qquad \mathbf{x} \in \Omega,\ t > 0, \tag{5.2}$$

on a spatial domain $\Omega = [0,\ 2\pi/a] \times [0,\ 2\pi/b]$, with periodic boundary conditions. We consider the time-periodic force density

$$\mathbf{f}(t) = \begin{bmatrix} \alpha^2 A \cos(ax)\sin(by)\cos(t) + A(a^2+b^2)\cos(ax)\sin(by)(1+\sin(t)) \\ \alpha^2 B \sin(ax)\cos(by)\cos(t) + B(a^2+b^2)\sin(ax)\cos(by)(1+\sin(t)) \end{bmatrix},$$

The incompressibility condition implies that the parameters have to satisfy $aA + bB = 0$. This periodic forcing leads to the time-periodic steady state

$$\mathbf{u}_{\mathrm{sol}}(\mathbf{x}, t) = \begin{bmatrix} A\cos(ax)\sin(by)(1+\sin(t)) \\ B\sin(ax)\cos(by)(1+\sin(t)) \end{bmatrix}, \tag{5.3}$$

with

$$p_{\mathrm{sol}}(\mathbf{x}, t) = -\mathrm{Re}\frac{A^2\cos(2ax) + B^2\cos(2by)}{4}(1+\sin(t))^2. \tag{5.4}$$

This can be seen as a Taylor–Green vortex, that periodically changes its magnitude. The velocity field and pressure field at $t = \pi/4$ with the parameters $a = b = 1$, $A = 0.5$, $B = -0.5$, and $\mathrm{Re} = 10$ are visualized in Figure 5.12. These are also the parameters used throughout this section.

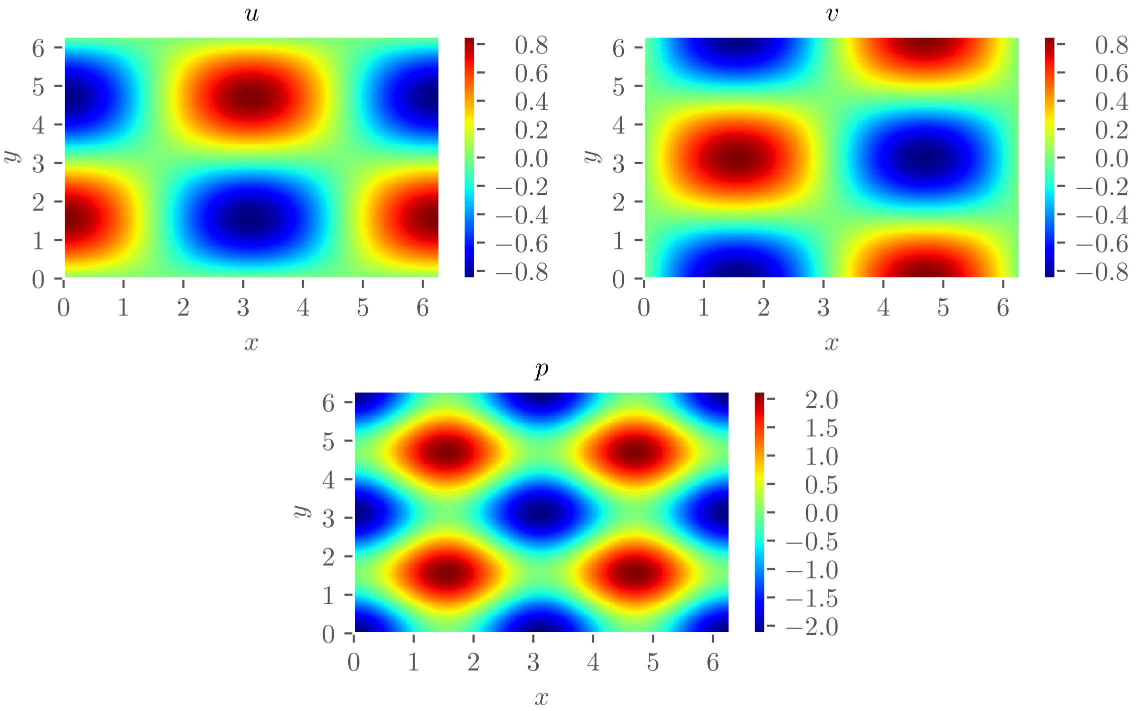


Figure 5.12: Velocity/pressure fields, for $a{=}b{=}1$, $A{=}{-}B{=}0.5$, $\mathrm{Re}{=}10$ at $t = \pi/4$.

The first method we use is a classical time-stepping method. It employs a mixed Crank–Nicolson Runge–Kutta time-integration, see [45]. The time-periodic structure of the problem is not exploited. We start with initial conditions $\mathbf{u}(\mathbf{x}, 0) = \mathbf{0}$ and $p(\mathbf{x}, 0) = 0$. The time-integration is stopped if the relative difference in the solution after one period of time is smaller than a prescribed tolerance. The time-stepping method is not parallelized in time.

The second method we use is the spectral in time method, described in Chapters 2 and 3. This method exploits the time-periodic structure of the problem. As the needed number of frequencies is known before hand we use fixed number of frequencies Algorithm 1, with $N_f = 2$. The Picard iteration is started with a zero initial guess. The linear Picard problem (3.10) is solved with flexible GMRES up to a tolerance of 10^{-2} with the 2-by-2 block-triangular preconditioner (3.28), where the (1,1) block is a multi-harmonic convection-diffusion matrix (3.37) and the (2,2) block is a least squares commutator based approximation of the Schur complement (3.35). The inner subproblems are also solved with flexible GMRES but up to a tolerance 10^{-1}. For the stationary convection-diffusion problems we use as preconditioner one multigrid V-cycle. As smoother we use two Jacobi iteration steps with a damping factor of one half. As coarse grid solver we use a four-direction Gauss–Seidel iteration. For explanations and references we refer to section 3.1.4.4. For the Poisson problem we use as preconditioner one multigrid V-cycle. We use Jacobi iteration as smoother and coarse grid solver with a damping factor of 6/7 with four smoothing sweeps.

To compare the two different approaches, we consider the relative error of the approximated with the analytical steady-state solution (5.3)

$$e = \frac{\|\mathbf{u}_{\text{sol}} - \mathbf{u}\|_2}{\|\mathbf{u}_{\text{sol}}\|_2}.$$

Here, $\|\cdot\|_2$ is the grid error function norm [61]. Note that the norm is taken over the whole spatial domain as well as over the whole period of time. For the spectral method this amounts to the square root of the sum of the squared norms of the Fourier coefficients.

The measured error is compounded of three different components, the spatial discretization error, the temporal discretization error, and the difference of the approximate to the steady-state solution. The latter we call the methodical error. If we want to estimate just one error component, we have to make sure that the other two are much smaller.

Since the 6th order finite difference stencil on the fine 128×128 grid implies a spatial discretization error of about the order of 10^{-9} we only consider the temporal and methodical errors in sections 5.2.2 and 5.2.3, respectively. The strong scaling performance of both methods is investigated in section 5.2.4.

5.2.2 Convergence of the time discretization

First we consider the temporal discretization error. The problem is solved for different numbers of time steps per period. To make sure we just measure the temporal discretization error we stop each computation when there is no change in the approximate solution anymore, so that the methodical error is smaller than the temporal discretization error. The spectral approach has no discretization error if $N_f \geq 2$ for this particular problem. For other problems one has to consider also the truncation error, which can be controlled through spectral refinement Algorithm 2 or Algorithm 3.

Note that for small Womersley numbers at least 128 time steps per period have to be used to satisfy the CFL condition. The rate of convergence is independent of the Womersley number in both cases. In Figure 5.13, we can see that for higher Womersley

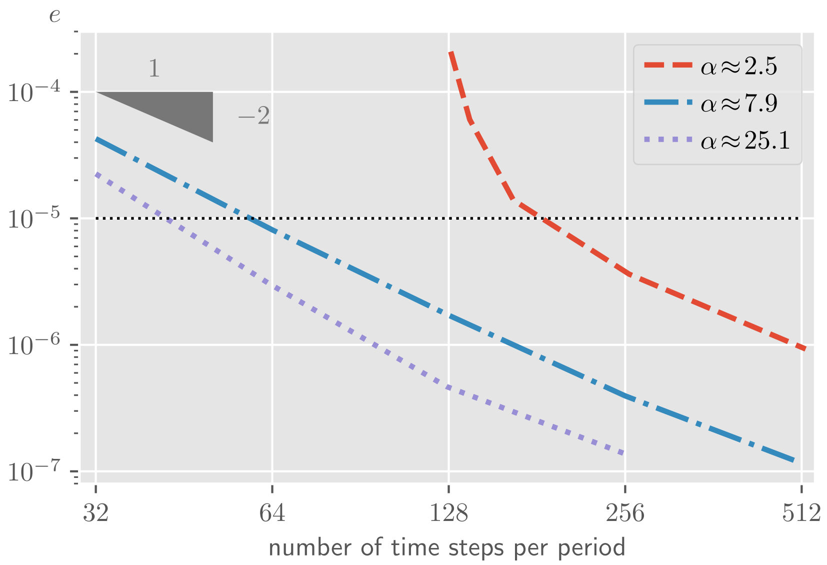


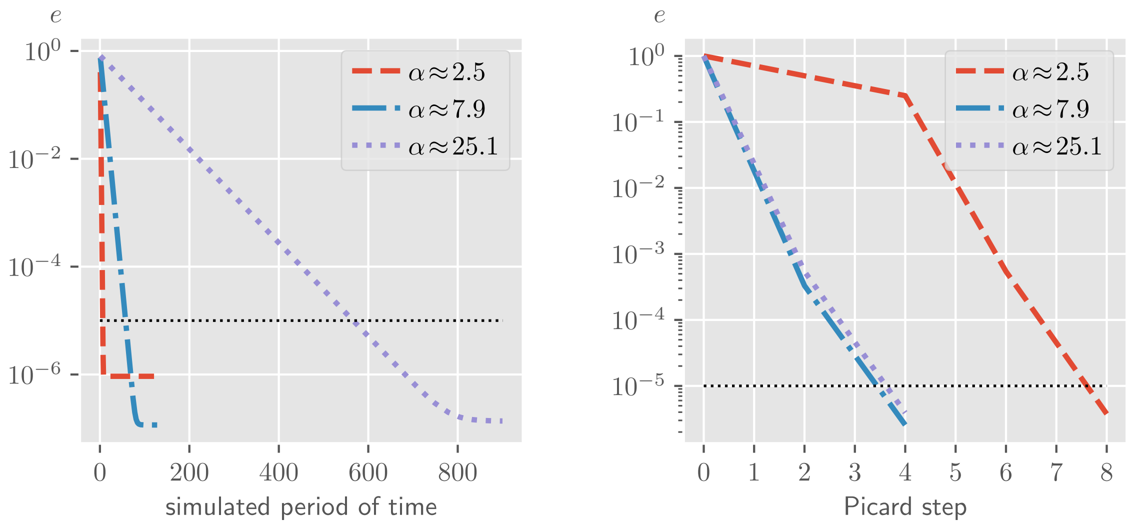
Figure 5.13: Convergence of the temporal discretization error over the number of time steps per period.

numbers the discretization error is smaller and that the expected convergence order of two is obtained.

5.2.3 Convergence of the method

In this section we investigate the methodical error (the difference between the approximated solution and the steady-state solution). In the time-stepping method we fix the time step and use 128 time steps per period. Note that the decay in the error over the number of periods of time does not depend on the spatial and temporal discretization, because it is a physical property of the problem. Hence it depends only on the Womersley number. In Figure 5.14a, we see that the transition phase (the time until the steady state is reached) is much shorter for smaller Womersley numbers than for larger ones. In the latter case it takes hundreds of time periods to get to the steady state.

The decay in the error after each nonlinear solver step depends on the solver, here the Picard method, and the stopping criterion of the inner iterative solver. Here, we set a tolerance of 10^{-2} for the scaled residual. We always set $N_f = 2$. In Figure 5.14b, we see that only a few Picard steps are needed to obtain a solution with an error smaller than 10^{-5}. For Womersley numbers $\alpha = 7.9$ and $\alpha = 25.1$ the spectral method converges in four steps. For $\alpha = 2.5$ in the first four Picard steps, the backtracking algorithm has to cut the step width in half to enforce a decrease in the residual norm. The decay in the error is small in these steps.



(a) Time-stepping method.

(b) Spectral method.

Figure 5.14: Methodical error over method iteration step. In case of the time-stepping one method method iteration step is simulating one period of time, in case of the spectral method one method method iteration step is one Picard step.

5.2.4 Scaling

All computations and timings were done on the Euler cluster[1] at ETH Zürich. We employ Euler III with quad-core Intel Xeon E3-1285Lv5 processors at 3.0–3.7 GHz connected by a 10Gb/s–40Gb/s Ethernet network and Euler II with dual 12-core Intel Xeon E5-2680v3 at 2.7–3.3 GHz connected by a 56 Gb/s InfiniBand FDR network. Both are equipped with DDR4 memory clocked at 2133 MHz. Euler III has 32 GB memory per node, whereas Euler II has varying memory from 64 GB up to 512 GB per node. The GCC 4.8.2 compiler is used in combination with OpenMPI 1.6.5. For the time-stepping method, the computation time of one time period is presented. For the spectral method, the execution time of the last Picard step is given. This is essentially the time for the solution of the linear system with the highest condition number [25]. To get rid of overhead from the runtime environment and the operating system, the minimal computation time of ten runs is presented. The time-discretization is fixed. For the time-stepping

---

[1] `https://scicomp.ethz.ch/wiki/Euler`

method the numbers of time steps per period are 184, 58, and 42 for the Womersley numbers $\alpha = 2.5$, 7.9, and 25.1, respectively. By this choices of parameters the CFL condition is satisfied and the temporal discretization error is approximately equal (cf. Figure 5.13). For the spectral method, $N_f = 2$ is used which means that the problem can be divided to three subgroups of processing units. Spatial grids of size 64×64, 128×128, 256×256, and 512×512 are considered. The results can be seen for Euler III in Figure 5.15 and for for Euler II in Figure 5.16.

From the left column of Figure 5.15 we see that for the time-stepping method we only get reasonable speedups for up to four cores, in particular for small system sizes. This is due to the relatively small system sizes and the slow network connection. The execution time is increasing with the Womersley number.

In the right column of Figure 5.15, we see that the spectral method can obtain decent speedups up to 48 cores despite the small system sizes and the slow network connection. The execution time is decreasing with the Womersley number.

In comparison the scaling behavior overall is better on Euler II, due to slower processing units, a faster network and more cores per node, cf. Figure 5.15 and Figure 5.16.

5.2.5 Time to solution

Both methods are very different in terms of computing the time-periodic steady state of the Navier–Stokes equations, so that they can only be compared by their times to solution. To this end we compare the computing time needed to reduce the error below 10^{-5}. We compute the time to solution by multiplying the necessary periods of time from section 5.2.3 with the best wall-clock time to compute one period of time, or by multiplying the necessary number of Picard steps with the wall-clock time to compute one Picard step (cf. Figure 5.15). We consider only the spatial discretization with 128^2 grid points. In Table 5.1, we can see that the time to compute one period of time is decreasing with the Womersley number α, as fewer time steps have to be executed. But the overall time to solution is increasing as the transition phase extends even faster. Furthermore, we see that for the spectral method the time to compute one Picard step is decreasing with the Womersley number, as the preconditioner has higher quality for higher Womersley numbers. This will be also observed in section 5.3.

In summary we observe that for the Womersley number $\alpha = 2.5$ the time-stepping method and the spectral in time method take roughly the same time. But for $\alpha = 7.9$

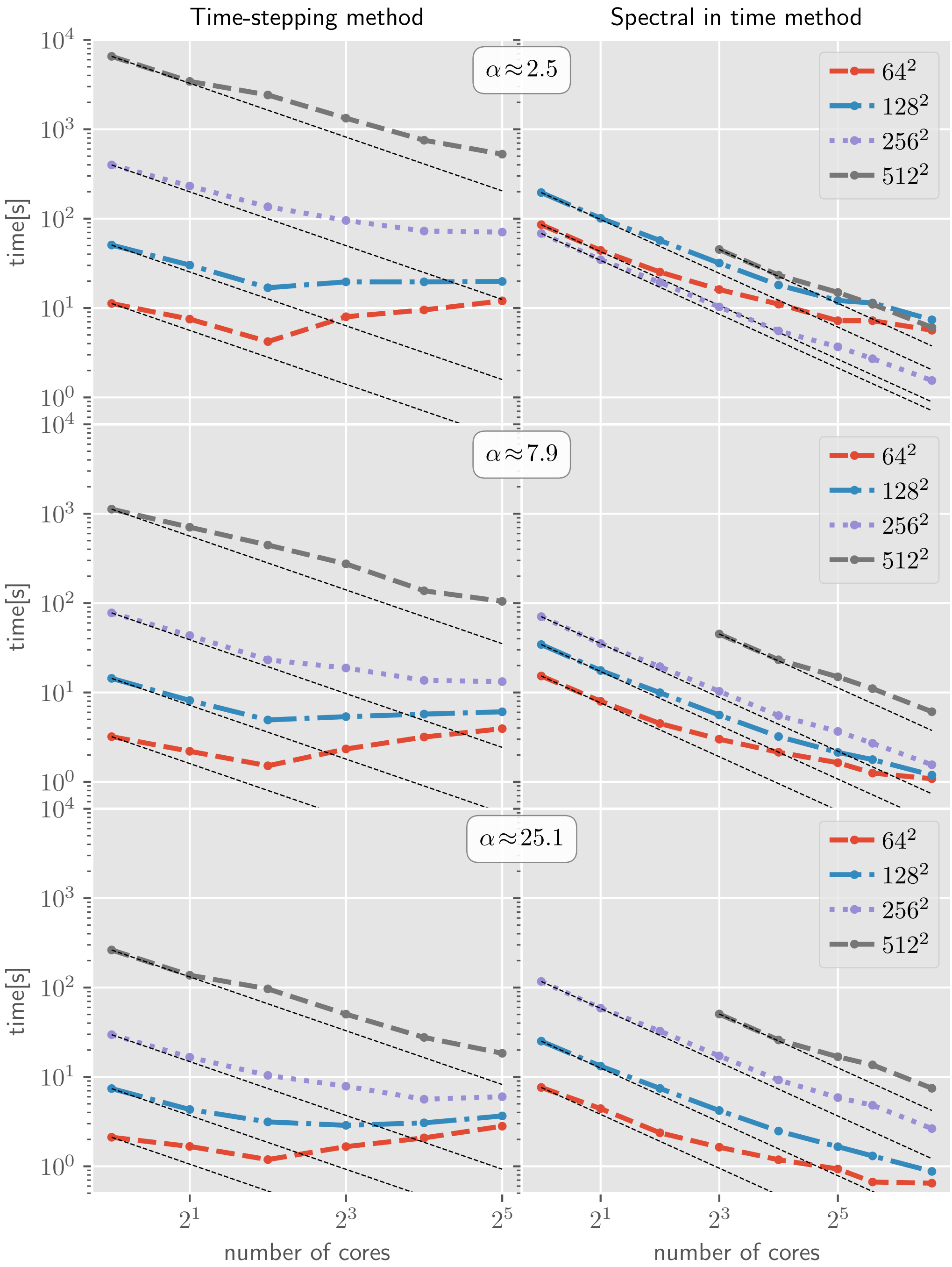


Figure 5.15: Execution times for the time-stepping method (left column) and the spectral method (right column), for different Womersley numbers $\alpha = 2.5$, 7.9, and 25.1 (first, second, and third row, respectively), for systems sized 64×64, 128×128, 256×256, and 512×512 (with dashed, dashed-dotted, dotted, and solid line, respectively) on Euler III. The ideal speedups are indicated by thin dashed black lines. 512×512 starts at 8 cores for memory reasons.

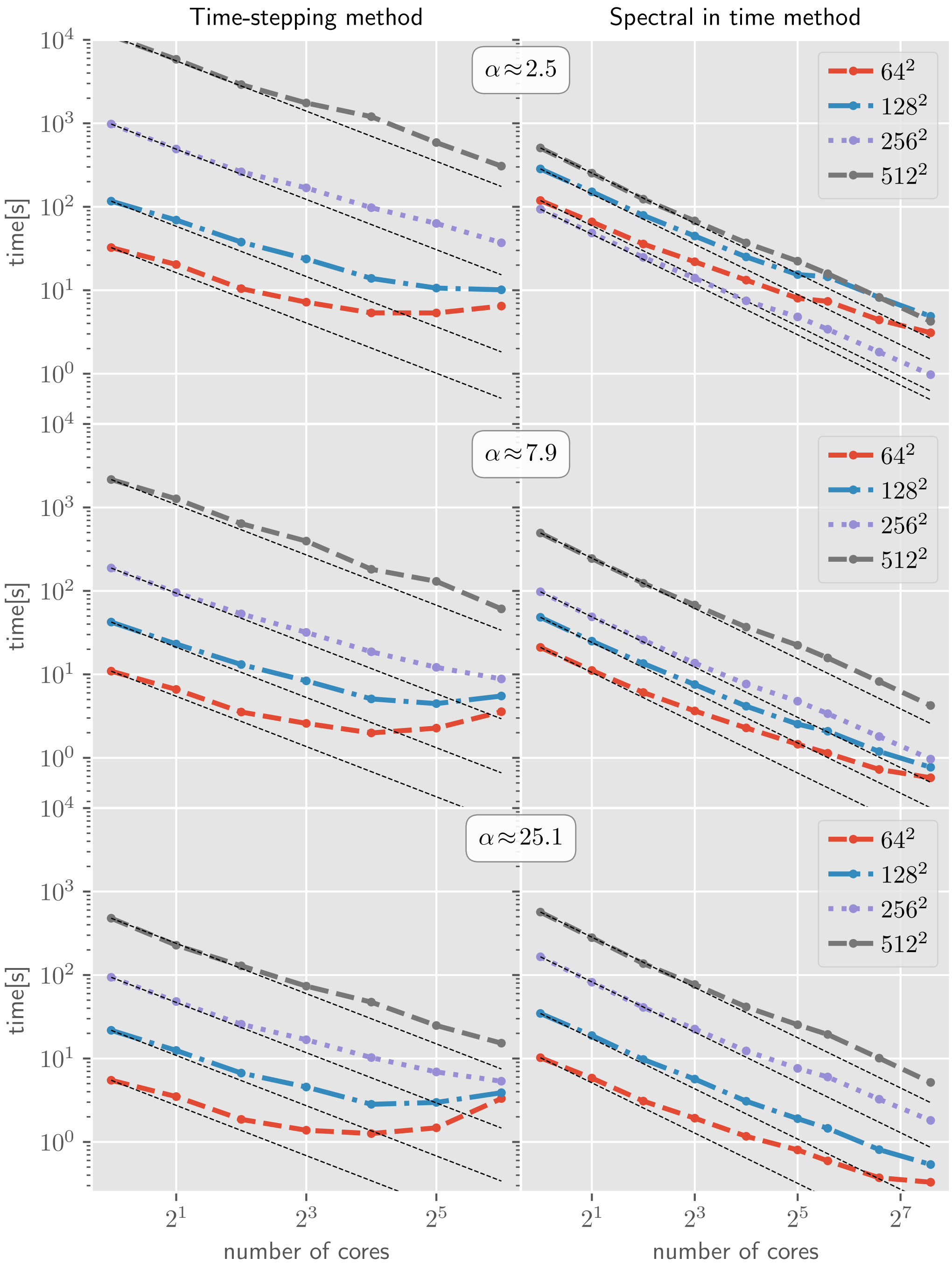


Figure 5.16: Execution times for the time-stepping method (left column) and the spectral method (right column), for different Womersley numbers $\alpha = 2.5$, 7.9, and 25.1 (first, second, and third row, respectively), for systems sized 64×64, 128×128, 256×256, and 512×512 (with dashed, dashed-dotted, dotted, and solid line, respectively) on Euler II. The ideal speedups are indicated by thin dashed black lines.

Table 5.1: Time to solution in seconds.

		Time-stepping		Spectral in time	
α		Euler III	Euler II	Euler III	Euler II
2.5	number of periods/Picard step	6		8	
	time to compute one period/step	16.87	10.13	7.376	4.877
	time to solution	**101.2**	**60.75**	**59.01**	**39.02**
7.9	number of periods/Picard step	57		4	
	time to compute one period/step	4.928	4.464	1.189	0.771
	time to solution	**280.9**	**254.4**	**4.756**	**3.085**
25.1	number of periods/Picard step	560		4	
	time to compute one period/step	2.874	2.827	0.8771	0.5368
	time to solution	**1609.**	**1583.**	**3.508**	**2.147**

the spectral method is 60 to 80 times faster, and for $\alpha = 25.1$ it is even 400 to 700 times faster than the time stepping method. This is due to the extended transition phase that has to be traversed by the time-stepping method, and the limited degree of parallelism by just using spatial domain decomposition.

5.2.6 Conclusions

We have compared two methods for computing time-periodic steady states of the Navier–Stokes equations. For the relatively small system sizes that we considered the time-stepping method already suffers from the limited degree of parallelism that is offered by the partitioning of only the spatial dimensions. We note that this saturation of the parallelism also occurs for larger problems, but then for higher number of cores. Additionally, the transition phase extends with the Womersley number which increases the number of time steps needed to reach the time-periodic state.

In contrast, the spectral in time method shows a good scaling behavior, even for small problem sizes, thanks to the (implicit) parallelization in time. Also there are just a few Picard steps necessary to compute a good approximation of the solution. So, we can conclude that if the Womersley number is sufficient large and there are enough cores available, then the spectral in time method can drastically outperform a time-stepping approach. If the Womersley number is low and the number of available cores is limited, it is faster to traverse the relatively short transition phase by a time-stepping method.

These findings indicate that the use of the spectral method will also be beneficial for larger and more complex problems, where no analytical steady state is known. More complex problems entail the need to use more Fourier coefficient, which will also entail more parallelism. The number of Picard's iterations is expected to grow, but so will the transition time for the time-stepping method.

5.3 Channel flow with oscillating obstacle

In this section a channel flow with an oscillating spherical obstacle is computed by the *spectral in time* method, cf. section 3.1. The spectral refinement is evaluated as well as the scaling behavior of the time parallelization. Parts of this section have been published in [50].

5.3.1 Problem

To test the proposed *spectral in time* method, we consider a three-dimensional plane channel flow in the spatial domain $\Omega = [0,8] \times [0,2] \times [0,4]$. This flow is a solution of the three-dimensional Navier–Stokes equations

$$\begin{aligned} \alpha^2 \partial_t \mathbf{u} + \mathrm{Re}(\mathbf{u} \cdot \boldsymbol{\nabla})\mathbf{u} - \boldsymbol{\Delta}\mathbf{u} + \mathrm{Re}\boldsymbol{\nabla} p &= \mathrm{Re}\,\mathbf{f}, \\ \boldsymbol{\nabla} \cdot \mathbf{u} &= 0, \end{aligned} \quad \mathbf{x} \in \Omega,\ t \in [0, 2\pi]. \tag{5.5}$$

Unless mentioned otherwise, a spatial grid of $257 \times 65 \times 129$ grid points is used. We set the tolerances $\mathrm{tol}_{\mathbf{r}} = 10^{-6}$ and $\mathrm{tol}_f = 10^{-4}$. The reference length scale L_{ref} is set to be half the channel width. We enforce a Poiseuille flow at $x = 0$ and $x = 8$ as Dirichlet boundary conditions and use the centerline velocity as reference velocity U_{ref}. The channel flow is disturbed by a periodic pulsating force density

$$\mathbf{f}(\mathbf{x}, t) = \begin{bmatrix} -2 \\ \cos(t) \\ 0 \end{bmatrix} \exp\left(-\frac{(x-1)^2 + (y-1)^2 + (z-2)^2}{0.2^2} \right). \tag{5.6}$$

A Poiseuille flow is used as the initial guess for the velocity field. The schematic set-up can be seen in Figure 5.17.

The linear Picard problem (3.10) and the inner problems are solved with flexible GMRES up to a tolerance of 10^{-1}. The 2-by-2 block-triangular preconditioner (3.28) is used, where the (1,1) block is a multi-harmonic convection-diffusion problem and the (2,2) block is an approximation of the Schur complement. For the stationary convection-diffusion problems we use as preconditioner one multigrid V-cycle. As smoother we use four Jacobi iterations steps with a damping factor of one half. As coarse grid solver we use GMRES. For the Poisson problem we use as preconditioner one multigrid V-cycle

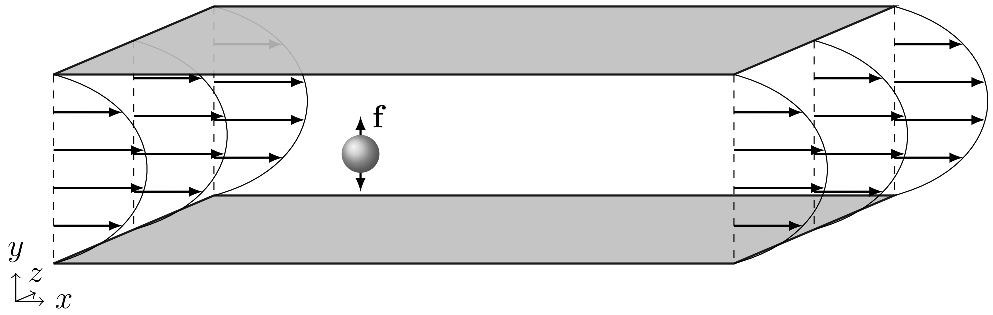


Figure 5.17: Schematic set-up of channel with oscillating obstacle force.

with a Jacobi iteration as smoother with a damping factor of 6/7 with four smoothing sweeps. As coarse grid solver we use a direct solver.

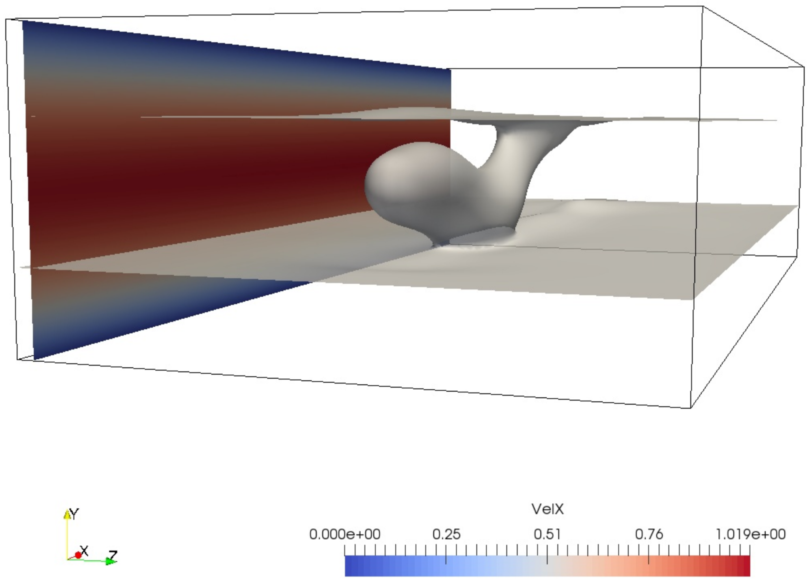


Figure 5.18: Isosurfaces of velocity magnitude $\|\mathbf{u}(\mathbf{x}, t = 0)\| = 0.8$.

To visualize a typical result, we simulate this flow for $\mathrm{Re} = 200$ and Strouhal number 0.2 which corresponds to $\alpha \approx 16$. The stopping criterion tol_f is matched for $N_f = 7$. A time snapshot of the isosurface for $t = 0$ can be seen in Figure 5.18. Isosurfaces of the first few resulting Fourier modes can be seen in Figures 5.19 and 5.20. We find that the disturbance is spread over the whole domain and the higher modes are excited as well. Furthermore, the periodic forcing modifies the mean flow shown in Figure 5.19.

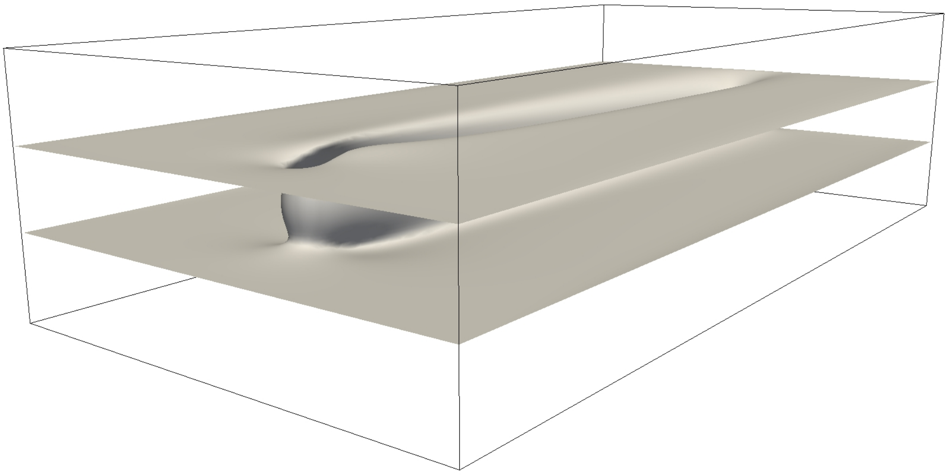
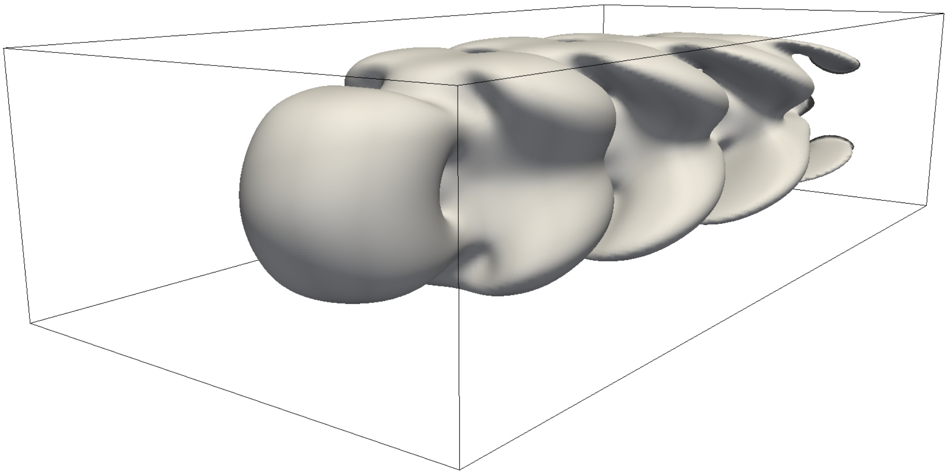


Figure 5.19: Isosurfaces of the velocity magnitude of the zero Fourier mode $\|\hat{\mathbf{u}}^0(\mathbf{x})\|_2 = 0.9$.

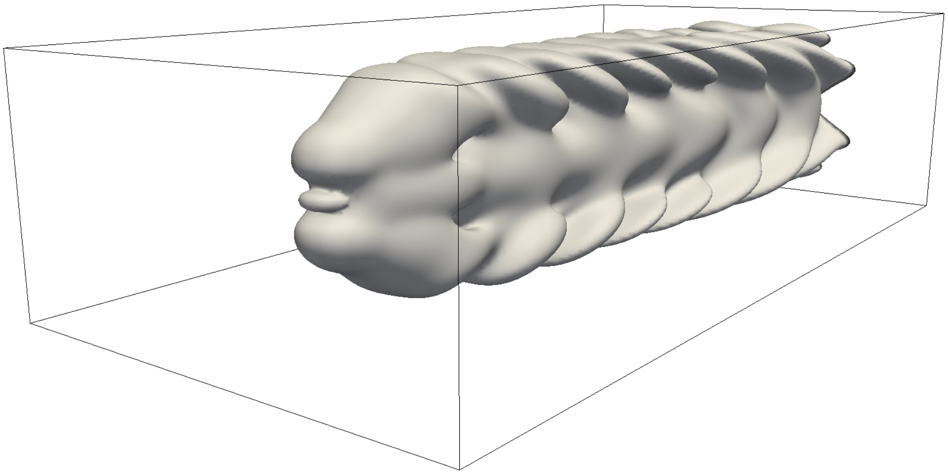
5.3.2 Performance

First we assess the performance of the proposed nonlinear solver. We investigate how residuals decrease when using *spectral refinement* with $N_f^{\text{start}} = 1 \leq N_f \leq N_f^{\text{end}} = 7$ and a fixed number $N_f = 7$ of Fourier modes, respectively. We assume that the computational cost of one nonlinear iteration step is proportional to N_f times the linear iteration steps for solving (3.25). For both methods we plot the residual against the computational costs cf. Figure 5.21. We find that although the residual is increased by the *spectral refinement*, indicated by kinks, the overall convergence is much faster.

In Table 5.2, we give the numbers of nonlinear iteration steps necessary for getting a converged solution. The Reynolds number ranges from 10 and 200. The Strouhal number St varies from 0.05 to 0.4 which corresponds to a Womersley number ranging from 1.7 to 22.4. We observe that for higher Strouhal numbers and higher Reynolds numbers the number of nonlinear solver steps increases. This is to be expected as nonlinear effects become stronger and the solution gets more contributions in the higher modes. To investigate the quality of the proposed preconditioner we look at the averaged numbers of linear iteration steps per solve. In Table 5.3, we observe similar behavior for the linear solver of the full Picard system as for the nonlinear solver, namely increasing numbers of iterations for increasing Re. But for different Strouhal numbers it seems that the maximum is reached between St = 0.1 and 0.2. Note that the Strouhal number of the



(a) $\|\hat{\mathbf{u}}_1^c(\mathbf{x})\|_2 = 0.009$.



(b) $\|\hat{\mathbf{u}}_2^c(\mathbf{x})\|_2 = 0.0009$.

Figure 5.20: Isosurfaces of the velocity magnitude of the first (a) and second (b) cos coefficient.

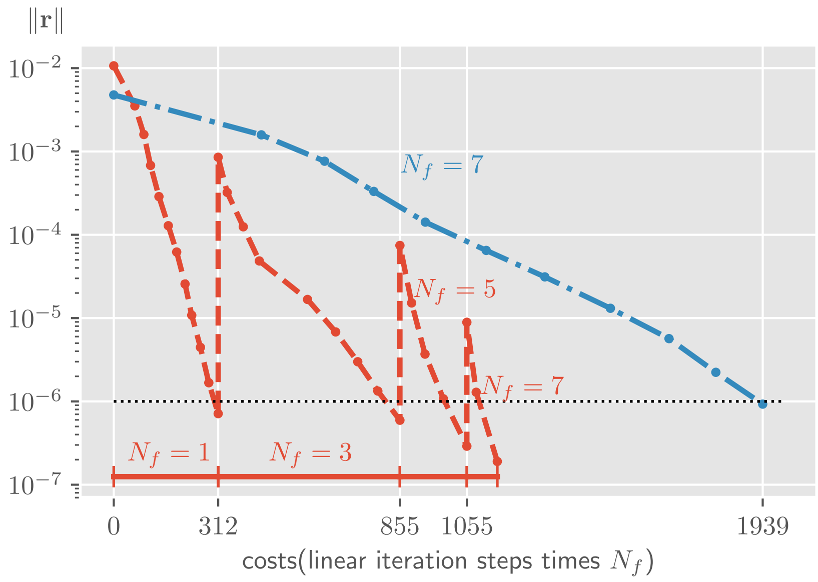


Figure 5.21: Comparison between *spectral refinement* ● and fixed N_f ●. Note that $\mathrm{tol}_{\mathbf{r}} = 10^{-6}$.

Table 5.2: Number of Picard iteration steps.

	Picard steps			
	St			
Re	0.05	0.1	0.2	0.4
10	6	4	11	11
100	16	18	18	16
200	23	25	25	23

vortex shedding behind a cylindrical obstacle is expected to be approximately 0.2 [57, Chapter 10]. For solving the time-periodic convection-diffusion problem, the number of iterations is very low which means that it is well approximated by the diagonal blocks. In Table 5.4, we investigate the linear iteration steps for the different stages of N_f in our method. We consider only the Strouhal number 0.2. Again we list the averaged numbers of iterations for one solve with the Picard matrix (3.25). The number of iterations is not increasing with the number of modes.

Table 5.3: Average number of linear iteration steps for the full problem and the upper left block $\mathcal{F}$.

	linear iteration steps for $\mathcal{H}$				linear iteration steps for $\mathcal{F}$			
	St							
Re	0.05	0.1	0.2	0.4	0.05	0.1	0.2	0.4
10	9.5	5.0	8.9	8.2	1.0	1.0	1.0	1.0
100	13.5	11.5	10.9	8.9	2.2	2.0	2.0	2.0
200	17.0	18.9	16.9	11.2	3.0	2.9	2.9	3.0

Table 5.4: Averaged number of linear iteration steps for the full problem and the upper left block.

	linear iteration steps for $\mathcal{H}$				linear iteration steps for $\mathcal{F}$			
	N_f							
Re	1	3	5	7	1	3	5	7
10	11.9	6.0			1.0	1.0		
100	16.6	10.2	6.0		1.9	1.9	2.0	
200	28.4	22.6	10.0	6.5	2.4	2.7	3.3	3.0

5.3.3 Scaling

To assess the parallel performance of our implementation, we measure the wall-clock time of one Picard update step. We consider the case Re = 100 and St = 0.1. All computations and timings were done on the Brutus cluster[2] at ETH Zürich with AMD Opteron 6174 12-core CPUs connected by an Infiniband QDR network.

---

[2] `https://scicomp.ethz.ch/wiki/Brutus`

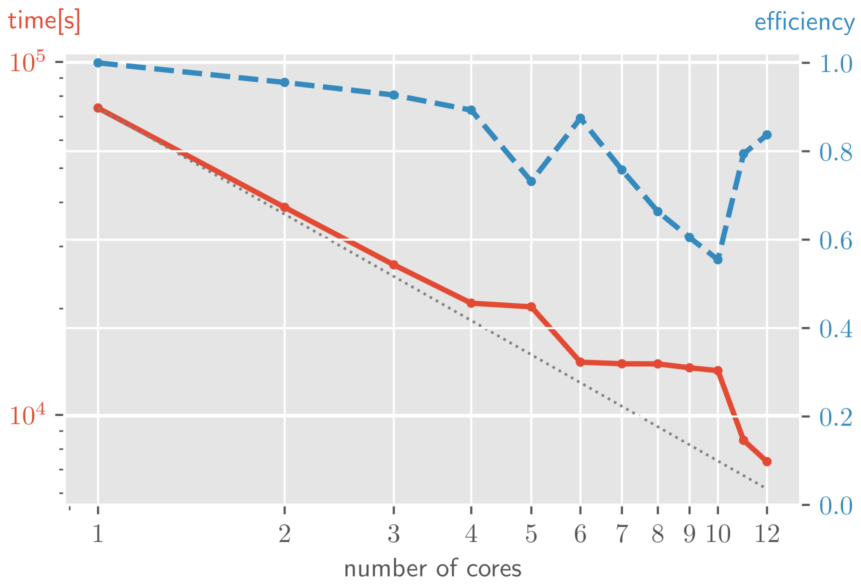


Figure 5.22: Strong scaling on multiple cores on a 12-core CPU. Wall-clock time is indicated by red solid line. Efficiency is indicated by blue dashed line.

First, we consider the performance on one CPU. A spatial grid with 49^3 grid points is used with twelve Fourier modes ($N_f = 11$). The number of cores varies between one and twelve. We consider only the parallelization in time, i.e. the spatial domain is not distributed to different processing units in this experiment. In Figure 5.22, we observe good parallel performance if the number of Fourier modes is a multiple of the number of cores. If this is not the case, i.e., for 5, 7, 8, 9, and 10 cores, the execution time equals the one of the next smaller dividing core number. The efficiency drops accordingly. The case of 11 cores is special. Here, each core executes one of the full modes 1 to 11 consisting of a cosine and a sine part. One of the cores has to compute in addition the zero mode. Since the zero mode contains only one mode instead of two the computational costs are only half the costs of the higher modes. Thus, the load of the core that computes mode 0 is only 50% higher than the average load. Therefore, the execution time with 11 cores is shorter compared to the 6 cores case.

Second, we consider the performance on multiple CPUs. Here, the spatial discretization is fixed with 65^3 grid points. In the strong scaling test we fix the number of Fourier modes N_f such that the problem size remains the same as the core count increases. Ideally, the execution time is inversely proportional to the number of used cores. In the weak scaling test we increase the Fourier modes in proportion with the increasing number of cores. Ideally, the execution time stays constant. In Figure 5.23, timings are plotted in double-logarithmic scale for both strong and weak scaling relative to runs on 32 cores. We observe good strong and weak scaling behavior.

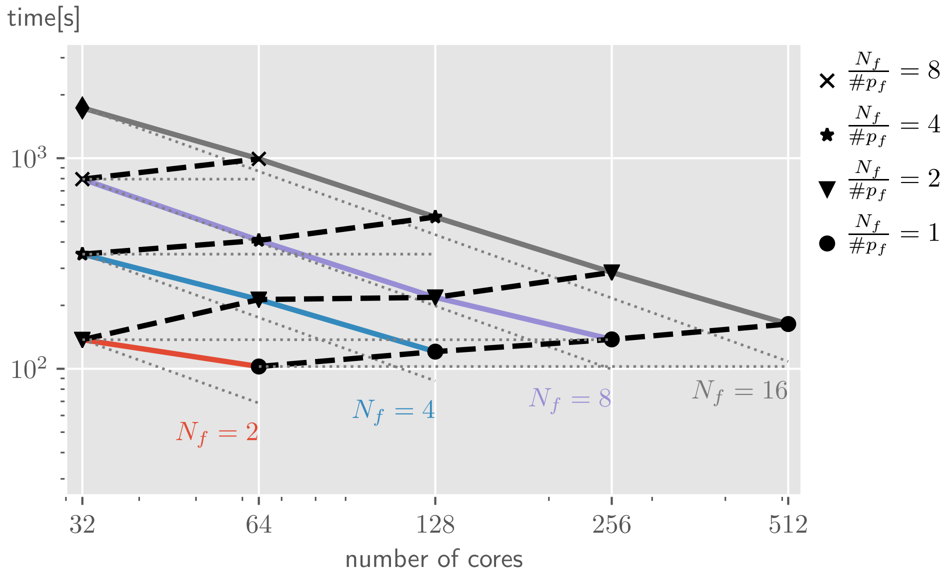


Figure 5.23: Strong scaling in color with solid lines, weak scaling in black and dashed lines, ideal scaling in gray and dotted lines.

5.3.4 Conclusions

The efficiency of the spectra in time solver has been demonstrated by numerical experiments. The method combines the fast convergence of Fourier series with the linear convergence of Picard's method. A parameter study has been performed to investigate the solver behavior for different Reynolds and Strouhal number. It shows that the cost for the nonlinear and the linear solvers rise for higher Reynolds numbers. It has been observed that the maximal cost is achieved for Strouhal numbers St between 0.1 and 0.2. These experiments show that the good properties of the least-squares commutator carry over to the time-periodic problem formulation. Furthermore the block diagonal approximation of the multi-harmonic convection-diffusion problem shows very good properties, such that for this problem the diagonal approximation of the multi-harmonic convection-diffusion problem could have been directly used in the triangular preconditioner as in (3.39). This is done in the next section. The scaling results presented in

Figure 5.23 promise good parallel performance, especially if very large problems with millions of spatial grid points and tenth of temporal Fourier modes will be considered.

5.4 Swept Hiemenz flow

In this section we investigate the applicability of the *spectral in time* method (cf. section 3.1) to the swept Hiemenz flow. The swept Hiemenz flow [47] is a model of a boundary layer at the attachment line of a swept wing. This flow can be seen as the composition of a two-dimensional boundary layer along the streamwise coordinate and a two-dimensional Hiemenz flow along the chordwise coordinate. Thus the swept Hiemenz flow is a fully three-dimensional flow. The stability and the transition to turbulence of this flow has been studied in great detail with many methods [22, 71, 86]. We can therefore validate our method with previous results computed by a time-stepping method. The transition to turbulence is happening along the streamwise coordinate. Therefore we can limit the computation domain along this coordinate to consider the flow as laminar, transitional or turbulent. We explore limits of our approach for these three cases.

5.4.1 Problem

We have the three-dimensional velocity profile of the swept Hiemenz flow, which we denote by $\mathbf{u}_{\mathrm{SHF}}(\mathbf{x})$. This flow $\mathbf{u}_{\mathrm{SHF}}(\mathbf{x})$ is a solution of the stationary tree-dimensional Navier–Stokes equations on the domain $\Omega = [0,\ \infty] \times [0,\ \infty] \times [-\infty,\ \infty]$. We consider on the other hand only the spatial subdomain $\Omega = [0,\ L_x] \times [0,\ L_y] \times [-L_z/2,\ L_z/2]$. On which we solve the tree-dimensional time-periodic Navier–Stokes equation

$$\begin{aligned} \alpha^2 \partial_t \mathbf{u} + \mathrm{Re}(\mathbf{u} \cdot \boldsymbol{\nabla})\mathbf{u} - \boldsymbol{\Delta}\mathbf{u} + \mathrm{Re}\boldsymbol{\nabla} p &= \mathbf{0}, \\ \boldsymbol{\nabla} \cdot \mathbf{u} &= 0, \end{aligned}$$

for $x \in \Omega$ and $t \in [0,\ 2\pi]$. On the spatial domain boundaries we use Dirichlet boundary conditions, where we prescribe the swept Hiemenz flow plus a time-periodic disturbance $\mathbf{u}_{\mathrm{SHF}}(\mathbf{x}) + \mathbf{u}_{\mathrm{PER}}(\mathbf{x}, t)$. $\mathbf{u}_{\mathrm{PER}}(\mathbf{x}, t)$ will be defined later. The x, y, and z-coordinate are the so-called wall-normal, streamwise, and chordwise direction, respectively. The reference length is given by $L_{\mathrm{ref}} = \sqrt{\nu / S^*}$. The reference velocity U_{ref} is given by the free-stream streamwise velocity v_∞. We will later in this section compare our results with previously computed results from [71], so we consider the same Reynolds number $\mathrm{Re} = 300$. A schematic set-up of the problem can be seen in Figure 5.24.

We disturb this base flow with an inflow perturbation at $y = 0$. This perturbation

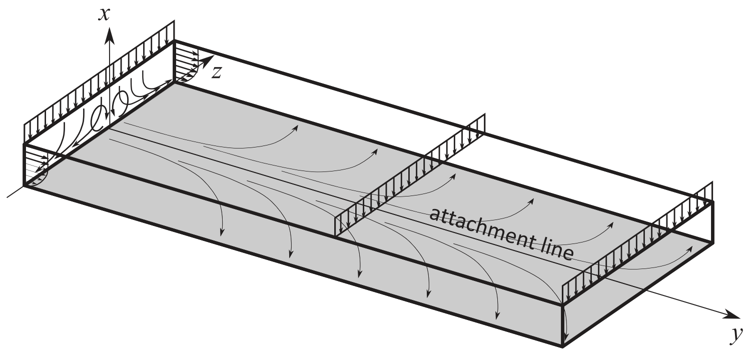


Figure 5.24: Schematic set-up of the time-periodically disturbed swept Hiemenz flow. This figure has been extracted and adjusted from [71, Figure 1].

changes its magnitude periodically $\mathbf{u}_{\mathrm{PER}}(\mathbf{x},t) = (A_1 + A_2 \sin t)[u'(x,z),\ \ 0,\ \ w'(x,z)]^T$. The disturbance velocities u' and w' are given for $x<b$ by

$$u'(x,z) = \begin{cases} \frac{1}{2}\sin\frac{\pi(z-z_c)}{b}\left(-\cos\frac{\pi(x-x_c)}{b}+\cos\frac{\pi(z+z_c)}{b}\right) & \text{if } |z-z_c|<b \text{ and } |z+z_c|<b,\\ \frac{1}{2}\sin\frac{\pi(z-z_c)}{b}\left(1+\cos\frac{\pi(x-x_c)}{b}\right) & \text{if } |z-z_c|<b \text{ and } |z+z_c|>b,\\ -\frac{1}{2}\sin\frac{\pi(z-z_c)}{b}\left(1+\cos\frac{\pi(x+x_c)}{b}\right) & \text{if } |z-z_c|>b \text{ and } |z+z_c|>b,\\ 0 & \text{otherwise.} \end{cases}$$

$$w'(x,z) = \begin{cases} -\frac{1}{2}\sin\frac{\pi(x-x_c)}{b}\left(\cos\frac{\pi(z-z_c)}{b}+\cos\frac{\pi(z+z_c)}{b}\right) & \text{if } |z-z_c|<b \text{ and } |z+z_c|<b,\\ -\frac{1}{2}\sin\frac{\pi(x-x_c)}{b}\left(1+\cos\frac{\pi(z-z_c)}{b}\right) & \text{if } |z-z_c|<b \text{ and } |z+z_c|>b,\\ \frac{1}{2}\sin\frac{\pi(x-x_c)}{b}\left(1+\cos\frac{\pi(z+z_c)}{b}\right) & \text{if } |z-z_c|>b \text{ and } |z+z_c|<b,\\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

The disturbance velocities are visualized in Figure 5.25. This perturbation is considered optimal [38], which means that the energy growth along the streamwise direction is maximized. We consider the case $A_1 = 10^{-1}$, $A_2 = 10^{-3}$, $\alpha \approx 7.93$. This Womersley number and Reynolds number corresponds to the reference frequency $f_{\mathrm{ref}} = 1/30$. From previous studies [71] we know that for these parameters, the break down to turbulence occurs the earliest in streamwise direction, namely for $y > 200$. So we investigate the

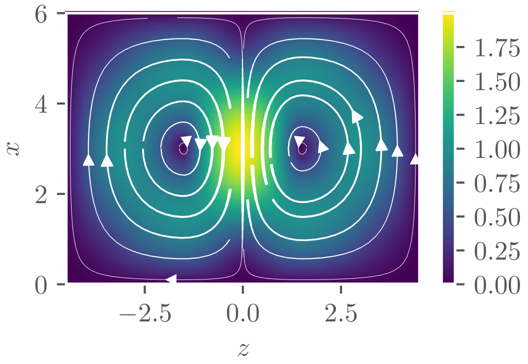


Figure 5.25: Streamlines of the perturbation field $[u', w']^T$ at $y = 0$. White lines indicate the streamlines. The color indicates the velocity magnitude.

performance of our solver by controlling the streamwise length. For the streamwise domain length $L_y = 75$ the solution is laminar, for $L_y = 150$ the solution is transient and for $L_y = 300$ the flow becomes turbulent. The parameters of the spatial discretization (grid and degrees of freedom (DOF)), parallelization and domain size are given in Table 5.5. The distances between grid points in the wall-normal coordinate are stretched due to the velocity profile of the base flow.

Table 5.5: Domain and spatial discretization

	L_x	L_y	L_z	spatial grid	spatial DOF	$np_x \cdot np_y \cdot np_z$
laminar	22.5	**75**	169	$65 \times 129 \times 385$	$12'900'000$	24
transitional	22.5	**150**	169	$65 \times 257 \times 385$	$25'700'000$	48
turbulent	22.5	**300**	169	$65 \times 513 \times 385$	$51'400'000$	96

The residuals based spectral refinement (Algorithm 3) is used with the residual stopping criterion (3.15) with a tolerance $\mathrm{tol}_{\mathbf{r}} = 10^{-6}$ and for the refinement $\mathrm{tol}_{\text{refine}} = 2$. Note that the norms are scaled with the number of coefficients $2N_f + 1$. The linear Picard problem (3.10) is solved with flexible GMRES up to a tolerance of 10^{-2} with the 2-by-2 block-triangular preconditioner (3.39), where the (1,1) block is a diagonal approximation of the multi-harmonic convection-diffusion matrix (3.38) and the (2,2) block is a least squares commutator based approximation of the Schur complement (3.35). The inner subproblems are also solved by the transpose-free quasi-minimal residual method (TFQMR) [31]. For the stationary and harmonic convection-diffusion subproblems we use a tolerance of 10^{-1}. The convection-diffusion problem is preconditioned with two V-cycles of multigrid with Gauss–Seidel iteration as smoother and as coarse grid solver with downwinding in y-direction and alternating directions in x and z direction. For explanations and references we refer to section 3.1.4.4. Also the Poisson problem is solved by TFQMR but with fixed number of iteration steps, that is 14. The reason for that is that this leads to a more equal load for all cores. As preconditioner for the Poisson problem we use eight multigrid V-cycles. As smoother and coarse grid solver we use eight Jacobi iteration steps. These are set much higher than in the previous experiments. This is necessary because of the grid stretching.

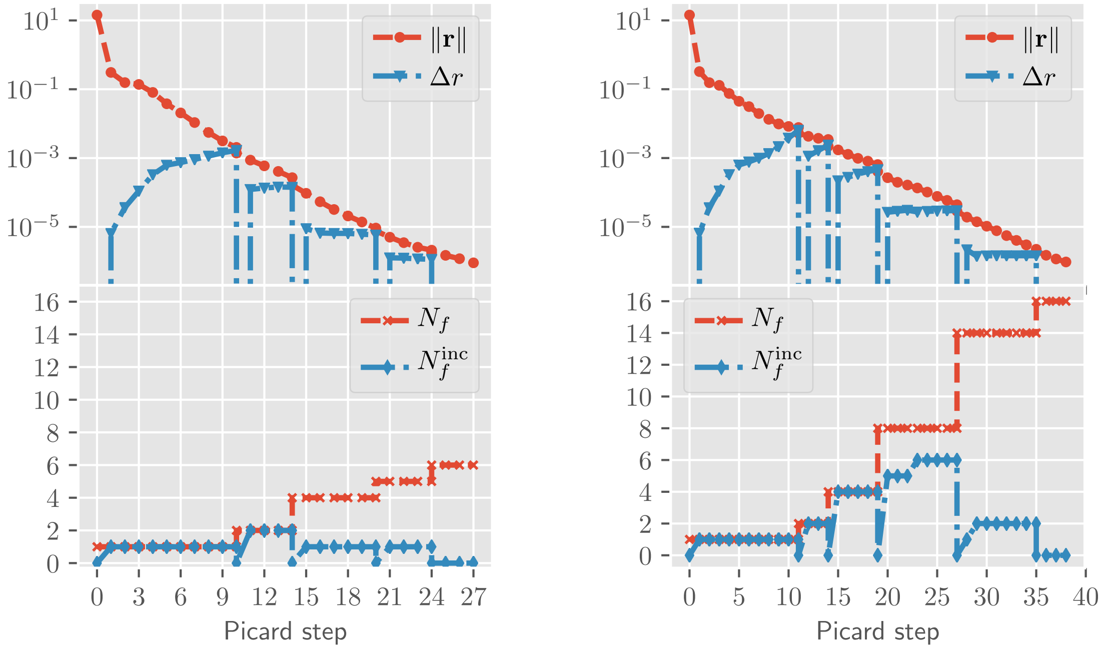
5.4.2 Results

First we analyze the results in terms of performance then with regard to the solution in comparison with the result of a time-stepping method.

In Figure 5.26, we can observe the performance of the *residual based spectral refinement* Algorithm 3 for the laminar ($L_y = 75$), the transitional ($L_y = 150$), and the turbulent case ($L_y = 300$). In the laminar and transitional case we observe convergence until the overall residual is below 10^{-6}. Furthermore we observe that the increase of the number of frequencies does not lead to a substantial increase of the residual as in the previous section where Algorithm 2 was used. We conclude that no compute power is wasted by not solving the lower number of frequencies problems completely. Furthermore the algorithm nicely adapts to the more complex flow structure of the laminar case by including more frequencies. But in the turbulent case we observe that the method is diverging after the 10th Picard step. There the algorithm is stopped, when the number of linear iteration steps is reaching the prescribed maximum of 200, cf. Figure 5.27. Note that despite the diverging residual, see Figure 5.26c, our approximate solution, see Figure 5.30b, is still converging to the expected solution, see Figure 5.29b. We expect that we get growing contributions to the residual that can not be resolved by the truncated Fourier series.

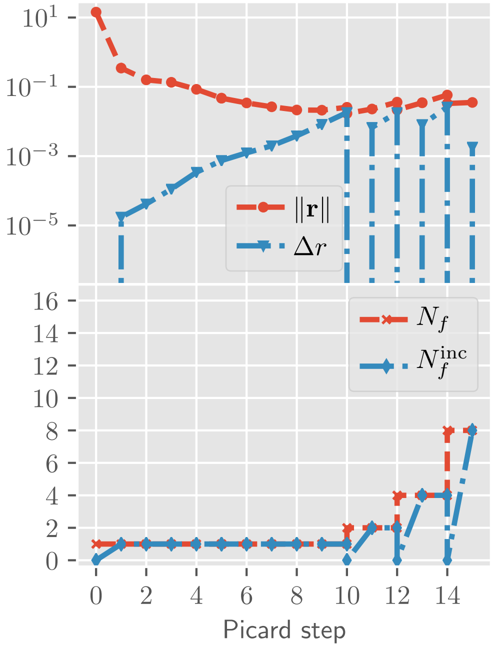
In Figure 5.27, we show the number of linear iteration steps that are necessary to solve the Picard problem for the three different cases. For the laminar case we observe that the number of linear iteration steps is around 16 and does not increase with the number of frequencies. For the transitional case we observe that the number of linear iteration steps is only a little bit larger than for the laminar case roughly around 20. But the number of linear iteration steps increases up to 30 along with the number of frequencies. Also the turbulent case has in the beginning only a slight increase of the number of linear iteration steps at around 24 in contrast to the laminar and transitional case. But as the number of frequencies is increasing also the number of linear iteration steps is increasing drastically. When the number of linear iterations steps is surpassing 400 iteration steps, we stop the linear and nonlinear solver. It cannot expected that this number of linear iteration steps is dropping for continuing the Picard iteration or through increasing the number of frequencies.

In Table 5.6 we show the performance of the linear solvers of the inner subproblems. We show the averaged number of linear iteration steps for the stationary and harmonic



(a) $L_y = 75$.

(b) $L_y = 150$.



(c) $L_y = 300$.

Figure 5.26: Performance of the *residual based spectral refinement* algorithm for the three different cases. On the top we show the residual in red with circular markers and show the residual increase through potential refinement (3.22) in blue with triangular markers. On the bottom we show the number of frequencies in red with crossed markers and the number of potential additional frequencies in blue with diamond markers.

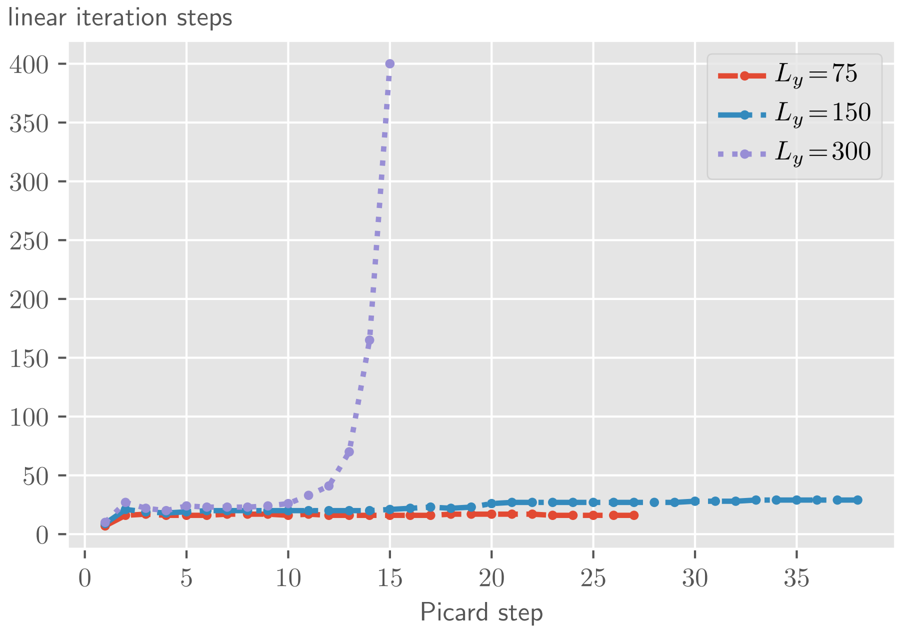


Figure 5.27: Linear iteration steps for the Picard problem.

Table 5.6: Average of the number of linear iteration steps for the stationary and harmonic convection-diffusion problem. Average of the achieved tolerance for the Poisson problem.

	$L_y = 75$	$L_y = 150$	$L_y = 300$
Stationary convection-diffusion problem	3.26	2.96	2.77
Harmonic convection-diffusion problem	18.7	20.3	40.6
Poisson problem	$1.79 \cdot 10^{-4}$	$8.22 \cdot 10^{-5}$	$7.67 \cdot 10^{-5}$

convection-diffusion problem for one solve. We observe that average number of linear iterations per solve is slightly decreasing for the stationary convection-diffusion problem, which is ideal. The averaged number of linear iterations for the harmonic convection-diffusion problem however is relatively large and is increasing from 19 to 44. As the number of linear iteration steps is fixed for the Poisson problem they do not vary, but the achieved tolerance varies, therefore we show the averaged achieved tolerance for one solve. The averaged relative achieved tolerance for Poisson problem stays the same and is independent of the problem size.

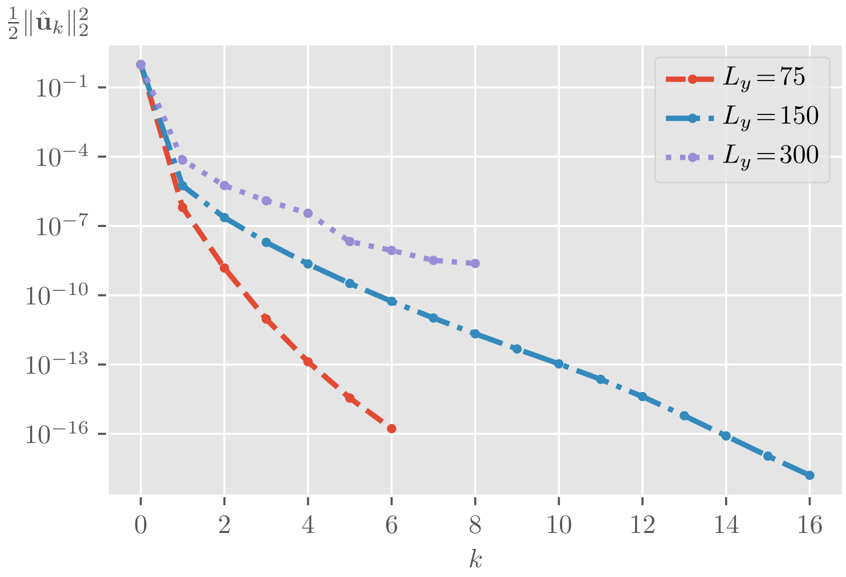


Figure 5.28: Energy spectrum.

In Figure 5.28, we show the energy density over the different coefficients k. Note that the norm here is scaled with the inversed domain size $1/(L_x L_y L_z)$. We observe that for the laminar case much fewer frequencies are needed and the energy is decaying much faster than for the transitional case. In case of the turbulent flow the energies are even higher. We note that our intermediate solution of the turbulent flow is not deemed converged with regard to the residual stopping criterion. We expect many more frequencies for a converged solution.

To validate the correctness of our solution we analyze the solution by its Fourier-Hermite modal energy densities. This modal energy density analysis, introduced in [73], transforms the disturbance flow velocities into Fourier series in time and then into Hermite polynomials in the chordwise direction. For the spectral in time method we get the

Fourier coefficients out of the box. It is only necessary to transfer the individual Fourier coefficients into Hermite modes. We only consider the pure disturbance without the base flow. As the base flow is constant in time, only the zero mode has to be adjusted to get the pure disturbance,

$$\hat{\mathbf{u}}^{0'} = \hat{\mathbf{u}}^{0} - \mathbf{u}_{\mathrm{SHF}}. \tag{5.8}$$

The Hermite polynomials are orthogonal with regard to the scalar product

$$(f \cdot g) = \int_{-\infty}^{\infty} fg \exp\left(-\frac{z^2}{2\gamma^2}\right) \mathrm{d}z.$$

They are recursively defined by

$$\begin{aligned} \mathrm{H}_0(z) &= \frac{1}{(\sqrt{2\pi}\gamma)^{\frac{1}{2}}}, \\ \mathrm{H}_1(z) &= \frac{1}{(\sqrt{2\pi}\gamma)^{\frac{1}{2}}} z, \\ \mathrm{H}_n(z) &= \frac{z\mathrm{H}_{n-1}(z) - \sqrt{n-1}\mathrm{H}_{n-2}(z)}{\sqrt{n}}, \quad n > 1. \end{aligned} \tag{5.9}$$

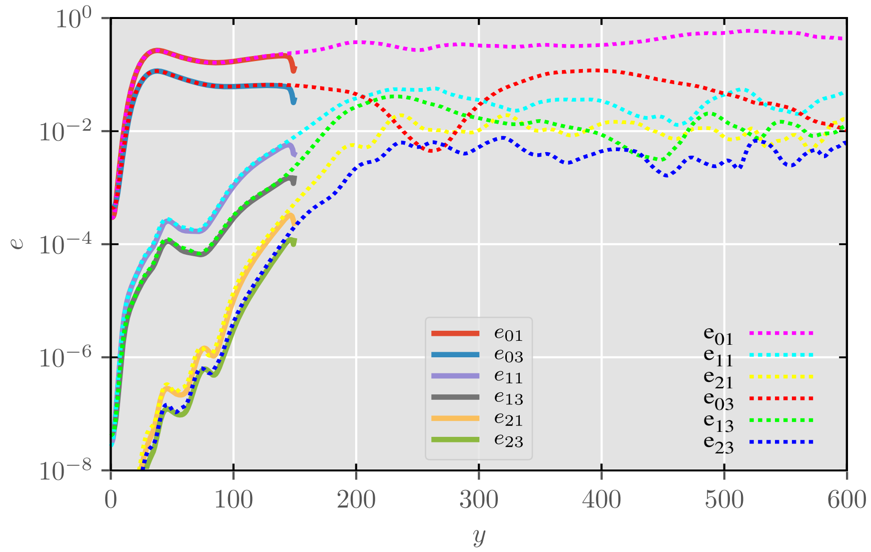
We transfer the velocity coefficients in x-direction into the chordwise Hermite modes by

$$\begin{aligned} \hat{u}'_{0n}(x,y) &= \int_{-L_z/2}^{L_z/2} \hat{u}^{0'} \mathrm{H}_n(z/\gamma) \exp\left(-\frac{z^2}{2\gamma^2}\right) \mathrm{d}z, \\ \hat{u}^{c/s}_{kn}(x,y) &= \int_{-L_z/2}^{L_z/2} \hat{u}^{c/s}_{k} \mathrm{H}_n(z/\gamma) \exp\left(-\frac{z^2}{2\gamma^2}\right) \mathrm{d}z. \end{aligned} \tag{5.10}$$

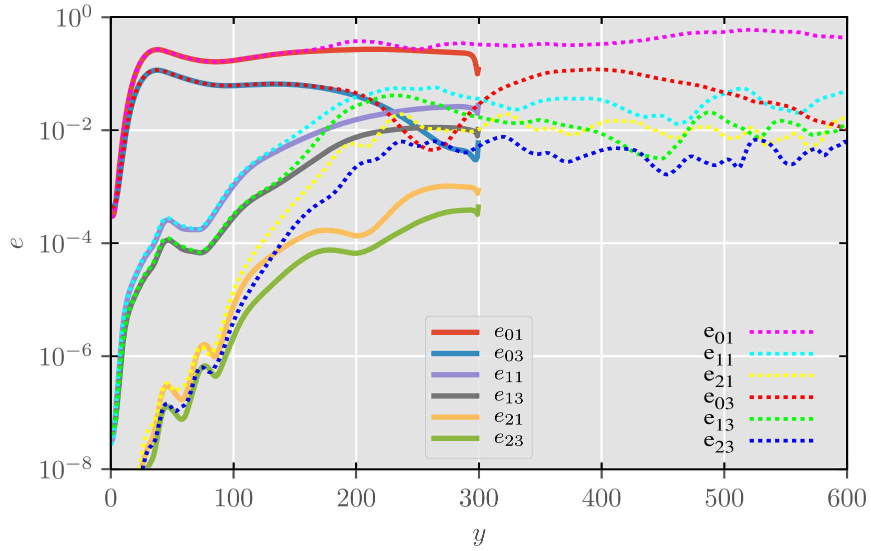
The velocities coefficients in the y and z-direction are treated the same as the velocity coefficients in x-direction. To get the modal Fourier-Hermite energy densities along the streamwise coordinate y, we sum up the squares of all three velocity directions and integrate over the wall-normal coordinate x

$$\begin{aligned} e_{0n}(y) &= \frac{1}{\pi} \int_0^{\infty} \left(|\hat{u}'_{0n-1}|^2 + |\hat{v}'_{0n-1}|^2 + |\hat{w}'_{0n}|^2\right) \mathrm{d}x, \\ e_{kn}(y) &= \frac{1}{\pi} \int_0^{\infty} \left(|\hat{u}^{c}_{kn-1}|^2 + |\hat{v}^{c}_{kn-1}|^2 + |\hat{w}^{c}_{kn}|^2 + |\hat{u}^{s}_{kn-1}|^2 + |\hat{v}^{s}_{kn-1}|^2 + |\hat{w}^{s}_{kn}|^2\right) \mathrm{d}x. \end{aligned} \tag{5.11}$$

Note that, we combine the order n in the chordwise coordinate with the $n-1$ order in wall-normal and streamwise coordinates. This is motivated by the structure of the vortices [72].



(a) Transitional flow.



(b) Turbulent flow.

Figure 5.29: Modal energy densities. Dashed lines are extracted from [71, Figure 5b]. Sold lines have been computed by the spectral in time method.

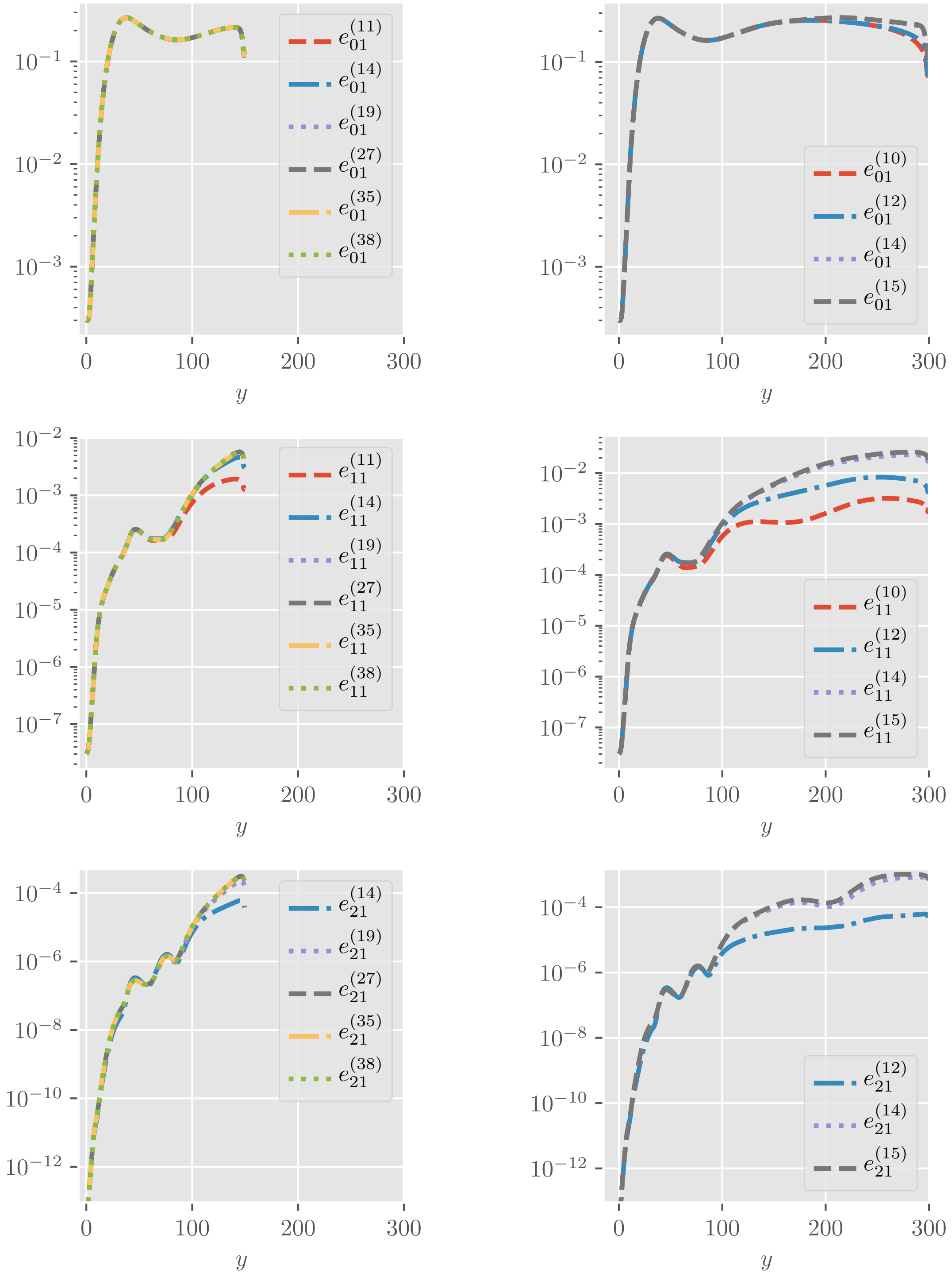
For the comparison in Figure 5.29a and 5.29b we extracted the data from [71, Figure 5b]. We compare the energy densities of the solution of the spectral in time method and the solution of a time-stepping method. For the time-stepping method a explicit Runge–Kutta scheme has been used. The case $L_y = 75$ is not shown as the data matches the same way as for the case $L_y = 150$. The transitional case matches the data from the time-stepping perfectly. The turbulent case matches the data only up to $y = 150$, afterwards the two results differ, which is expected as the solution is not converged.

In Figure 5.30, we can see the energy modes for different Picard steps. It can be observed that the solution converges more quickly upstream and that for lower modes a good approximation of the solution is obtain after only a couple of Picard steps.

To visualize our results we show the streamwise velocity component in a plane at $x=1$ for the zero coefficient in Figure 5.31 and for the first cosine coefficient in Figure 5.32.

In Figure 5.31, we compare temporal snapshots of the spectral in time method with the time-stepping method from [71]. For the the spectral in time method, we have added the coefficient to get the field for a time snapshot at $t=0$ $(\hat{\mathbf{u}}^0+\sum_{i=1}^{N_f}\hat{\mathbf{u}}_k^c)$. We show the streamwise velocity component in a plane at $x=1$.

At last we show the isosurface of the λ_2 vortex criterion [52] in Figure 5.34. We show the result from the spectral in time method in comparison with the time-stepping method from [71]. They should be the same up to $y=100$ according to Figure 5.29b. We assume that the differnce is due to the slightly different spatial discretization.



(a) $L_y = 150$.

(b) $L_y = 300$.

Figure 5.30: Energy densities for first Hermite mode and for the zero Fourier mode on top, the first Fourier mode in the middle, and the second Fourier mode on the bottom. On the left side is the transitional case and on the right side is the turbulent case. They are plotted at the Picard steps when the solution is refined.

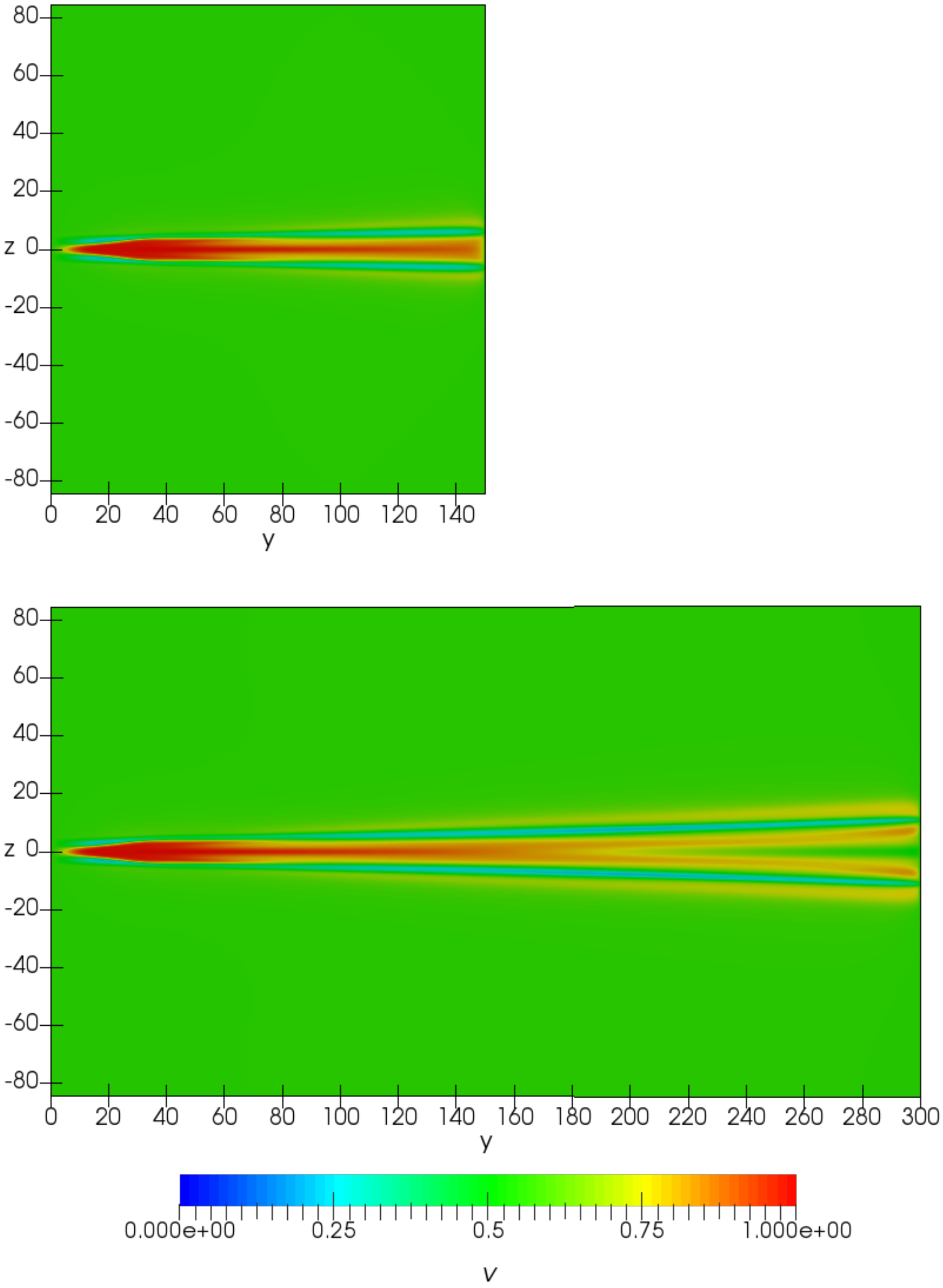


Figure 5.31: Streamwise velocity of $\hat{\mathbf{u}}^0$ at $x = 1$.

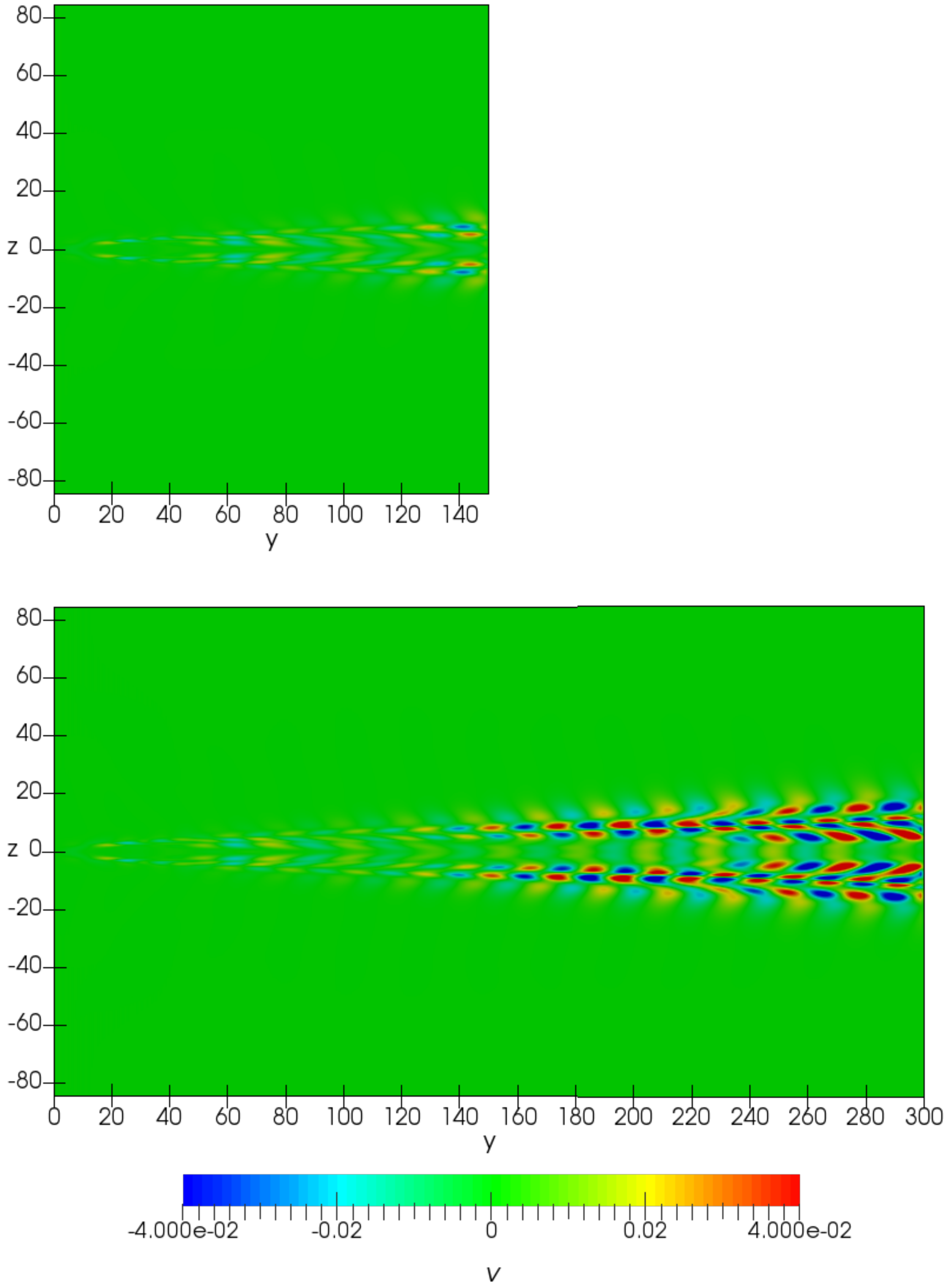
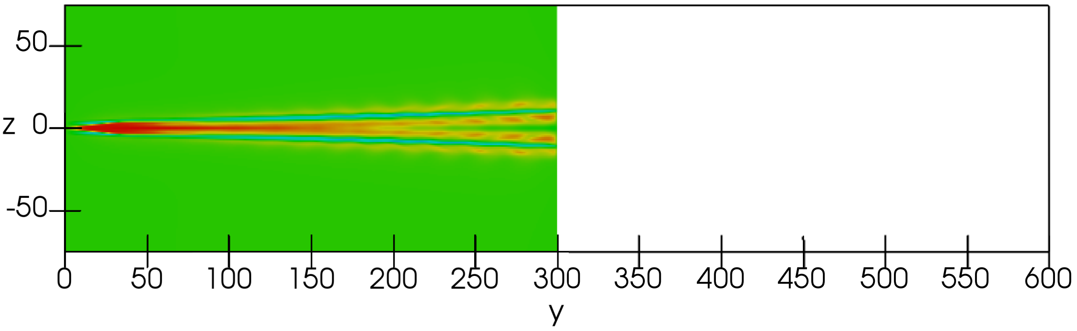
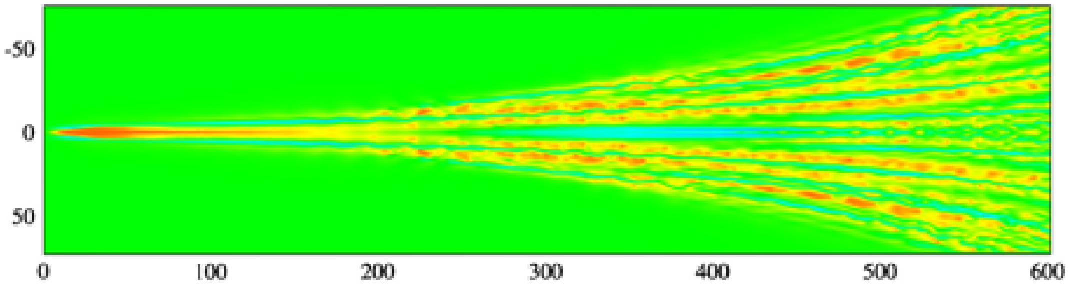


Figure 5.32: Streamwise velocity of $\hat{\mathbf{u}}_1^c$ at $x = 1$.

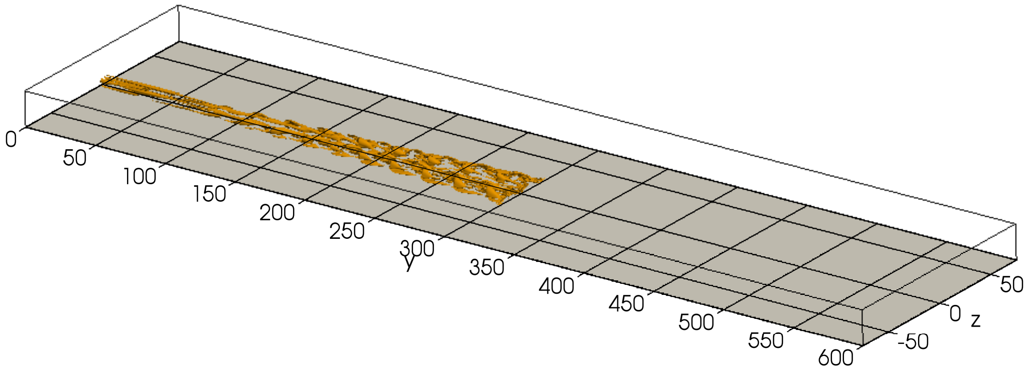


(a) Spectral in time method.

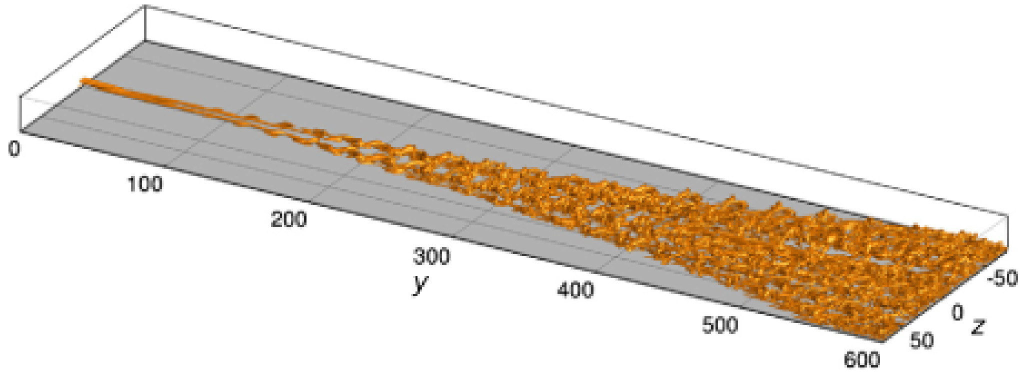


(b) Time-stepping method. The green color indicates a velocity of 0.5 which corresponds to the base flow configuration, whereas red and blue indicate high and low velocity, respectively.

Figure 5.33: Snapshots of the streamwise velocity in a plane at $x = 1$ for the spectral in time method on top and the time-stepping method below. The plot of the time-stepping method has been extracted and adjusted from [71, Figure 4.b].



(a) Spectral in time method.



(b) Time-stepping method.

Figure 5.34: Isosurface of $\lambda_2 = -10^{-4}$. For comparison, we show the result from the spectral in time method on top and the result from the time-stepping method below. The plot of the time-stepping method has been extracted and adjusted from [71, Figure 3].

5.4.3 Conclusions

We applied the spectral in time method to a “real world” problem. We could show that the spectral in time method works well for laminar and transitional problems and produces correct results. Only for turbulent problems the method shows deficits. This is not too surprising as the core assumption of a pure time-periodic flow is violated. Nevertheless we expect that the spectral in time method could be improved such that it can compute efficiently close an approximation of the turbulent flow.

We expect that the diverging of the nonlinear solver can be overcome by using more frequencies. As we expect that the residual is getting more and more contributions that can be only resolved by coefficients belonging to higher frequencies. But to do that it is necessary to be able to solve the linear Picard problem in a reasonable amount of time.

The current linear solver for the Picard problem is suffering from a strong increase of linear iteration steps when the solution is getting close to the turbulent solution. The reason for that is that presumably the approximation of multi-harmonic convection-diffusion is losing quality. So we propose to include in the preconditioner also more off-diagonals, which might mean that the parallelization in the time domain has to be reduced. This does not necessarily mean that efficiency is reduced, as the traversing of the transition phase is still prohibited and as considering multiple Fourier coefficients for every spatial grid points increases the parallelize-ability in space.

The performance of the harmonic convection-diffusion problem is not fully satisfactory. The iteration numbers are relatively high and they are increasing with the system size. The solution of these subproblems is becoming the most time-consuming part of the overall time to solution. This means if the preconditioner could be improved such that by equal costs the iteration numbers is reduced from 44 to 10 then the time to solution could be approximately decreased four times. We suppose this can be achieved by constructing a preconditioner that better incorporates the convective terms. This could be done for example by multigrid that is applied to the full harmonic convection-diffusion problems not only to its diagonal blocks. This has been implemented but a good smoother has not yet been found. We detect a potential decrease of the time to solution but this will have no influence on the two main problems, the diverging Picard iteration and the increasing number of linear iteration steps for the Picard problem, discussed above.

6

Conclusions and future work

In this chapter we conclude this thesis and give an outlook on potential future work.

6.1 Conclusions

We developed and implemented new efficient solvers for time-periodic forced Navier–Stokes problems. The new solvers are parallel in time, so that the implementation has better scalability than just using spatial domain decomposition. Furthermore, the solvers are able to circumvent the necessity to simulate long transition phases for obtaining the time-periodic steady state. Different physical problems have been solved for verification and optimization such as a Rayleigh streaming, a Taylor–Green vortex, a three-dimensional channel flow with an oscillating obstacle, and a swept Hiemenz flow. In Figure 6.1, we show for which parameters we applied the spectral in time method. Overall we observed that if the Womersley number α or the Strouhal number St is large enough our spectral in time method is faster than a time-stepping method. But in general it is hard to predict if our spectral in time method can be applied successfully. As seen for the swept Hiemenz flow this strongly depends on the property of the flow, namely if the flow is laminar, transitional, or turbulent. The classification of a flow in those categories is a research topic in itself.

We could answer several research questions. First we could show that the new spectral in time method promises good strong and weak temporal scalability. Also we could show that the scalability can be extended beyond the saturation of the spatial parallelization. Furthermore we addressed the question if it is possible to compute the time-periodic steady state faster than a commonly used time-stepping method. We could show that for certain cases, by comparing our spectral in time method with a classical time-stepping method. Furthermore we investigated what happens in the case of turbulence when the time-periodic assumption holds only for averaged quantities. It appears that the

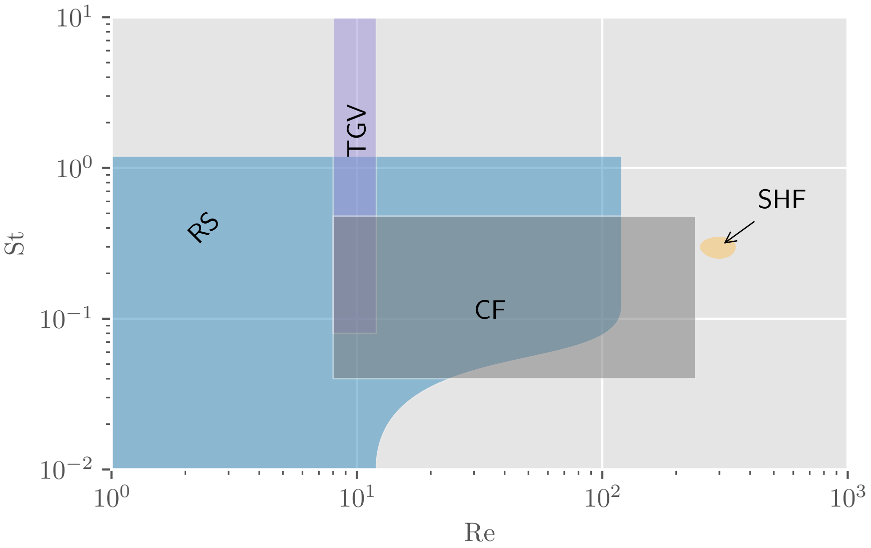


Figure 6.1: The colored areas denote the partially successful application of the spectral in time method. RS, TGV, CF, and SHF stands for Rayleigh streaming, Taylor–Green vortex, channel flow, and swept Hiemenz flow, respectively.

current method is not able to handle turbulent flows as the method is not converging to a solution in a reasonable amount of compute time and compute resources. Further investigations are necessary.

To compare the spectral in time method with parallel-in-time methods, one has to keep in mind that they are designed for two different problems. Most parallel-in-time methods are designed for general transient and unsteady problems, whereas the spectral in time method is limited to time-periodic problems with a priori known fundamental frequency. So parallel-in-time methods can traverse the transition faster than a sequential time-stepping method, but will not circumvent the traversal. So the application area for time-stepping methods can be broaden by introducing parallel-in-time methods. But nevertheless if the transition phase is long enough the spectral in time method can be still faster.

The novel software tool can highly efficiently solve time-periodic Navier–Stokes problems using different temporal discretization. The software is put under a free software license, so that it can be used by all researchers who are studying time-periodic Navier–Stokes problems. These contributions are also significant to a wider scientific community, because the modular approach of the parallelization concept ensures its applicability to other existing flow solvers. This means that the scheme for discretizing and parallelizing the time axis does not impose any requirements on (possibly already existing) discretizations (e.g. finite element method, finite volume method or spectral method) and/or parallelization concepts for the spatial dimensions. The resulting method can also be adapted to time-periodic problems that arise in other research fields, for example in chemistry or physics.

6.2 Future work

Possible future work can be divided in two categories. The first category are possible applications where the developed methods can be used. The second category are improvements of the current methods such that either the current applications can be solved even more efficiently or the application range can be broadened.

To start with possible applications, we propose to apply our method to various flow problems that occur in biofluid dynamics. In biofluid dynamics flows are investigated that occur inside or outside of living beings. Many such flows are strongly periodically driven for example human blood flow.

Another possible application is the uncertainty quantification of time-periodic steady states. Most methods for uncertainty quantification need the solution of many flow problems for different sets of parameters. The application of a classical time-stepping method becomes unfeasible as the transition phase to reach the steady state has to be traversed for every parameter set. For our spectral in time or finite differences in time method however not only the cost of computing the transition phase is circumvented but also solutions from previously computed sets of parameters can be reused as initial guesses for a new set of parameters. These initial guesses are potentially close to the solution of the new set of parameters.

We showed that the efficiency of the spectral in time method could be improved by spectral refinement. This has been done for whole spatial domains. We assume that the efficiency could be further increased by introducing the spectral refinement not to the global spatial domain, but individually to spatial subdomains. This means that spatial areas with no or little temporal variation are discretized by few coefficients, and areas with strong temporal variation are discretized with many coefficients. This can reduce memory consumption and reduce the amount of computations.

Another way to make this time-periodic method applicable to turbulent flows can be turbulent modeling. We propose turbulent modeling in such a way that one solves only for the averaged quantities that are indeed time-periodic. This could be done for example by using the described discretizations to the Reynolds averaged Navier–Stokes equations.

A

List of integrals of trigonometric functions

We provide here a list of integrals of trigonometric functions, that occur in section 2.2.1. We assume $k, l, n \in \mathbb{N}^+$.

$$\int_0^{2\pi} \sin(kt)\,\mathrm{d}t = 0 \tag{A.1}$$

$$\int_0^{2\pi} \cos(kt)\,\mathrm{d}t = 0 \tag{A.2}$$

$$\frac{1}{\pi}\int_0^{2\pi} \sin(kt)\sin(lt)\,\mathrm{d}t = \delta_{kl} \tag{A.3}$$

$$\frac{1}{\pi}\int_0^{2\pi} \sin(kt)\cos(lt)\,\mathrm{d}t = 0 \tag{A.4}$$

$$\frac{1}{\pi}\int_0^{2\pi} \cos(kt)\cos(lt)\,\mathrm{d}t = \delta_{kl} \tag{A.5}$$

$$\frac{1}{\pi}\int_0^{2\pi} \sin(nt)\sin(kt)\sin(lt)\,\mathrm{d}t = 0 \tag{A.6}$$

$$\frac{1}{\pi}\int_0^{2\pi} \sin(nt)\cos(kt)\cos(lt)\,\mathrm{d}t = 0 \tag{A.7}$$

$$\frac{1}{\pi}\int_0^{2\pi} \sin(nt)\sin(kt)\cos(lt)\,\mathrm{d}t = \frac{1}{2}\left[+\delta_{n(k+l)} - \delta_{(n+k)l} + \delta_{(n+l)k}\right] \tag{A.8}$$

$$\frac{1}{\pi}\int_0^{2\pi} \sin(nt)\cos(kt)\sin(lt)\,\mathrm{d}t = \frac{1}{2}\left[+\delta_{n(k+l)} + \delta_{(n+k)l} - \delta_{(n+l)k}\right] \tag{A.9}$$

$$\frac{1}{\pi}\int_0^{2\pi} \cos(nt)\sin(kt)\sin(lt)\,\mathrm{d}t = \frac{1}{2}\left[-\delta_{n(k+l)} + \delta_{(n+k)l} + \delta_{(n+l)k}\right] \tag{A.10}$$

$$\frac{1}{\pi}\int_0^{2\pi} \cos(nt)\cos(kt)\cos(lt)\,\mathrm{d}t = \frac{1}{2}\left[+\delta_{n(k+l)} + \delta_{(n+k)l} + \delta_{(n+lk)}\right] \tag{A.11}$$

B

Backtracking

There are different problems that can occur solving nonlinear problems by an inexact fixed-point iteration. The initial guess might be outside of the ball of convergence. The inexact linear solver might fail to find a good reducing direction. If these problems occur the residual will not decrease but increase. But we are still assuming a descent direction.

So if the residual is increasing $\|\mathbf{r}^{(m+1)}\| > \|\mathbf{r}^{(m)}\|$, there are different line search methods. A line search method tries to find a step width λ such that

$$\begin{aligned} &\|\mathbf{r}^{(m+1)}\| \leq \|\mathbf{r}^{(m)}\| \text{ with} \\ &\mathbf{q}^{(m+1)} = \mathbf{q}^{(m)} + \lambda\delta\mathbf{q} \end{aligned} \tag{B.1}$$

Backtracking [55, Chapter 8] is a simple and robust line search method. It starts with the step width $\lambda = 1$, if the new solution does not lead to a reduction of the residual norm, the step width is halved. This is repeated until a step width is found, that decreases the residual norm or a minimal step width λ_{min} is reached (cf. Algorithm 5). Therefore we replace the steps 6–7/7–8 of Algorithm 1, 2, and 3 with backtracking.

Algorithm 5 Backtracking

```
$\lambda \leftarrow 1$.
$\mathbf{q}^{(m+1)} \leftarrow \mathbf{q}^{(m)} + \lambda\delta\mathbf{q}$
Compute $\mathbf{r}^{(m+1)}$.
while $\|\mathbf{r}^{(m+1)}\| > \|\mathbf{r}^{(m)}\|$ and $\lambda > \lambda_{\text{min}}$ do
    $\lambda \leftarrow \lambda/2$.
    $\mathbf{q}^{(m+1)} \leftarrow \mathbf{q}^{(m)} + \lambda\delta\mathbf{q}$
    Recompute $\mathbf{r}^{(m+1)}$.
```

C

Computing null space of the pressure problem

As the matrix in (3.52) is singular it is necessary to make sure that the right hand side is in the image of the matrix $\operatorname{span}(\mathrm{DJ}^{-1}\mathrm{G}) = \operatorname{null}((\mathrm{DJ}^{-1}\mathrm{G})^T)^{\perp}$. To make sure of that, the right hand side vector r_p is orthogonalized to a null space vector Ψ of $\mathrm{DJ}^{-1}\mathrm{G}$

$$\hat{r}_p' = \hat{r}_p - \frac{(\Psi \cdot \hat{r}_p)}{(\Psi \cdot \Psi)}\Psi. \tag{C.1}$$

A vector $\Psi \neq 0$ in the left null space satisfies

$$\Psi^T \mathrm{DJ}^{-1}\mathrm{G} = 0^T, \tag{C.2}$$

which is equivalent to

$$\mathrm{G}^T \mathrm{J}^{-T} \mathrm{D}^T \Psi = 0. \tag{C.3}$$

For periodic boundary conditions, J is the identity matrix, and the rows of D^T sum to zero, such that $\Psi = 1$ everywhere. The right and left null space are then the same. In the case of Dirichlet boundary conditions D and J have full rank. The rank deficiency is due to G. We expand the three operators with regard to the individual velocity components

$$\mathrm{D}^T = \begin{pmatrix} D_x^T \\ D_y^T \\ D_z^T \end{pmatrix}, \quad \mathrm{J}^T = \begin{pmatrix} J_x^T & & \\ & J_y^T & \\ & & J_z^T \end{pmatrix}, \quad \mathrm{G}^T = \begin{pmatrix} G_x^T & G_y^T & G_z^T \end{pmatrix}. \tag{C.4}$$

Here, D_i and G_i denote the partial derivatives in direction i. J_i denotes the ith component of the diagonal J. Those operators have the dimensions

$$\begin{aligned}
&D_x^T \in \mathbb{R}^{(N_x+1)N_yN_z \times N_xN_yN_z}, \quad J_x^T \in \mathbb{R}^{(N_x+1)N_yN_z \times (N_x+1)N_yN_z}, \quad G_x^T \in \mathbb{R}^{N_xN_yN_z \times (N_x+1)N_yN_z},\\
&D_y^T \in \mathbb{R}^{N_x(N_y+1)N_z \times N_xN_yN_z}, \quad J_y^T \in \mathbb{R}^{N_x(N_y+1)N_z \times N_x(N_y+1)N_z}, \quad G_y^T \in \mathbb{R}^{N_xN_yN_z \times N_x(N_y+1)N_z},\\
&D_z^T \in \mathbb{R}^{N_xN_y(N_z+1) \times N_xN_yN_z}, \quad J_z^T \in \mathbb{R}^{N_xN_y(N_z+1) \times N_xN_y(N_z+1)}, \quad G_z^T \in \mathbb{R}^{N_xN_yN_z \times N_xN_y(N_z+1)}.
\end{aligned} \tag{C.5}$$

So equation (C.3) can be rewritten as

$$\left(D_x^T J_x^{-T} G_x^T + D_y^T J_y^{-T} G_y^T + D_z^T J_z^{-T} G_z^T\right) \Psi = 0. \tag{C.6}$$

As the operators are defined over three dimensional domains and we are using finite differences they can be written as dyadic products. The operators in x-direction read as

$$D_x = \tilde{D}_x \otimes \tilde{I}_y \otimes \tilde{I}_z, \quad J_x = \tilde{J}_x \otimes \tilde{I}_y \otimes \tilde{I}_z, \quad G_x = \tilde{G}_x \otimes \tilde{I}_y \otimes \tilde{I}_z, \tag{C.7}$$

where $\tilde{D}_x^T \in \mathbb{R}^{N_x+1 \times N_x}$ and $\tilde{G}_x^T \in \mathbb{R}^{N_x \times N_x+1}$ are the one dimensional transposed first derivative operators in x direction. $\tilde{I}_x \in \mathbb{R}^{N_y \times N_y}$ and $\tilde{I}_z \in \mathbb{R}^{N_z \times N_z}$ are the identity matrices. The operator $\tilde{J}_x \in \mathbb{R}^{N_x+1 \times N_x+1}$ has the following structure in case of Dirichlet boundary conditions

$$\tilde{J}_x = \begin{pmatrix}
c_0 & c_1 & \cdots & c_{bu} & \\
0 & 1 & & & \\
 & & \ddots & & \\
 & & & 1 & 0\\
 & c_{N_x-bu} & \cdots & c_{N_x-1} & c_{N_x}
\end{pmatrix}. \tag{C.8}$$

bl and bu are the lower and upper stencil widths. c are the coefficients to interpolate the velocity values to the boundary. The $\tilde{J}_x$ operator can be easily transposed and inverted. The transposed inverse of $\tilde{J}_x$ is

$$\tilde{J}_x^{-T} = \begin{pmatrix}
1/c_0 & 0 & & & \\
-c_1/c_0 & 1 & & & -c_{N_x-bu}/c_{N_x}\\
\vdots & & \ddots & & \vdots\\
-c_{bu}/c_0 & & & 1 & -c_{N_x-1}/c_{N_x}\\
 & & & 0 & 1/c_{N_x}
\end{pmatrix}. \tag{C.9}$$

Because of structure in (C.7), we can compute the left null space in x-direction by

$$\tilde{G}_x^T \tilde{J}_x^{-T} \tilde{D}_x^T \tilde{\Psi}_x = 0. \tag{C.10}$$

For simplicity we show the computation of the left null space vector only in one dimension $\tilde{\Psi}_x$, the other dimension $\tilde{\Psi}_y$ and $\tilde{\Psi}_z$ can be treated analogously. A left null space vector in two or three dimension can then be constructed from the dyadic product.

$$\Psi = \tilde{\Psi}_x \otimes \tilde{\Psi}_y \otimes \tilde{\Psi}_z \tag{C.11}$$

As stated before $\tilde{D}_x$ and $\tilde{J}_x$ have full rank but not $\tilde{G}_x$. So to find $\tilde{\Psi}_x$ from (C.10), we construct the null space vector first for $\tilde{G}_x^T$ and from that go to the right. The structure of $\tilde{G}_x$ is given by

$$\begin{pmatrix}
0 & 0 & \cdots & 0 & & & & & \\
g_{1,0} & g_{1,1} & \cdots & g_{1,g_u} & & & & & \\
g_{2,-1} & g_{2,0} & g_{2,1} & \cdots & g_{2,g_u} & & & & \\
\vdots & \ddots & \ddots & \ddots & & \ddots & & & \\
g_{1-g_l,g_l} & \cdots & g_{1-g_l,-1} & g_{1-g_l,0} & g_{1-g_l,1} & \cdots & g_{1-g_l,g_u} & & \\
 & \ddots & & \ddots & \ddots & \ddots & & \ddots & \\
 & & g_{N_x-1-g_u,g_l} & \cdots & g_{N_x-1-g_u,-1} & g_{N_x-1-g_u,0} & g_{N_x-1-g_u,1} & \cdots & g_{N_x-1-g_u,g_u} \\
 & & & \ddots & \ddots & \ddots & \ddots & & \vdots \\
 & & & & g_{N_x-1,g_l} & \cdots & \cdots & g_{N_x-1,-1} & g_{N_x-1,0} \\
 & & & & 0 & \cdots & \cdots & 0 & 0
\end{pmatrix}, \tag{C.12}$$

where $g_{i,j}$ is the coefficient to compute the first derivative at the grid point x_i from the function values at x_j. The lower and upper stencil width are denoted by g_l and g_u. The first and last rows of $\tilde{G}_x$ are zero as the pressure gradient does not contribute on the boundaries. The null space of $\tilde{G}_x^T$ is therefore

$$\operatorname{null}\left(\tilde{G}_x^T\right) = \operatorname{span} \underbrace{\left\{ \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \right\}}_{:=\tilde{\Phi}_x}. \tag{C.13}$$

To compute the null space of the $\tilde{G}_x \tilde{J}_x^{-T}$ we transform the null space (C.13) $\tilde{G}_x$ with

$\tilde{J}_x^T$ from the left. So we obtain

$$\begin{aligned}\operatorname{null}\Big(\tilde{G}_x^T \tilde{J}_x^{-T}\Big) &= \operatorname{span}\Big\{\tilde{J}_x^T \tilde{\Phi}_x\Big\}\\ &= \underbrace{\operatorname{span}\left\{\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{bu} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c_{N_x-bl} \\ \vdots \\ c_{N_x-1} \\ c_{N_x} \end{bmatrix}\right\}}_{:=\tilde{\chi}_x}.\end{aligned} \tag{C.14}$$

So we finally can compute $\tilde{\Psi}_x$ from

$$\tilde{D}_x^T \tilde{\Psi}_x = \tilde{\chi}_x. \tag{C.15}$$

For the left null space of the full Picard problem we define a new vector

$$\mathbf{\Phi} = \begin{bmatrix} \tilde{\Phi}_x \otimes \tilde{I}_y \otimes \tilde{I}_z \\ \tilde{I}_x \otimes \tilde{\Phi}_y \otimes \tilde{I}_z \\ \tilde{I}_x \otimes \tilde{I}_y \otimes \tilde{\Phi}_z \end{bmatrix}. \tag{C.16}$$

From (C.13) we know already that $\mathrm{G}^T\mathbf{\Phi} = \mathbf{0}$. Furthermore we can easily see that $\mathbf{\Delta\Phi} = \boldsymbol{\chi}$, where $\boldsymbol{\chi}$ is the three dimensional expanded version of χ_i. The null space of the transposed Picard system is therefore given by

$$\operatorname{null}\Big(\mathcal{H}^T\Big) = \operatorname{span}\left\{\begin{bmatrix} \mathbf{\Phi} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ -\Psi \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{0} \\ \mathbf{\Phi} \\ \vdots \\ \mathbf{0} \\ 0 \\ -\Psi \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{\Phi} \\ 0 \\ 0 \\ \vdots \\ -\Psi \end{bmatrix}\right\}. \tag{C.17}$$

D

Getting Pimpact

The afore mentioned implementation is called Pimpact. Pimpact is the acronym for **p**eriodic **i**ncompressible Navier–Stokes solver on **m**assively **pa**rallel **c**ompu**t**ers.

To get the code, one has to execute

```
git clone https://github.com/huppd/PINTimpact.git
```

in a shell. This will download the folder `Pimpact` into the folder of execution. This folder has the following structure

```
/PINTimpact
    /run
    /src
        /src_c
        /src_f
        /test
    /XML
```

The `run` folder contains various python scripts, that start multiple jobs and create folder hierarchies for parameter studies or scaling tests. The `src` folder contains three subfolders, `src_c` for the C++ part of the code, `src_f` for the Fortran90 part of the code, and `test` for the unit-tests. The `XML` folder contains various `xml` parameter files. These are used to set up the parameters and problems for the Pimpact solver.

The documentation can be created by executing

```
doxygen Doxyfile
```

in the `PINTimpact` folder. This will generate the documentation in the `doc` folder. The documentation can be read by opening the file `doc/html/index.html` in the preferred browser. Further information about compiling and using the code can be found there.

Bibliography

[1] P. Arbenz, A. Hiltebrand, and D. Obrist, *A Parallel Space-Time Finite Difference Solver for Periodic Solutions of the Shallow-Water Equation*, in Parallel Process. Appl. Math., R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., vol. 7204 of LNCS, Springer Berlin Heidelberg, 2012, pp. 302–312.

[2] P. Arbenz, D. Hupp, and D. Obrist, *A Parallel Solver for the Time-Periodic Navier–Stokes Equations*, in Parallel Process. Appl. Math., R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., vol. 8385 of LNCS, Springer Berlin Heidelberg, 2014, pp. 291–300.

[3] ——, *Comparison of Parallel Time-Periodic Navier-Stokes Solvers*, in Parallel Process. Appl. Math., R. Wyrzykowski, J. Dongarra, E. Deelman, and K. Karczewski, eds., vol. 10777 of LNCS, Cham, 2018, Springer International Publishing, pp. 57–67.

[4] F. Bachinger, U. Langer, and J. Schöberl, *Numerical analysis of nonlinear multiharmonic eddy current problems*, Numer. Math., 100 (2005), pp. 593–616.

[5] F. Bachinger, U. Langer, and J. Schöberl, *Efficient solvers for nonlinear time-periodic eddy current problems*, Comput. Vis. Sci., 9 (2006), pp. 197–207.

[6] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist, *Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems*, Sci. Program., 20 (2012), pp. 241–255.

[7] K. Beck, *Test-driven development : by example*, Addison-Wesley, Boston, 18th ed., 2013.

[8] P. Benedusi, *A parallel multigrid solver for time periodic incompressible Navier–Stokes equations*, master's thesis, Università della Svizzera Italiana, 2015.

[9] P. Benedusi, D. Hupp, P. Arbenz, and R. Krause, *A Parallel Multigrid Solver for Time-Periodic Incompressible Navier–Stokes Equations in 3D*, in Numer. Math. Adv. Appl. ENUMATH 2015, B. and others Karasözen, ed., vol. 112, Springer, 2016, pp. 265–273.

[10] J. Bey and G. Wittum, *Downwind numbering: robust multigrid for convection-diffusion problems*, Appl. Numer. Math., 23 (1997), pp. 177–192.

[11] M. Bolten, D. Moser, and R. Speck, *A multigrid perspective on the parallel full approximation scheme in space and time*, Numer. Linear Algebr. Appl., 24 (2017), p. e2110.

[12] A. Brandt, *Multi-Level Adaptive Solutions to Boundary-Value Problems*, Math. Comput., 31 (1977), p. 333.

[13] A. Brandt and I. Yavneh, *On multigrid solution of high-reynolds incompressible entering flows*, J. Comput. Phys., 101 (1992), pp. 151–164.

[14] P. N. Brown and H. F. Walker, *GMRES On (Nearly) Singular Systems*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 37–51.

[15] A. Brüger, B. Gustafsson, P. Lötstedt, and J. Nilsson, *High order accurate solution of the incompressible Navier–Stokes equations*, J. Comput. Phys., 203 (2005), pp. 49–71.

[16] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral Methods Fundamentals in Single Domains*, Scientific Computation, Springer, Berlin, Heidelberg, 2006.

[17] A. A. J. Christlieb, C. B. C. Macdonald, and B. B. W. Ong, *Parallel high-order integrators*, SIAM J. Sci. Comput., 32 (2010), pp. 818–835.

[18] A. J. Christlieb, R. D. Haynes, and B. W. Ong, *A Parallel Space-Time Algorithm*, SIAM J. Sci. Comput., 34 (2012), pp. C233–C248.

[19] R. Croce, D. Ruprecht, and R. Krause, *Parallel-in-Space-and-Time Simulation of the Three-Dimensional, Unsteady Navier-Stokes Equations for Incompressible Flow*, in Model. Simul. Optim. Complex Process. - HPSC 2012, H. G. Bock, X. P. Hoang, R. Rannacher, Schlöder, and J. P., eds., Springer International Publishing, Cham, 2014, pp. 13–23.

[20] G. Cvijetic, H. Jasak, and V. Vukcevic, *Finite Volume Implementation of the Harmonic Balance Method for Periodic Non-Linear Flows*, in 54th AIAA Aerosp. Sci. Meet., vol. Im, Reston, Virginia, jan 2016, American Institute of Aeronautics and Astronautics, pp. 1–21.

[21] H. De Gersem, H. Vande Sande, and K. Hameyer, *Strong coupled multiharmonic finite element simulation package*, COMPEL - Int. J. Comput. Math. Electr. Electron. Eng., 20 (2001), pp. 535–546.

[22] A. Dimas, B. Mowli, and U. Piomelli, *Large-eddy simulation of subcritical transition in an attachment-line boundary layer*, Comput. Math. with Appl., 46 (2003), pp. 571–589.

[23] A. Eghbal, A. G. Gerber, and E. Aubanel, *Acceleration of unsteady hydrodynamic simulations using the parareal algorithm*, J. Comput. Sci., 19 (2017), pp. 57–76.

[24] H. Elman, V. E. Howle, J. Shadid, R. Shuttleworth, and R. Tuminaro, *Block preconditioners based on approximate commutators*, SIAM J. Sci. Comput., 27 (2006), pp. 1651–1668.

[25] H. Elman, D. Silvester, and A. Wathen, *Finite Elements and Fast Iterative Solvers*, Oxford University Press, 2014.

[26] H. C. Elman and R. S. Tuminaro, *Boundary conditions in approximate commutator preconditioners for the Navier–Stokes equations*, Electron. Trans. Numer. Anal., 35 (2009), pp. 257–280.

[27] M. Emmett and M. L. Minion, *Toward an efficient parallel in time method for partial differential equations*, Commun. Appl. Math. Comput. Sci., 7 (2012), pp. 105–132.

[28] R. D. Falgout, A. Katz, T. V. Kolev, and J. B. Schroder, *Parallel Time Integration with Multigrid Reduction for a Compressible Fluid Dynamics Application*, 2014.

[29] R. D. Falgout, T. A. Manteuffel, B. O'Neill, and J. B. Schroder, *Multigrid Reduction in Time for Nonlinear Parabolic Problems: A Case Study*, SIAM J. Sci. Comput., 39 (2017), pp. S298–S322.

[30] M. P. I. Forum, *MPI: A Message-Passing Interface Standard, Version 3.1*, High Performance Computing Center Stuttgart (HLRS), 2015.

[31] R. W. Freund, *A Transpose-Free Quasi-Minimal Residual Algorithm for Non-Hermitian Linear Systems*, SIAM J. Sci. Comput., 14 (1993), pp. 470–482.

[32] S. Friedhoff, R. D. Falgout, T. V. Kolev, J. B. Schroder, and S. P. MacLachlan, *A Multigrid-in-Time Algorithm for Solving Evolution Equations in Parallel*, in Sixt. Copp. Mt. Conf. Multigrid Methods, 2012.

[33] M. J. Gander, *50 Years of Time Parallel Time Integration*, in Mult. Shoot. Time Domain Decompos. Methods MuS-TDD, T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, eds., Springer International Publishing, Heidelberg, 2015, pp. 69–113.

[34] M. J. Gander, Y.-L. Jiang, B. Song, and H. Zhang, *Analysis of Two Parareal Algorithms for Time-Periodic Problems*, SIAM J. Sci. Comput., 35 (2013), pp. A2393–A2415.

[35] M. J. Gander and M. Neumüller, *Analysis of a New Space-Time Parallel Multigrid Algorithm for Parabolic Problems*, SIAM J. Sci. Comput., 38 (2016), pp. A2173–A2208.

[36] A. Gopinath and A. Jameson, *Time Spectral Method for Periodic Unsteady Computations over Two- and Three- Dimensional Bodies*, 43rd AIAA Aerosp. Sci. Meet. Exhib., (2005), pp. 1–14.

[37] T. Guédeney, A. Gomar, and F. Sicot, *Multi-frequential harmonic balance approach for the computation of unsteadiness in multi-stage turbomachines*, 21ème Congrès Français de Mécanique, (2013), pp. 1–6.

[38] A. Guégan, P. J. Schmid, and P. Huerre, *Spatial optimal disturbances in swept attachment-line boundary layers*, J. Fluid Mech., 603 (2008), pp. 179–188.

[39] W. Hackbusch, *Fast Numerical Solution of Time-Periodic Parabolic Problems by a Multigrid Method*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 198–206.

[40] W. Hackbusch and T. Probst, *Downwind Gauß-Seidel Smoothing for Convection Dominated Problems*, Numer. Linear Algebr. Appl., 4 (1997), pp. 85–102.

[41] K. C. Hall, K. Ekici, J. P. Thomas, and E. H. Dowell, *Harmonic balance methods applied to computational fluid dynamics problems*, Int. J. Comput. Fluid Dyn., 27 (2013), pp. 52–67.

[42] K. C. Hall, J. P. Thomas, and W. S. Clark, *Computation of unsteady nonlinear flows in cascades using a harmonic balance technique*, AIAA J., 40 (2002), pp. 879–886.

[43] T. Hara, T. Naito, and J. Umoto, *Time-periodic finite element method for nonlinear diffusion equations*, IEEE Trans. Magn., 21 (1985), pp. 2261–2264.

[44] R. Henniger, *Direct and large-eddy simulation of particle transport processes in estuarine environments*, doctoral thesis, ETH Zürich, 2011.

[45] R. Henniger, D. Obrist, and L. Kleiser, *High-order accurate solution of the incompressible Navier–Stokes equations on massively parallel computers*, J. Comput. Phys., 229 (2010), pp. 3543–3572.

[46] M. A. Heroux and J. M. Willenbring, *A new overview of the Trilinos project*, Sci. Program., 20 (2012), pp. 83–88.

[47] K. Hiemenz, *Die Grenzschicht an einem in den gleichförmigen Flüssigkeitsstrom eingetauchten geraden Kreiszylinder*, PhD thesis, Universität Göttingen, 1911.

[48] A. Hiltebrand, *Parallel solution of time-periodic problems*, master's thesis, ETH Zürich, 2010.

[49] D. Hupp, *A parallel space-time solver for Navier–Stokes*, master's thesis, ETH Zürich, 2013.

[50] D. Hupp, P. Arbenz, and D. Obrist, *A parallel Navier–Stokes solver using spectral discretisation in time*, Int. J. Comput. Fluid Dyn., 30 (2016), pp. 489–494.

[51] D. Hupp, D. Obrist, and P. Arbenz, *Multigrid preconditioning for time-periodic Navier–Stokes problems*, in Proc. Appl. Math. Mech., vol. 15, 2015, pp. 595–596.

[52] J. Jeong and F. Hussain, *On the identification of a vortex*, J. Fluid Mech., 285 (1995), p. 69.

[53] O. A. Karakashian, *On a Galerkin–Lagrange Multiplier Method for the Stationary Navier–Stokes Equations*, SIAM J. Numer. Anal., 19 (1982), pp. 909–923.

[54] D. Kay, D. Loghin, and A. Wathen, *A Preconditioner for the Steady-State Navier–Stokes Equations*, SIAM J. Sci. Comput., 24 (2002), pp. 237–256.

[55] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, vol. 16, Society for Industrial and Applied Mathematics, Philadelphia, 1995.

[56] M. Kolmbauer and U. Langer, *A Robust Preconditioned MinRes Solver for Time-periodic Eddy Current Problems*, Comput. Methods Appl. Math., 13 (2013), pp. 1–20.

[57] P. K. Kundu, I. M. Cohen, and D. R. Dowling, *Fluid Mechanics*, Elsevier, Boston, 6th ed., 2016.

[58] J. Leffell, *An Overset Time-Spectral Method for Relative Motion*, doctoral thesis, Stanford University, 2014.

[59] J. Leffell, S. Murman, and T. Pulliam, *An Extension of the Time-Spectral Method to Overset Solvers*, in 51st AIAA Aerosp. Sci. Meet. Incl. New Horizons Forum Aerosp. Expo., Reston, Virigina, jan 2013, American Institute of Aeronautics and Astronautics, pp. 1–25.

[60] J. I. Leffell, J. Sitaraman, V. K. Lakshminarayan, and A. M. Wissink, *Towards Efficient Parallel-in-Time Simulation of Periodic Flows*, in 54th AIAA Aerosp. Sci. Meet., no. January, Reston, Virginia, jan 2016, American Institute of Aeronautics and Astronautics, pp. 4–8.

[61] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2007.

[62] S. J. Lighthill, *Acoustic streaming*, J. Sound Vib., 61 (1978), pp. 391–418.

[63] J. Linden, G. Lonsdale, B. Steckel, and K. Stüben, *Multigrid for the steady-state incompressible Navier-Stokes equations: A survey*, in 11th Int. Conf. Numer. Methods Fluid Dyn., vol. 323, Springer Berlin Heidelberg, Berlin, Heidelberg, 1989, pp. 57–68.

[64] J.-l. Lions, Y. Maday, and G. Turinici, *A “parareal” in time discretization of PDE’s*, C. R. Math. Acad. Sci. - Ser. I - Math., 332 (2001), pp. 661–668.

[65] D. J. Mavriplis and Z. Yang, *Time Spectral Method for Periodic and Quasi-Periodic Unsteady Computations on Unstructured Meshes*, Math. Model. Nat. Phenom., 6 (2011), pp. 213–236.

[66] M. Minion, *A hybrid parareal spectral deferred corrections method*, Commun. Appl. Math. Comput. Sci., 5 (2010), pp. 265–301.

[67] M. F. Murphy, G. H. Golub, and A. J. Wathen, *A Note on Preconditioning for Indefinite Linear Systems*, SIAM J. Sci. Comput., 21 (2000), pp. 1969–1972.

[68] M. Neumüller, *Space-Time Methods: Fast Solvers and Applications*, doctoral thesis, Graz University of Technology, 2013.

[69] W. L. M. Nyborg, *Acoustic Streaming*, in Phys. Acoust., vol. 2, ACADEMIC PRESS INC., 1965, pp. 265–331.

[70] D. Obrist, R. Henniger, and P. Arbenz, *Parallelization of the time integration for time-periodic flow problems*, in Proc. Appl. Math. Mech., vol. 10, 2010, pp. 567–568.

[71] D. Obrist, R. Henniger, and L. Kleiser, *Subcritical spatial transition of swept Hiemenz flow*, Int. J. Heat Fluid Flow, 35 (2012), pp. 61–67.

[72] D. Obrist and P. J. Schmid, *On the linear stability of swept attachment-line boundary layer flow. Part 1. Spectrum and asymptotic behaviour*, J. Fluid Mech., 493 (2003), pp. 1–29.

[73] ——, *On the linear stability of swept attachment-line boundary layer flow. Part 2. Non-modal effects and receptivity*, J. Fluid Mech., 493 (2003), pp. 31–58.

[74] M. R. Osborne, *A note on the numerical solution of a periodic parabolic problem*, Numer. Math., 7 (1965), pp. 155–158.

[75] L. Rayleigh, *On the circulation of air observed in Kundt's tubes and some allied acoustical problems*, Philos. Trans. R. Soc. London, 175 (1884), pp. 1–21.

[76] N. Riley, *Steady Streaming*, Annu. Rev. Fluid Mech., 33 (2001), pp. 43–65.

[77] Y. Saad, *A Flexible Inner-Outer Preconditioned GMRES Algorithm*, SIAM J. Sci. Comput., 14 (1993), pp. 461–469.

[78] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, Philadelphia, PA, 2nd ed., 2003.

[79] Y. Saad and M. H. Schultz, *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[80] D. Samaddar, D. Newman, and R. Sánchez, *Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm*, J. Comput. Phys., 229 (2010), pp. 6558–6573.

[81] W. E. Schiesser and G. W. Griffiths, *A Compendium of Partial Differential Equation Models: Method of Lines Analysis with Matlab*, Cambridge University Press, Cambridge, 2009.

[82] D. Silvester, H. Elman, D. Kay, and A. Wathen, *Efficient preconditioning of the linearized Navier–Stokes equations for incompressible flow*, J. Comput. Appl. Math., 128 (2001), pp. 261–279.

[83] S. Sivaloganathan, *The use of local mode analysis in the design and comparison of multigrid methods*, Comput. Phys. Commun., 65 (1991), pp. 246–252.

[84] B. Song and Y.-L. Jiang, *Analysis of a new parareal algorithm based on waveform relaxation method for time-periodic problems*, Numer. Algorithms, 67 (2014), pp. 599–622.

[85] ——, *A new parareal waveform relaxation algorithm for time-periodic problems*, Int. J. Comput. Math., 92 (2015), pp. 377–393.

[86] P. Spalart, *Direct Numerical Study Of Leading-Edge Contamination*, in AGARD, Fluid Dyn. Three-Dimensional Turbul. Shear Flows Transit., no. 438, 1988, p. 13.

[87] R. Speck, D. Ruprecht, R. Krause, M. Emmett, M. Minion, M. Winkel, and P. Gibbon, *A massively space-time parallel N-body solver*, in 2012 Int. Conf. High Perform. Comput. Networking, Storage Anal., Salt Lake City, UT, nov 2012, IEEE, pp. 1–11.

[88] Y. K. Suh and S. Kang, *Acoustic Streaming*, in Encycl. Microfluid. Nanofluidics, Springer US, Boston, MA, 2014, pp. 1–15.

[89] G. I. Taylor and A. E. Green, *Mechanism of the Production of Small Eddies from Large Ones*, Proc. R. Soc. A Math. Phys. Eng. Sci., 158 (1937), pp. 499–521.

[90] G. J. Tee, *An application of p-cyclic matrices, for solving periodic parabolic problems*, Numer. Math., 6 (1964), pp. 142–159.

[91] A. Toselli and O. Widlund, *Domain Decomposition Methods - Algorithms and Theory*, vol. 34 of Springer Series in Computational Mathematics, Springer-Verlag, 2004.

[92] U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, 2001.

[93] W. M. van Rees, A. Leonard, D. I. Pullin, and P. Koumoutsakos, *A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers*, J. Comput. Phys., 230 (2011), pp. 2794–2805.

[94] S. Vanka, *A calculation procedure for three-dimensional steady recirculating flows using multigrid methods*, Comput. Methods Appl. Mech. Eng., 55 (1986), pp. 321–338.

[95] S. Vanka, *Block-implicit multigrid solution of Navier–Stokes equations in primitive variables*, J. Comput. Phys., 65 (1986), pp. 138–158.

[96] S. Yamada and K. Bessho, *Harmonic field calculation by the combination of finite element analysis and harmonic balance method*, Magn. IEEE Trans., 24 (1988), pp. 2588–2590.

[97] I. Yavneh, C. H. Venner, and A. Brandt, *Fast Multigrid Solution of the Advection Problem with Closed Characteristics*, SIAM J. Sci. Comput., 19 (1998), pp. 111–125.

Curriculum Vitae

Personal details

Name	Daniel Hupp
Birth	September 9, 1987 in Wangen im Allgäu
Citizenship	Germany

Education

7.2013	–	5.2018	PhD Studies in Computer Science at ETH Zürich, Switzerland
2.2012	–	6.2013	MSc Studies in Computational Science and Engineering at ETH Zürich, Switzerland
9.2007	–	11.2012	BSc Studies in Computational Science and Engineering at ETH Zürich, Switzerland
9.1998	–	6.2007	High school, Droste-Hülshoff-Gymnasium in Freiburg im Breisgau, Germany

Publications

- D. Hupp, M. Mendoza, I. Bouras, S. Succi, and H. J. Herrmann, *Relativistic lattice Boltzmann method for quark-gluon plasma simulations*, Phys. Rev. D, 84 (2011), p. 125015.
- P. Arbenz, D. Hupp, and D. Obrist, *A Parallel Solver for the Time-Periodic Navier–Stokes Equations*, in Parallel Process. Appl. Math., vol. 8385 of LNCS, Springer Berlin Heidelberg, 2014, pp. 291–300.
- D. Hupp, D. Obrist, and P. Arbenz, *Multigrid preconditioning for time-periodic Navier–Stokes problems*, in Proc. Appl. Math. Mech., vol. 15, 2015, pp. 595–596.
- P. Benedusi, D. Hupp, P. Arbenz, and R. Krause, *A Parallel Multigrid Solver for Time-Periodic Incompressible Navier–Stokes Equations in 3D*, in Numer. Math. Adv. Appl. ENUMATH 2015, vol. 112, Springer, 2016, pp. 265–273
- D. Hupp, P. Arbenz, and D. Obrist, *A parallel Navier–Stokes solver using spectral discretisation in time*, Int. J. Comput. Fluid Dyn., 30 (2016), pp. 489–494.
- P. Arbenz, D. Hupp, and D. Obrist, *Comparison of Parallel Time-Periodic Navier-Stokes Solvers*, in Parallel Process. Appl. Math., vol. 10777 of LNCS, Cham, 2018, Springer International Publishing, pp. 57–67.